

Мы уже умеем применять стили к элементам с помощью разных селекторов. Однако иногда в файле стилей может быть сразу несколько правил для одного и того же элемента. И тогда возникает резонный вопрос: а что из них в итоге сработает?

Для этого существует специфичность селекторов.

**Специфичность** — это правила, по которым браузер применяет те или иные стили для элемента. В тексте будем называть это приоритетом селекторов.

Начнём с метафоры.

Приоритет селекторов можно сравнить с очередью на концерт.

**Идентификаторы (#id)** — это *VIP*-места. Первый ряд, самые классные места в центре. С них удобно смотреть на любимых исполнителей. И есть шанс, что тебя заметят и позовут на сцену.

**Классы и псевдоклассы** — это места в общих рядах, которые имеют средний приоритет. Они не самые дорогие, но и не самые дешёвые. С них многое видно, но хорошенько рассмотреть любимого артиста мешают чьи-то головы и руки.

**Теги и псевдоэлементы** из прошлого юнита — это места в заднем ряду. У них самый низкий приоритет. Оттуда почти ничего не видно, кое-что слышно, но и стоят они дешевле всего.

Есть ещё такой вариант: **комбинации селекторов**. Их можно сравнить с местами в центре, но в общем ряду. Это всё ещё не так удобно, как на *VIP*-местах. Но лучше уж сидеть в центре общего ряда, чем где-нибудь с краю.

В коде это может выглядеть так. У нас есть параграф:

```
<p class="paragraph" id="text">У меня тут немного текста.  
Интересует?</p>
```

Пишем стили:

```
.paragraph {  
    color: red;  
}
```

```
p.paragraph {  
    color: blue;  
}
```

Как думаете, кто победил в этой схватке? Конечно, второй вариант. Ведь в нём комбинация сразу двух селекторов. Однако идентификатор им не победить даже сообща.

Если мы зададим вот такой стиль...

```
#text {  
    color: orange;  
}
```

...то текст станет оранжевым. Без вариантов.

## Упражнение 1

1 point possible (graded)

**Какой будет рамка div-а с классом content?**

```
div.content {  
    border: 1px dashed purple;  
}
```

```
.content {  
    border: 2px solid green;  
}
```

```
div {  
    border: 3px dotted orange;  
}
```

## Упражнение 1

1/1 point (graded)

Какой будет рамка `div`-а с классом `content` ?

```
div.content {  
  border: 1px dashed purple;  
}  
  
.content {  
  border: 2px solid green;  
}  
  
div {  
  border: 3px dotted orange;  
}
```

- ☐ Сплошной, оранжевой, 3 пикселя в ширину
- ☒ Пунктирной, фиолетовой, 1 пиксель в ширину
- ☐ В точку ( `dotted` ), оранжевой, 3 пикселя в ширину
- ☐ Сплошной, зелёной, 2 пикселя в ширину
- ☐ Пунктирной, зелёной, 2 пикселя в ширину



[Show answer](#)

Отправить

## !important

Если мы говорим о приоритетах селекторов, то надо знать и о ключевом слове `!important`.

Представим, что `CSS` – это государство. Тогда `!important` – закон, который никто не может игнорировать. Даже если не согласен с ним.

Ниже увидим, как это работает на деле.

У нас есть один простой `div`. Мы задали ему и `class`, и `id`:

```
<div id="my-div" class="my-class">Это мой div элемент</div>
```

А потом пошли в файл со стилями и написали следующее:

```
#my-div {  
  font-size: 24px;  
}
```

```
.my-class {  
  font-size: 20px;  
}
```

```
div {  
  font-size: 16px!important;  
}
```

В этом примере мы применяем стили сразу по трём селекторам: по идентификатору, классу и тегу. И добавляем к последнему правилу `!important`.

По-хорошему, к элементу должен примениться стиль через `id`. Он перевесит и класс, и тег. Но `!important` смешал все карты. И теперь размер текста — 16px. Всё, что ниже и выше — уже не важно. Переопределить `!important` может только другой `!important`.

На реальном проекте это может стать большой проблемой. Например, предыдущий код писали не вы. Стилей очень много. И где-то там среди них одному из элементов задан стиль с `!important`.

Придется искать его, убирать, а потом чинить то, что его исчезновение сломало. И только потом уже переписывать нужные стили. Это отнимет много сил, времени и нервов. Лучше просто не использовать `!important`.

И ограничиться знанием, что он существует. Где-то там. Но не в вашем проекте.

## Приоритет `inline`-стилей

Есть ещё один нюанс. Вы помните, что стили можно задавать не только через внешний файл, но и внутри разметки. А ещё помните, что есть *inline*-стили. Это когда мы пишем стили прямо в тег в атрибуте `style`.

Примерно так:

```
<span style="color: red">Всем привет!</span>
```

Если вы зададите стили вот здесь, в атрибуте, то они будут иметь больше веса, чем самый приоритетный идентификатор, классы и даже комбинации классов. Но не `!important`. Это – самый главный босс.

Как и `!important`, *inline*-стилей лучше избегать. Их будет очень сложно переопределять, если проект разрастётся. И другие разработчики тоже спасибо не скажут.

Итак, занесём в рамочку:

**При вычислении приоритета учитываются следующие веса селекторов:**

- **Inline-стили** – наивысший приоритет.
- **Идентификаторы** (`#id`) – высокий приоритет.
- **Классы** (`.class`) и псевдоклассы (например, `:hover`) — средний приоритет.
- **Теги** (`div`, `p`, `h1` и т.д.) и **псевдоэлементы** (например, `::before`) — наименьший приоритет.
- `+`, `>`, `~` и `*` – не добавляют веса селекторам.

Но что будет, если стиль определён несколько раз для одного и того же элемента?

В этом случае будет применено последнее правило.

У нас есть такой код:

```
<div class="example">Пример</div>
```

И два блока стилей:

```
.example {  
  color: red;  
  font-size: 20px;  
}
```

```
.example {  
  color: blue;  
  font-size: 16px;
```

}

Для `div`-а с классом `example` будет применён только последний блок стилей. Цвет текста будет синим, а размер шрифта — 16px. Первый блок стилей будет проигнорирован.

## Упражнение 2

Попрактикуйте использование `!important`.

Попробуйте задать другой цвет и другой размер для элемента. Но не меняйте старые стили и не удаляйте `!important`.

## Best practice

Слишком много нюансов? Да, не все из них получится запомнить с первого раза. Поэтому давайте перечислим самые главные правила при работе с CSS-селекторами:

- **Используйте преимущественно селекторы классов.** Даже если речь идёт о заголовке или другом уникальном элементе.
- **Не применяйте к одному элементу чрезмерное количество классов.**
- **Избегайте встроенных стилей**, таких как `<a style="color: red;">`.
- **Не рассчитывайте на модификатор `!important`.** Лучше, если вы просто будете знать о его существовании, не более.

Классы — лучший способ не увязнуть в приоритетах и специфичности. Классы – это классно.

Но если нужно посчитать специфичность селекторов, то есть онлайн-калькуляторы: [Specificity Calculator](#) и, например, [CodeCaptain](#). Они основаны на подсчете баллов селекторов.

Посмотрите на эту картинку:

тег элемента

style=""	→	1,0,0,0
#id	→	0,1,0,0
.class	→	0,0,1,0
[attr]=value	→	0,0,1,0
li (по тегу)	→	0,0,0,1
*	→	0,0,0,0

По этой методике приоритет селекторов считается в баллах:

- Каждый *inline*-стиль – это 1000 баллов;
- Каждый идентификатор добавляет 100 баллов;
- Каждый класс, селектор по атрибуту или псевдокласс добавляет 10 баллов;
- Каждый тег добавляет 1 балл.

В случае комбинаций баллы складываются. Селекторы с большим количеством баллов имеют более высокий приоритет. Всё просто.

### Упражнение 3

1/1 point (graded)

Какой из этих селекторов имеет наивысший приоритет в CSS?

☒ #nav ul li a:hover

☐ #nav a

☐ .nav a:hover

☐ body div a



Show answer

Отправить

✓ Верно (1/1 балл)

### Упражнение 4

1/1 point (graded)

Как переопределить *inline*-стили?

☐ С помощью идентификатора

☒ Применить к новым стилям `!important`

☐ С помощью класса

☐ Никак, это невозможно



Show answer

Отправить

✓ Верно (1/1 балл)



В предыдущих юнитах, чтобы задать цвета, мы использовали ключевые слова.

*Red* — делал текст красным, *blue* — синим, *purple* — фиолетовым. Это самые базовые цвета. Но есть список, в котором ключевых слов гораздо больше.

Посмотреть их можно [здесь](#). Там вы сможете найти редкие цвета, которых мы даже не касались. Возможно, о некоторых из них вы даже не подозревали.

Плюс ключевых слов — **нечувствительность к регистру**. *LIGHTBLUE* и *lightblue* — один и тот же светло-голубой цвет.

Ключевые слова — это круто и быстро. Но в реальных проектах они почти не используются. Как же тогда задавать цвета?



## ***RGB и RGBA***

Чаще всего разработчики задают цвета с помощью *RGB* или *RGBA*. Давайте начнём с того, что это такое.

***RGB*** — это цветовая модель.

Она основана на комбинации красного (*R*), зелёного (*G*) и синего (*B*) цветов. Каждый цвет может принимать значения от 0 до 255.

Например, чтобы задать красный цвет, зададим ему значение `rgb(255, 0, 0)`. Зелёный цвет будет выглядеть вот так: `rgb(0, 255, 0)`.

## **Упражнение 1**

Задайте таким же образом синий цвет. Можете сделать это в своём редакторе кода или [здесь](#).

## **Но вот вопрос: откуда берётся число 255?**

*RGB* использует 8-битное представление для каждого цвета. Оно может использовать 2 в 8-ой степени (или 256) различных оттенков для каждого цвета. Поэтому максимальное значение для каждого цвета — 255, включая 0.

*RGBA* — это расширение *RGB*. Помимо красного, зелёного и синего цветов оно позволяет выбрать прозрачность. Именно прозрачность добавляет букву *A*.

***A* — *alpha*-канал.** И принимает значения от 0 до 1.

Например, зададим полупрозрачный красный цвет: `rgba(255, 0, 0, 0.5)`.

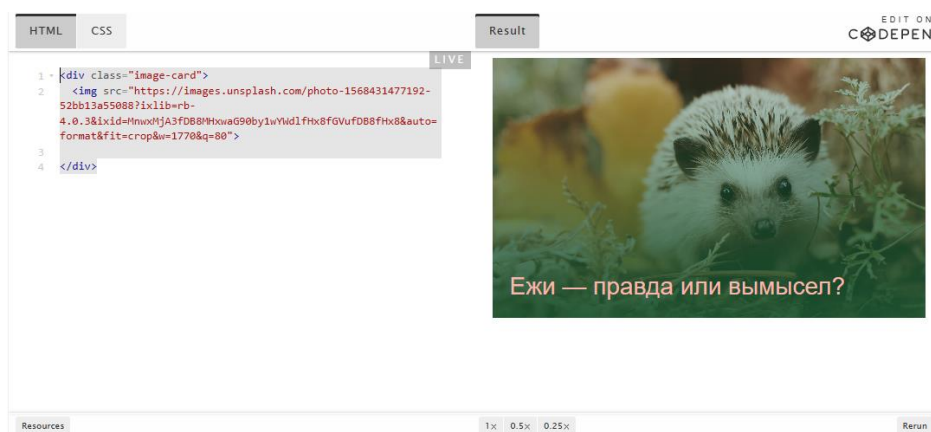
0.5 — отвечает за прозрачность текста. Чтобы сделать его ещё прозрачнее, можно уменьшить значение до 0.3. А чтобы уплотнить цвет — повысить его до 0.7.

Вы можете использовать краткую запись без нуля и не прописывать 0.7.

**Короткий вариант** — .7

*RGB* и *RGBA* удобнее ключевых слов. Они дают большую гибкость и точность в выборе цветов как разработчику, так и дизайнеру.

Кроме того, с *RGB*-цветами легче делать градиенты и плавные анимации. Помните нашу картинку с ежом из позапрошлого юнита? Давайте зададим ему градиент поинтереснее.



```

1  body {
2      font-family: helvetica;
3      font-weight: 500;
4      font-size: 28px;
5  }
6
7  .image-card {
8      position: relative;
9      margin: 0 auto;
10     width: 500px;
11     height: 300px;
12 }
13
14  img {
15     width: 100%;
16     height: 100%;
17     object-fit: cover;
18 }
19
20  .image-card::after {
21     content: "";
22     position: absolute;
23     top: 0;
24     right: 0;
25     bottom: 0;
26     left: 0;
27     background: linear-gradient(to bottom, rgba(131, 222, 151, .1), rgba(27, 87, 50, .9));
28 }
29
30  .image-card::before {
31     content: "Ежи — правда или вымысел?";
32     position: absolute;
33     bottom: 20px;
34     left: 20px;
35     color: rgb(254, 183, 174);
36     z-index: 1;
37 }
38

```

С помощью *RGBA*-цветов мы сделали на картинке градиент от светло-зелёного до глубокого зелёного цвета. Ключевые слова не смогли бы дать такую гибкость.

Кроме градиента мы изменили цвет текста и использовали для этого просто *RGB*-цвет, без прозрачности. Она здесь ни к чему.

Использовать *RGB* и *RGBA* сложнее, чем ключевые слова. Особенно для новичков, у которых почти нет опыта в выборе цветов.

Но на каждую сложную задачу где-то существует сервис, чтобы её решить. Поэтому вы можете выбирать и копировать цвета [здесь](#).

Не думайте, что использовать такие сервисы — это читерство и непрофессионализм. Для предыдущего примера я выбирала цвета именно в нём. 😊

## Упражнение 2

Поменяйте в предыдущем примере цвета на те, которые вам нравятся. Работайте с платформами, на которых можно искать и копировать *RGB*-коды цветов. Не используйте ключевые слова.

### *HEX*

*HEX*-коды — это ещё один способ задать цвет в *CSS*. Каждый цвет — это шестизначный код. Две первые цифры отвечают за красный цвет, вторые две — за зелёный, а две последние — за синий.

*HEX* представляет цвет в виде **шестнадцатеричного кода**, из 3 или 6 символов. Каждый символ — один байт (8 бит), от 0 до 9 и от A до F. Можно обозначить шаблон *HEX*-кода так: #RRGGBB.

Например, ярко-желтый в *HEX* будет выглядеть так: #FFFF00.

Здесь FF — максимальное значение красного и зелёного, а 00 — минимальное значение синего. В формате *RGB* мы бы представили его вот так: `rgb(255, 255, 0)`.

*HEX*-коды удобно использовать в *CSS*. Но вот читать их не всегда просто. А чтобы выставить прозрачность, как в *RGBA*, нужно добавить ещё два символа в конце. Итого — восемь.

Полупрозрачный жёлтый можно записать вот так: #FFFF0080.

Это совсем не интуитивно, в отличие от *RGBA*. Поэтому не очень часто встречается на реальных проектах.

## Упражнение 3

Вернитесь в *codepen*, в котором вы работали и примените новые знания. Теперь у вас есть ещё один способ задать те же цвета, только с помощью *HEX*-кода.

### *HSL* и *HSLA*

Есть и четвёртый способ, чтобы задать цвет в *CSS* – *HSL*.

***HSL*** — это тоже цветовая модель. Но она расшифровывается как тон (*Hue*), насыщенность (*Saturation*) и яркость (*Lightness*). *HSL* задаётся тремя числами: от 0 до 360 для цвета, от 0% до 100% для насыщенности и от 0% до 100% яркости.

**С 250 все понятно, а откуда 360?**



Цвета в *HSL* берутся из цветового круга. А в круге — 360 градусов. Для каждого градуса — свой тон. И свои оттенки яркости и насыщенности.

Зададим ярко-жёлтый цвет, но уже с помощью *HSL*. Жёлтому соответствует тон в 60 градусов. По картинке это можно определить визуально.

Насыщенность нам нужна максимальная, в 100%, а яркость — только в 50%. Иначе это уже будет не жёлтый, а белый.

Если любому оттенку в *HSL* задать 100% яркости, он станет **белым**. А при 0% — оттенок станет **черным**.

Запишем: `hsl(60, 100%, 50%)`.

Можно ли задать с помощью этой формулы прозрачность? Можно. Для этого есть *HSLA*. Как и в *RGBA*, А здесь — это *alpha*-канал. И он принимает значения от 0 до 1.

Снова попробуем задать полупрозрачный ярко-жёлтый цвет, но уже с помощью *HSLA*. Получится вот так: `hsla(60, 100%, 50%, 0.5)`.

Но если есть *RGB* и *HEX*, то зачем нужен ещё и *HSL*?

*HSL* понятнее человеческому глазу. Он не заставляет высчитывать биты от 0 до 255 и подстраиваться под уровни зелёного, красного и синего.

**У *HSL* интуитивные параметры:** яркость, тон, насыщенность. Ими легко управлять и менять цвет в ту сторону, которая вам нужна.

Нужно поярче? Увеличьте яркость. Понасыщенней? Измените насыщенность. И всё.

#### Упражнение 4

1/1 point (graded)

Какой *HSL*-код задает тот же цвет, что и `rgb(81, 255, 0)` ?

☐ `hsl(114, 100%, 23%)`

☐ `hsl(114, 100%, 75%)`

☒ `hsl(101, 100%, 50%)`

☐ `hsl(114, 84%, 16%)`



[Show answer](#)

Отправить

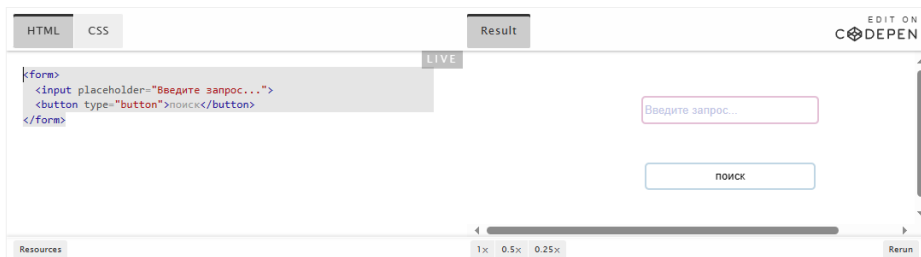
✓ Верно (1/1 балл)

## Transparent

Мы поговорили о цветах и о прозрачности. Но есть свойство, которое и без *RGB*, *HEX* и *HSL* делает элемент прозрачным. И это `transparent`.

Это значение часто используют при создании кнопок.

Посмотрим на небольшой пример:



```

1  form {
2    display: flex;
3    flex-flow: column;
4    align-items: center;
5    justify-content: space-evenly;
6    width: 600px;
7    height: 200px;
8    margin: 0 auto;
9  }
10
11  input {
12    width: 200px;
13    height: 2em;
14    border: 2px solid #e4c0d6;
15    border-radius: 5px;
16  }
17
18  input::placeholder {
19    color: #c0c4e4;
20  }
21
22  button {
23    width: 200px;
24    height: 2rem;
25    background-color: transparent;
26    border-radius: 5px;
27    border: 2px solid #c0d6e4;
28    cursor: pointer;
29  }
30
31  button:hover {
32    background-color: #c0c4e4;
33  }

```

Тут кнопка изначально имеет прозрачный фон. Если на неё навести мышкой — фон станет лиловым. Такую анимацию можно встретить на каждом втором сайте. Это просто, но эффективно.

Также обратите внимание, как в примере использованы инструменты из прошлых юнитов: псевдоклассы, псевдоэлементы и относительные единицы измерения. Не забывайте про них.

**Для работы с цветами есть очень много инструментов и платформ. От себя могу посоветовать вот эти:**

- [HSL picker](#) — ресурс для подбора цветов в кодировке *HSL*.
- [Color-hex](#) — платформа с *HEX*-цветами.
- [HTML color codes](#) — ещё один сайт для подбора цветов в разных кодировках.

А пока что предлагаю поиграться с цветами прямо здесь, в юните.

## Упражнение 5

1/1 point (graded)

Выберите все варианты, с помощью которых можно задать элементу фон цвета `rgb(255, 255, 0)` :

☒ `background-color: yellow`

☐ `background-color: #FFD700`

☐ `color: hsl(60, 100%, 50%)`

☒ `background-color: #FFFF00`

☒ `background-color: hsl(60, 100%, 50%)`



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

## Упражнение 6

Напишите стиль для заголовка второго уровня с идентификатором `heading` так, чтобы он стал розового цвета (равен `#ff00e1` по HEX). Используйте только HSL-код.

```
#heading {  
  color: ____;  
}
```

[Посмотреть ответ](#)

```
#heading {  
  color: hsl(307, 100%, 50%);  
}
```

Любая страница состоит из блоков. И у каждого блока есть размер: ширина и высота.

Мы привыкли к этим величинам и в жизни. У стола есть высота. И от неё зависит, удобно ли за ним сидеть.

У посылки, которую мы отправим другу, есть ширина, длина и высота. И ещё цена за коробку, если эти габариты больше, чем хотелось бы.

Привычные величины. Но веб-элементы измеряются не в сантиметрах, как коробки и столы. У них для этого есть свои единицы.

Давайте разберём их подробнее.

## Пиксели (px)

Пиксели — это самая популярная единица измерения в CSS. И **абсолютная**. Потому что она зависит от пикселей в нашем экране.



**Абсолютными** называют единицы измерения, которые обозначают физические размеры.

Кроме пикселей есть еще *cm* (сантиметры), *mm* (миллиметры), *in* (дюймы) и *pt* (пункты). Они перекочевали в вёрстку из типографии.

Эти единицы редко встретишь на проекте. Для адаптивной верстки они не годятся из-за **негибкости**.

Раньше они были полезны при выводе документа на печать. Но теперь, с появлением экранов разной плотности, не могут работать корректно.

Вернёмся к пикселям. Пиксели в большинстве случаев стабильны. И не меняются, если разрешение экрана стало больше или меньше.

Вот так, с помощью пикселей, мы нарисовали три симпатичных кубика. У каждого своя ширина и своя высота. Блоки будут выглядеть одинаково и на ноутбуке, и на планшете.



```
1 .wrap {
2   width: 100%;
3   margin: 0 auto;
4   display: flex;
5   justify-content: center;
6 }
7 .block-1 {
8   width: 200px;
9   height: 200px;
10  background-color: #feb7ae;
11 }
12
13 .block-2 {
14   width: 150px;
15   height: 150px;
16   background-color: #aefeb7;
17 }
18
19 .block-3 {
20   width: 100px;
21   height: 100px;
22   background-color: #b7aeef;
23 }
24
```

## Упражнение 1

Сейчас кубики в примере расположены по убыванию: от большого к маленькому. Попробуйте поменять порядок. Так, чтобы сначала шёл самый

маленький, потом средний, а в конце – самый большой. Сохраните размеры, просто поменяйте их местами.

```
1  .wrap {
2    width: 100%;
3    margin: 0 auto;
4    display: flex;
5    justify-content: center;
6  }
7  .block-3 {
8    width: 200px;
9    height: 200px;
10   background-color: #feb7ae;
11 }
12
13 .block-2 {
14   width: 150px;
15   height: 150px;
16   background-color: #aefeb7;
17 }
18
19 .block-1 {
20   width: 100px;
21   height: 100px;
22   background-color: #b7aeef;
23 }
24
```

## Проценты (%)

Но есть нюанс. Наши кубики заняли ширины на 450 пикселей. Это больше, чем экран смартфона. Если экран будет меньше самих элементов, вёрстка сломается. Этого допустить нельзя.

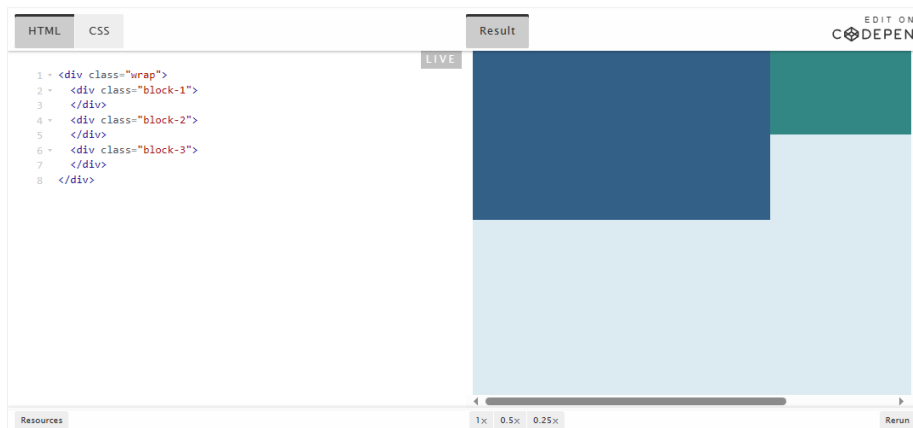
Поэтому разработчики придумали задавать размеры не пикселями, а процентами. Это более гибкий и эффективный вариант.

Пришла пора поговорить про **относительные** единицы измерения.

**Относительные** единицы зависят от размера родителя и изменяются вместе с ним.

Проценты, в отличие от пикселей, зависят от размера родительского элемента. И если меняются величины родителя, то величины потомка тоже изменятся.

Смотрим:



```
1  .wrap {
2    width: 700px;
3    height: 500px;
4    margin: 0 auto;
5    display: flex;
6    justify-content: center;
7    background-color: #dcebf2;
8  }
9  .block-1 {
10   width: 50%;
11   height: 50%;
12   background-color: #326087;
13 }
14
15 .block-2 {
16   width: 30%;
17   height: 30%;
18   background-color: #328784;
19 }
20
21 .block-3 {
22   width: 20%;
23   height: 20%;
24   background-color: #873260;
25 }
26
```

Зададим родителю фон, чтобы было нагляднее.

Посмотрите на блоки. У них и ширина, и высота равны друг другу. 50% и 50%, 30% и 30%. Но из-за того, что высота и ширина родителя — разные, то и в процентах это получаются разные числа. 50% от 700 — 350 пикселей. А 50% от 500 — 250 пикселей.

Поэтому наши кубики стали прямоугольниками. И если теперь изменить ширину родителя, то изменятся и их ширины. Поэкспериментируйте с этим примером и посмотрите, как размеры влияют друг на друга.

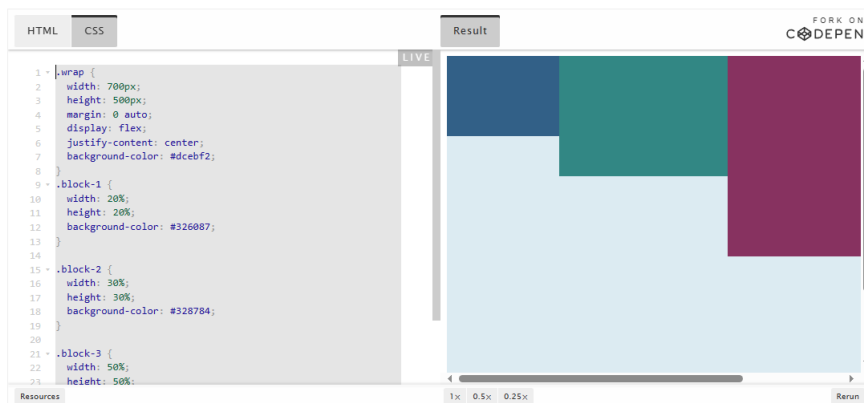
## **Упражнение 2**

Сделайте то же самое, что и в примере с пикселями. Поменяйте порядок по убыванию на порядок по возрастанию. И в соответствии с ним задайте прямоугольникам нужные размеры.

```

1  .wrap {
2      width: 700px;
3      height: 500px;
4      margin: 0 auto;
5      display: flex;
6      justify-content: center;
7      background-color: #dceb2;
8  }
9  .block-1 {
10     width: 20%;
11     height: 20%;
12     background-color: #326087;
13 }
14
15 .block-2 {
16     width: 30%;
17     height: 30%;
18     background-color: #328784;
19 }
20
21 .block-3 {
22     width: 50%;
23     height: 50%;
24     background-color: #873260;
25 }
26

```



## Em и Rem

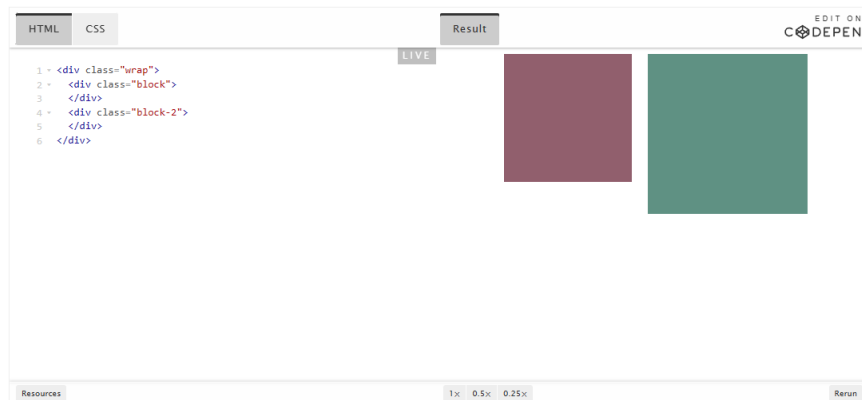
Пиксели и проценты встречаются на проектах чаще всего. Но есть и другие единицы измерения. Например, em и rem.

**Em** — зависит от размера шрифта родительского элемента.

**Rem** — зависит от размера шрифта корневого элемента (html).

Звучит не просто. Но с помощью примера станет понятнее.

Звучит не просто. Но с помощью примера станет понятнее.



```
1  html {
2    /* шрифт корневого элемента */
3    font-size: 20px;
4  }
5
6  .wrap {
7    margin: 0 auto;
8    display: flex;
9    gap: 20px;
10   justify-content: center;
11   /* шрифт родительского элемента */
12   font-size: 16px;
13 }
14
15 .block {
16   /* 10 * 16 = 160px */
17   width: 10em;
18   height: 10em;
19   background-color: #915f6d;
20 }
21
22 .block-2 {
23   /* 10 * 20 = 200px */
24   width: 10rem;
25   height: 10rem;
26   background-color: #5f9183;
27 }
28
```

У нас есть два блока, у них один и тот же родитель. И почти одинаковые размеры. Но всё же правый кубик чуть больше левого. Почему так?

Ширина и длина левого элемента зависят от размера шрифта родителя. Этот шрифт равен 16px. Значит, сторона кубика равна 160px.

А вот у правого кубика сторона больше. Потому что она зависит не от родительского шрифта, а от корневого шрифта. Шрифт документа — 20px. Отсюда и разница в габаритах.

## Упражнение 3

Обратитесь к этому примеру и поправьте код так, чтобы правый кубик стал меньше левого. Единицы измерения не меняйте.

Если левый кубик измеряется в em, то так и должно оставаться. Не важно, насколько меньше будет правый блок — используйте любые числа.

```
1  html {
2    /* шрифт корневого элемента */
3    font-size: 20px;
4  }
5
6  .wrap {
7    margin: 0 auto;
8    display: flex;
9    gap: 20px;
10   justify-content: center;
11   /* шрифт родительского элемента */
12   font-size: 16px;
13 }
14
15 .block {
16   /* 10 * 16 = 160px */
17   width: 10em;
18   height: 10em;
19   background-color: #915f6d;
20 }
21
22 .block-2 {
23   /* 10 * 20 = 200px */
24   width: 5rem;
25   height: 5rem;
26   background-color: #5f9183;
27 }
28
```

## Vw, vh, vmin, vmax

Если вы думали, что это всё — нет, это не всё. 😊



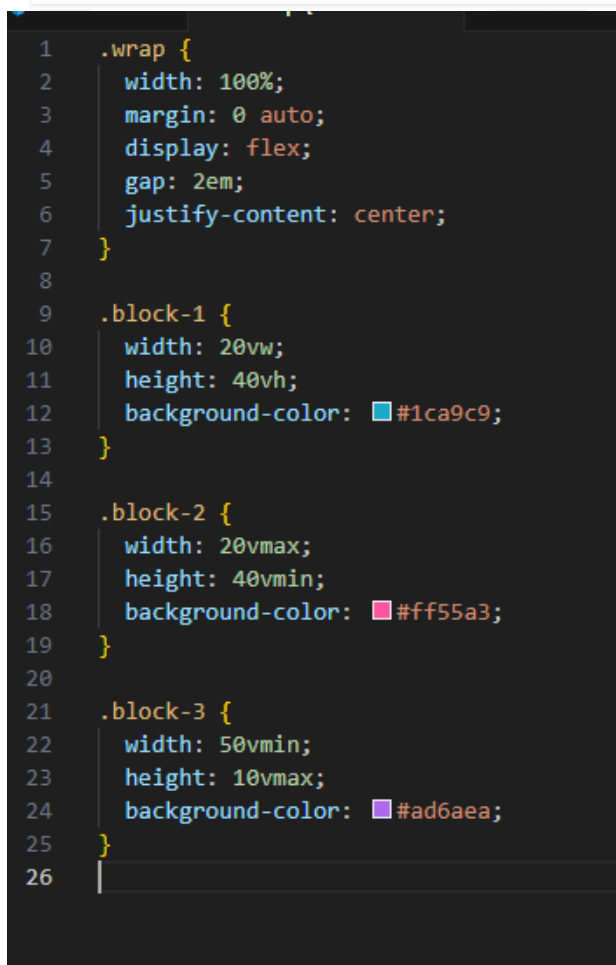
Для гибкой адаптивной вёрстки недостаточно процентов или `em/rem`. В арсенале разработчиков есть *viewport*-единицы.

Они зависят от размера видимой области браузера — или вьюпорта. Отсюда и название.

С помощью них можно масштабировать свой проект на разных устройствах: смартфонах, планшетах, ноутбуках и больших экранах.

### **Пройдёмся по каждой:**

- `vw` (от *viewport width*) — 1% от ширины видимой области браузера. Если у вас ширина экрана — 1440px, то 50vw от неё — это 720px. Это как с процентами.
- `vh` (*viewport height*) — то же самое, только от высоты видимой области браузера.
- `vmin` (*viewport minimum*) — 1% от минимального значения между шириной и высотой видимой области браузера. Если высота меньше ширины — вот от неё размер и будет отсчитываться. И наоборот.
- `vmax` (*viewport maximum*) — то же самое, только если ширина больше высоты, от неё и отсчитываются размеры.



Новая порция кубиков. Обратите внимания на первые два. В примере с процентами одна и та же запись давала разный результат. Тут ровно наоборот.

Их размеры идентичны. Но в коде написано разное. У первого кубика ширина — `20vw`. То есть 20% от ширины экрана. У второго — `20vmax` — 20% от той величины вьюпорта, которая больше. Странно?

Нет, никакого противоречия здесь нет. Если вы смотрите этот пример с ноутбука или компьютера, то ширина экрана у вас больше его высоты. А значит `vw` и `vm` — это одно и то же.

То же самое происходит и с их высотами. `Vh` и `vm` — это разные записи одной и той же величины. Поэтому кубики одинакового размера.

С третьим кубиком всё тоже просто. Его ширина зависит от высоты, так как она меньше. А высота — наоборот, от ширины. Потому что она больше. Не перепутайте. 😊

## Упражнение 4

У вас в примере уже есть два одинаковых блока. Сделайте так, чтобы третий стал таким же. Его ширина и высота должны быть равны ширине и высоте первых двух элементов.

## Ch, ex и fr

Если вы и в этот раз думали, что это всё, то снова не угадали.

Есть ещё несколько относительных единиц измерения, которые используются реже, но знать о них всё равно стоит.

**ch** — это единица, которая равна ширине одного символа моноширинного шрифта.

Разберём на примере. Если установить ширину поля ввода в `20ch`, то оно будет вмещать ровно 20 символов, независимо от их размера.

Запишем это так:

```
input {  
  width: 20ch;
```

}

ex равна высоте символа «x» текущего шрифта. Например, если установить высоту строки в 3ex, то она будет равна трём высотам символа «x».

Пример:

```
p {  
  font-size: 16px;  
  line-height: 3ex;  
}
```

В этом примере, высоту строки можно посчитать так: 3 \* высота символа x /\* при размере в 16px \*/.

Эта теория — не призыв к действию. Просто знайте, что такие единицы существуют. Возможно, когда-нибудь они окажутся нужным в проекте инструментом.

Теперь вы знаете основные единицы измерения в CSS и можете использовать их для создания адаптивного дизайна.

### Упражнение 5

1/1 point (graded)

Какие абсолютные единицы чаще всего используются в веб-разработке?

☐ Сантиметры

☐ Проценты

☐ Миллиметры

☒ Пиксели

☐ Дюймы



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

### Упражнение 6

1/1 point (graded)

Нужна гибкая верстка, которая будет хорошо смотреться на разных устройствах. Как это сделать?

☒ Использовать относительные единицы измерения. Они более гибкие, потому что меняются, если изменяется разрешение экрана.

☐ Абсолютные, потому что они надежные.

☐ Использовать только пиксели и ничего кроме них.



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

Мы уже говорили про коробки, кубики и блоки. Это очень важные образы для веб-разработки. Чтобы понять, как работает блочная модель документа — обратимся к жизни.

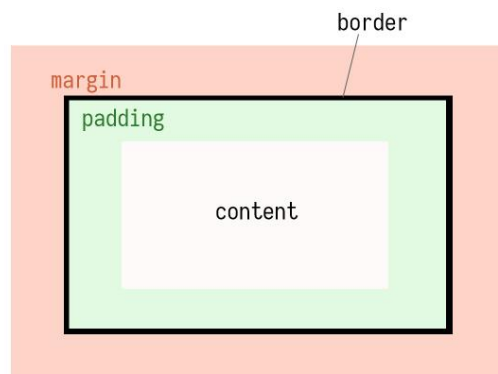
Блочный элемент действительно похож на коробку. У него есть границы, что-то внутри и что-то снаружи. В элемент можно что-то положить. Можно подвинуть его левее или правее. Или найти коробку побольше и положить его туда.

Или вот ещё метафора: вёрстка похожа на строительство. А блочные элементы — на кирпичи. Можно одни поставить рядышком, а другие друг от друга отодвинуть.

Теперь у нас есть образ в голове. Начинаем погружаться в то, как устроены виртуальные блоки.

У любого *HTML*-блока есть следующие свойства:

- **Content** — содержимое. Текст, картинки, другие блоки и так далее.
- **Ширина и высота**. Про них мы уже очень много знаем и можем задать с помощью разных единиц измерения.
- **Граница** или **border** — это линия, которая отделяет блок от других блоков. Как стенки у коробки.
- **Внутренние отступы** или **padding** — то, что отодвигает блок от его родителя.
- **Внешние отступы** или **margin** — то, что отодвигает блок снаружи от других блоков-соседей.

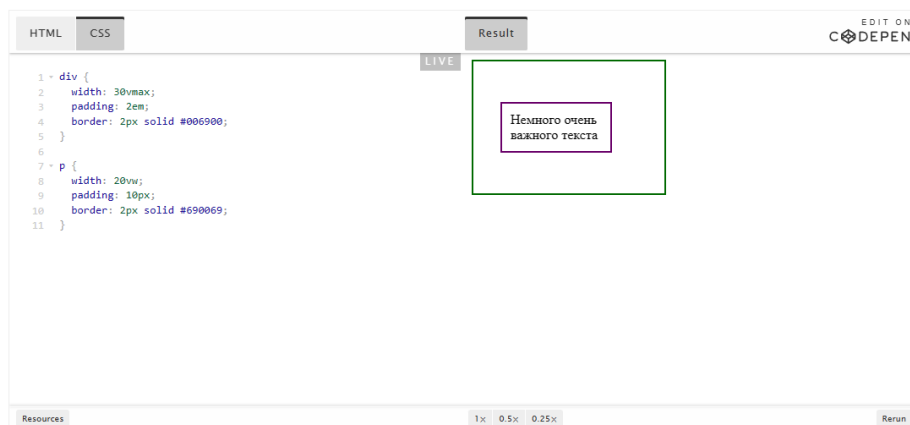
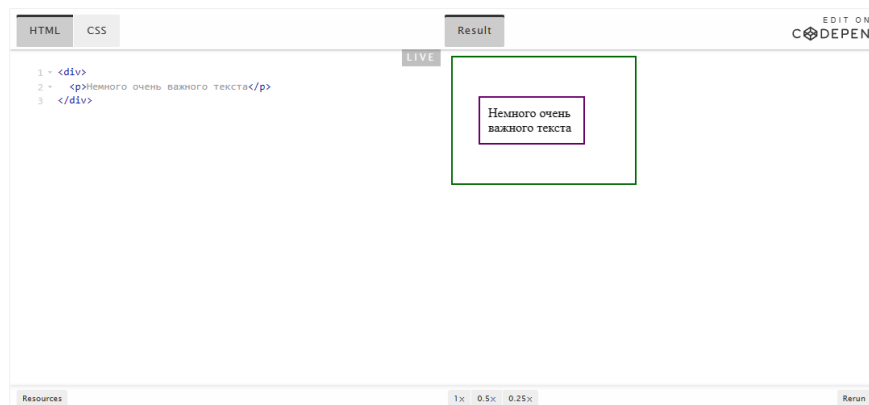


Посмотрим на картинку. Это универсальная модель блока. Он существует именно в таком порядке. Поэтому очень важно не путать местами отступы.

Не задавать `padding`-и там, где нужны `margin`-ы. И наоборот.

## Padding

Разберём отступы подробнее



`padding` задаёт расстояние между границей блока и контентом внутри. Представьте свою комнату. У вас есть стол, за которым вы сейчас сидите. Между ним и стеной наверняка есть какое-то свободное пространство.

Этот отступ — и есть `padding`. Отступ от родителя-комнаты до стола-содержимого.

Итак, время примеров.

Здесь у нас сразу два блочных элемента: `div` и `p`. У каждого есть границы, у каждого есть внутреннее содержимое. А также `padding`-и.

Параграф с текстом — родитель для текста. Поэтому, чтобы отодвинуть текст от границ блока, мы используем внутренние отступы — `padding`-и.

Но в то же время этот параграф — потомок `div`-а. И поэтому, чтобы отодвинуть его от границ родителя, мы не задаём ему никаких внешних отступов. Правильно будет задать внутренние отступы самому `div`-у.

Получается такая матрёшка.

## Упражнение 1

Сейчас в примере внутренние отступы есть и у `div`-а, и у параграфа с текстом. Ваша задача — оттолкнуть сам `div` от границ экрана. Для этого задайте нужные стили для `body` самого документа. Используйте любые числа и любые единицы измерения.

## Margin

А теперь — к внешним отступам. Вы все ещё за своим столом, читаете юнит. А за окном — другие дома. И до самого близкого из них можно дойти за 2 минуты. Вот это расстояние от дома до дома — `margin`. Потому что оба дома — соседи.

Теперь к примеру:





```
1  .wrap {
2    display: flex;
3    margin: 0 auto;
4    width: 500px;
5    min-height: 200px;
6    padding: 1em;
7    border: 3px solid hsl(80, 50%, 30%)
8  }
9  .house {
10   display: flex;
11   justify-content: center;
12   align-items: center;
13   width: 50%;
14   background-color: #c0d6e4;
15   margin-right: 2em;
16 }
17
18 .another-house {
19   display: flex;
20   justify-content: center;
21   align-items: center;
22   width: 50%;
23   background-color: #acc0cd;
24 }
```

Перенесём метафору с домами в код. У нас есть родитель и в нём два «дома». Один отодвинут от другого с помощью `margin-a`. Потому что они друг для друга соседи.

Но вот отступ от внешних границ родителя мы задали через `padding`. В этом вся разница. Выбор отступов зависит от того, в каких «отношениях» между собой состоят блоки.

## Упражнение 2

Поработайте с примером. Сейчас у нас есть один большой блок с классом `wrap`, а в нём — два потомка. Не трогайте то, что внутри блока `wrap`. Создайте ему внешнего соседа и отодвиньте эти блоки друг от друга. Воспользуйтесь внешними отступами.

## Пример с центрированием

Давайте рассмотрим один очень **полезный лайфхак** с использованием `margin`.

Часто на собеседованиях спрашивают: как бы вы поместили блок ровно по центру? И можно ответить так: с помощью `margin`.

Вернёмся к примеру с домами. Вы наверняка заметили, что блок находится ровно по центру. Это возможно благодаря двум свойствам:

```
width: 500px;  
margin: 0 auto;
```

Итак, для работы с этим способом обязательно нужно **задать блоку ширину**. Можно через пиксели, можно через проценты. Но она нужна.

После этого **задаём внешние отступы**. Сверху и снизу они нам не нужны. А вот по бокам — задаём `auto`. Благодаря этому трюку браузер выровняет наш блок прямо по центру.

## Синтаксис

Напоследок, давайте разберёмся, как можно кратко записывать внешние и внутренние отступы в коде. Такая запись подойдёт, если все отступы одинаковы:

```
margin: 20px;  
padding: 2em;
```

А вот как поступить, если со всех сторон размеры отступов отличаются:

```
margin-top: 10px;  
margin-right: 30px;  
margin-bottom: 20px;  
margin-left: 15px;
```

```
/* или короткая запись */
```

```
margin: 10px 30px 20px 15px; /* верх право низ лево */
```

```
/* если левый и правый отступы - равны */
```

```
margin: 20px 40px 30px;
```

```
/* если верхний и нижний отступы равны, а также левый и правый  
равны */
```

```
margin: 0 20px;
```

```
padding-top: 10px;  
padding-right: 30px;  
padding-bottom: 20px;  
padding-left: 15px;
```

```
/* или короткая запись */
```

```
padding: 10px 30px 20px 15px; /* верх право низ лево */
```

```
/* если левый и правый отступы - равны */
```

```
padding: 20px 40px 30px;
```

```
/* если верхний и нижний отступы равны, а также левый и правый  
равны */
```

```
padding: 0 20px;
```

Чтобы запомнить очерёдность, попробуйте смотреть на это как на часы. С 12 до 12 — по часовой стрелке. 12, 3, 6, 9. Со временем запомните, и эта шпаргалка больше не понадобится.

Ещё одна деталь: `margin` — **может быть отрицательным**. `Padding` — **никогда**. А еще отступы могут принимать значения в виде ключевых слов.

**Для `padding` ЭТО:**

- `initial` — устанавливает значение отступов по умолчанию;
- `inherit` — наследует значение отступов от родителя;
- `unset` — сбрасывает значение отступов до дефолтных или наследует значение, если такое есть, от родителя.

**В случае `margin` к предыдущим трём добавляется ещё одно:**

- `auto` — автоматически вычисляет отступы.

Например:

`margin-top: -20px;`

Этот стиль отодвинет блок на 20 пикселей от верхней границы родителя. Если бы минуса не было, то элемент опустился бы на 20 пикселей вниз от той же границы.

Этот трюк используют только в особенных ситуациях. Когда нужно сдвинуть элемент за края родителя. Или в других сложных композициях.

### Упражнение 3

1/1 point (graded)

Какие значения не может принимать свойство `padding` ?

☒ Отрицательные числа

☐ Положительные числа

☐ Относительные единицы измерения

☐ Ключевое слово `initial`



Show answer

Отправить

✓ Верно (1/1 балл)

## Border

Border — это контур блока. Он отделяет его от других элементов. Мы уже не раз задавали его в наших примерах. Но давайте подробнее поговорим о том, что ещё с ним можно делать и как стилизовать.

### Какие бывают border-ы:

- `solid` — сплошная линия.
- `dotted` — пунктирная линия.
- `dashed` — штриховая линия.
- `double` — двойная линия.
- `inset` — 3D-вдавленная линия.
- `outset` — 3D-выпуклая линия, напоминающая выступ.

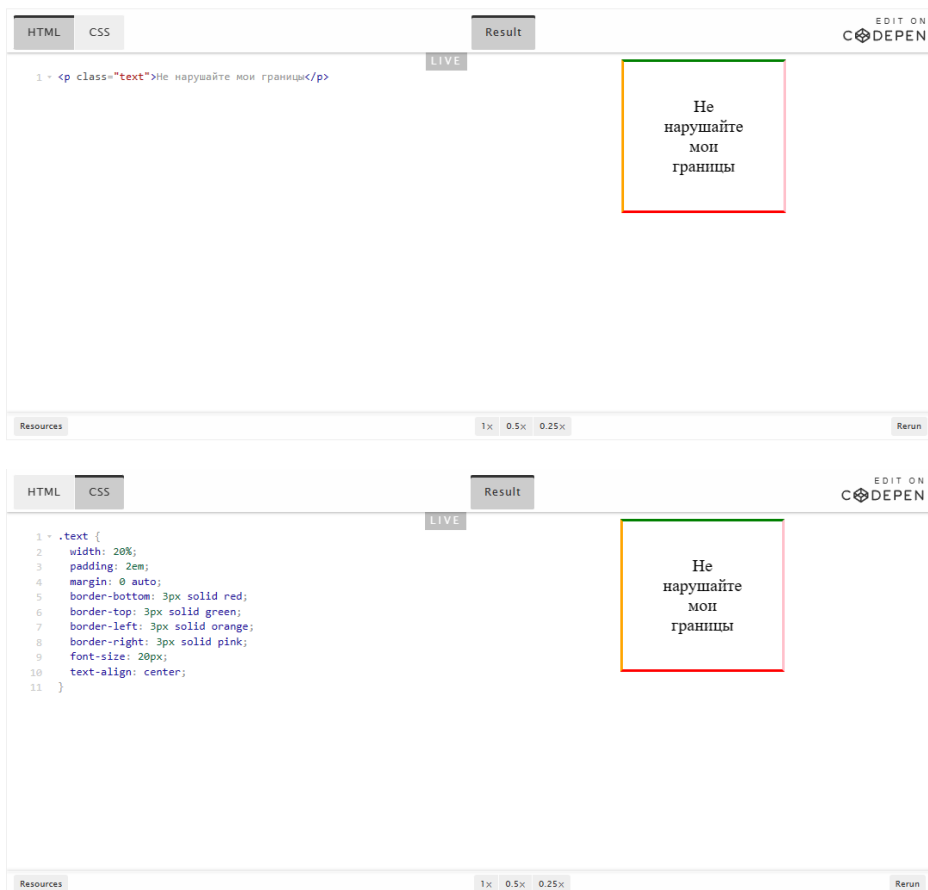
На практике чаще всего используют сплошные, пунктирные и двойные границы. Но об остальных тоже важно помнить.

Прелесть границ в том, что им можно менять не только стили. Посмотрим внимательнее на синтаксис:

**border:** `1px solid red`;

В одной строке мы задали **ширину, стиль и цвет** границы. И это еще не всё.

Мы можем задавать не только сплошную рамку. Если вы хотите границу только сверху или только слева — это посильная задача для CSS. Смотрим:



В нашем примере мы задали каждой стороне границу разного цвета. А могли бы её вообще не задавать — и тогда на месте `border`-а было бы пустое место.

В некоторых ситуациях нам нужно наоборот: **границы удалить**. Например, у кнопки и поля ввода по умолчанию есть границы. И это может противоречить взгляду дизайнера. Чтобы его не расстраивать, можно использовать вот такую запись:

**`border: none;`**

Больше никаких границ. И дизайнер доволен.

И последняя ситуация на сегодня. Вы наверняка видели, что не все кнопки на сайтах квадратные. Даже в этом модуле кнопки переключения уроков — **закруглённые**. Как добиться этого эффекта? Не сложнее, чем удалить все границы разом. Понадобится одна строчка:

```
border-radius: 10px;
```

Это правило закруглит все границы элемента. А если хочется сделать это **выборочно**, то так тоже можно:

```
border-bottom-left-radius: 5px; /* Закруглит нижний левый угол */
border-bottom-right-radius: 10px; /* Закруглит нижний правый
угол */
border-top-left-radius: 2px; /* Закруглит верхний левый угол */
border-top-right-radius: 7px; /* Закруглит верхний правый угол
*/
```

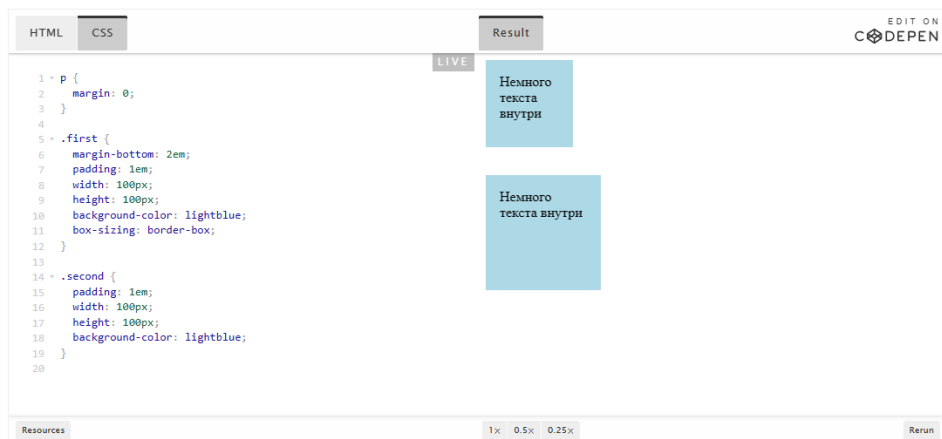
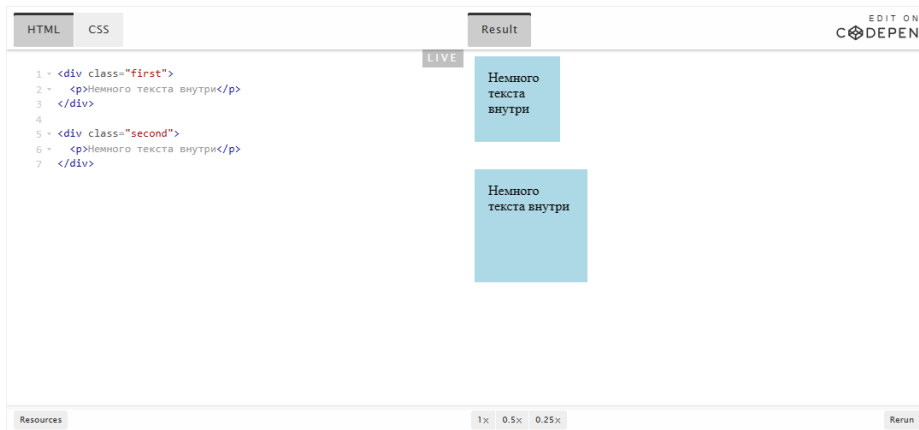
## Упражнение 4

Поработайте с примером самостоятельно. Закруглите углы блока с помощью `border-radius`. Попробуйте сделать каждому углу разные значения.

## Box-sizing

**Box-sizing** — это свойство CSS, которое позволяет регулировать размер элемента, включая его внутренние отступы и границы.

Посмотрите на пример.



У нас есть два одинаковых по ширине блока, с одинаковым текстом и внутренними отступами. Разница лишь в том, что к первому блоку мы применили свойство `box-sizing: border-box`. Поэтому он остался 100 пикселей в ширину. И в это число включены внутренние отступы.

В то время как у второго блока ширина изменилась: к 100 пикселям добавились внутренние отступы. И теперь он шире, чем первый блок.

Так и работает это свойство. Хорошая практика — использовать `box-sizing: border-box` на своём проекте. Так вы будете точно знать, какого размера будут ваши элементы.



## Упражнение 5

4/4 points (graded)

Расставьте в нужном порядке свойства блока (начиная с контента):

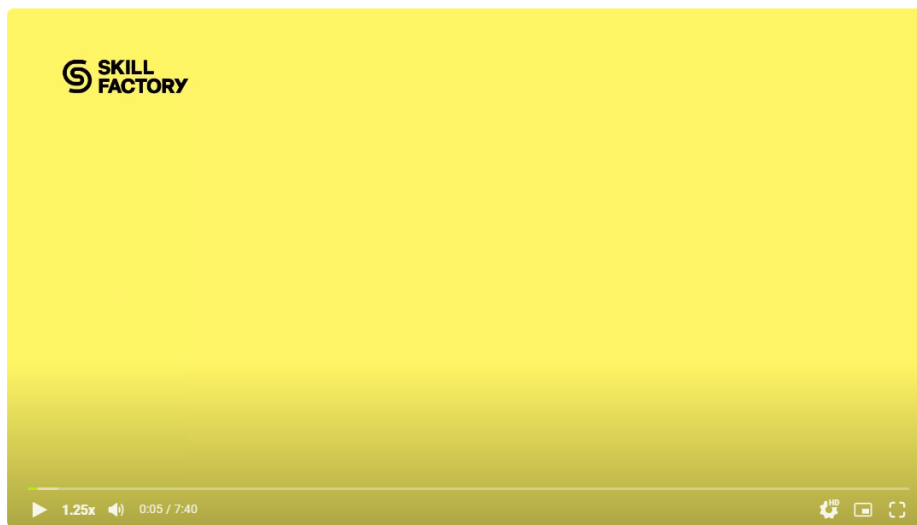
Контент	✓
Внутренние отступы	✓
Рамка	✓
Внешние отступы	✓

[Show answer](#)

Отправить

✓ Верно (4/4 балла)

### Скринкаст: Box-sizing



ПОСМОТРЕТЬ!- посмотрела

Верстать непросто. А когда что-то не работает или работает не так, как надо — это расстраивает.

Хорошая новость — в каждом современном браузере есть *DevTools*. Он создан для отладки вёрстки прямо в браузере, без изменения оригинального кода.

Если бы в реальной жизни был такой же отладчик, то супы никогда бы не пересаливались, а шахматные партии — не проигрывались. Но что имеем.

**В *DevTools* есть целый набор функций, который упрощает жизнь разработчика. Давайте пройдёмся по некоторым:**

- **Изменение стилей и *HTML*-кода.** Вы можете легко изменять стили и *HTML*-код на странице и видеть, как это влияет на верстку. Удалять элементы, передвигать элементы. Изменять их содержание. Маст хэв для новичков.
- **Отладка *JavaScript*.** *JavaScript* не за горами, поэтому имейте в виду. В *DevTools* можно отлаживать код, останавливать его на нужных вам строках и просматривать значения переменных. Без этого не обойдётся ни одна домашка.
- **Оптимизация работы страницы.** В отладчике можно посмотреть, насколько быстро загружается страница. Сколько занимает загрузка каждой картинки, файла со стилями и *JS*-кода. Очень полезно и в некоторых ситуациях важно.

Подробнее о том, как им пользоваться, вы узнаете из видео.

#### Упражнение 1

1/1 point (graded)

Что нельзя делать при помощи инструментов разработчика?

<input type="checkbox"/>	Менять текстовое содержимое элементов
<input type="checkbox"/>	Менять стили элементов
<input type="checkbox"/>	Скрывать элемент
<input checked="" type="checkbox"/>	Получать доступ к данным на диске пользователя
<input type="checkbox"/>	Смотреть, как сайт будет выглядеть на экране телефона
<input checked="" type="checkbox"/>	Создавать новые файлы на диске



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

Мы уже разобрались с блоками. Это наши кирпичики.

Но чтобы строить дальше, нужно знать о **двух важных вещах**:

- Позиционирование;
- Слои.

Они нужны для того, чтобы управлять расположением наших блоков, когда обычных инструментов недостаточно. Или когда планы дизайнера грандиозные.

Давайте разберёмся с ними подробнее.

## Position

Перечислим, какие значения может принимать свойство `position`:

1. `static` — значение по умолчанию. У любого элемента на странице изначальное положение — статичное.

Очень важно понимать, что если у элемента задано это значение, то он не является позиционированным. Его нельзя сместить через `top`, `right`, `bottom` и `left`. А ещё он не является опорным элементом. `Z-index` со `static` тоже не работает.

**Опорный элемент** — элемент, относительно которого позиционируются его потомки.

2. `relative` — почти то же самое, что и `static`, но теперь мы можем позиционировать элемент с помощью `top`, `right`, `bottom` и `left`. Это значение делает элемент опорным. Теперь к нему можно применять `z-index`.
3. `absolute` — элемент позиционируется относительно ближайшего позиционированного родительского элемента. К нему можно применить `top`, `right`, `bottom`, `left` и `z-index`.

Такой элемент выпадает из потока документа и перестаёт влиять на положение других элементов.

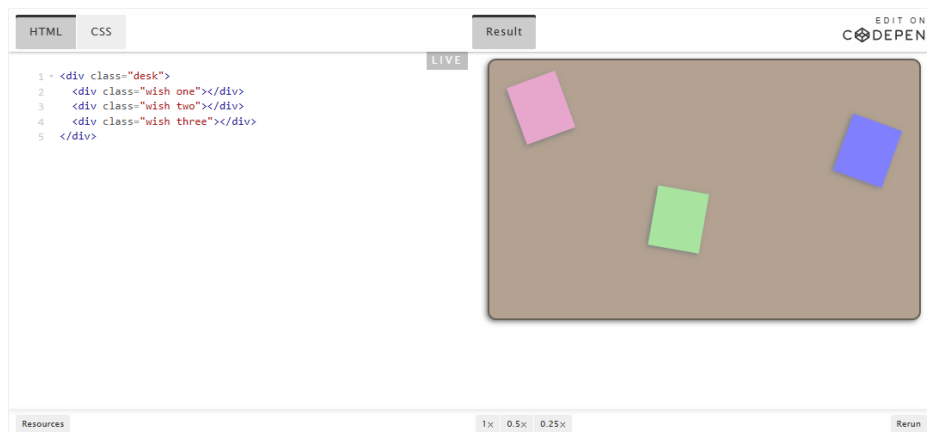
4. `fixed` — элемент позиционируется относительно окна браузера и не перемещается при прокрутке страницы. Можно использовать для вёрстки чата с тех-поддержкой.

Такой элемент тоже выпадает из потока документа и больше не влияет на соседние элементы. К нему можно применить `top`, `right`, `bottom`, `left` и `z-index`.

5. `sticky` — это значение приклеивает элемент к определённому месту на странице, но не сразу. Вспомните шапки сайта, которые не остаются на своём месте при прокрутке, а приклеиваются к верху экрана. Это как раз сделано с помощью `position: sticky`.

А теперь применим теорию на практике.

Представьте доску желаний. На неё можно вешать фотографии мест, куда хочется поехать. Или вещей, которые хочется купить. **Давайте создадим прототип такой доски в коде:**



```
1 .desk {
2   position: relative;
3   margin: 0 auto;
4   width: 500px;
5   height: 300px;
6   background-color: #b3a291;
7   border: 2px solid #665f55;
8   border-radius: 10px;
9   box-shadow: -2px 2px 6px grey;
10 }
11
12 .wish {
13   position: absolute;
14   width: 60px;
15   height: 70px;
16   box-shadow: -2px 2px 6px grey;
17 }
18
19 .one {
20   top: 20px;
21   left: 30px;
22   background-color: #e7a6cb;
23   transform: rotate(-20deg);
24 }
25
26 .two {
27   top: 70px;
28   right: 30px;
29   background-color: #8080ff;
30   transform: rotate(20deg);
31 }
32
33 .three {
34   top: 50%;
35   right: 50%;
36   background-color: #a8e4a0;
37   transform: rotate(10deg);
38 }
39
```

Мы сделали главный блок-родитель и три потомка. Чтобы расположить их на «доске», мы использовали позиционирование.

Мы уже касались его в юните про псевдоэлементы. Когда стилизовали картинку с ежом. Но сегодня уделим свойству `position` больше времени.

Итак, мы задали `position: relative` нашему блоку с классом `desk`. Как вы помните, в этот момент элемент становится опорным. Поэтому теперь

дочерние элементы позиционируются относительно границ опорного элемента. Блока с классом `desk`.

Каждому элементу с классом `wish` мы задали `position: absolute`. И теперь с помощью ключевых свойств `top`, `right`, `bottom` и `left` мы разбросали их по доске.

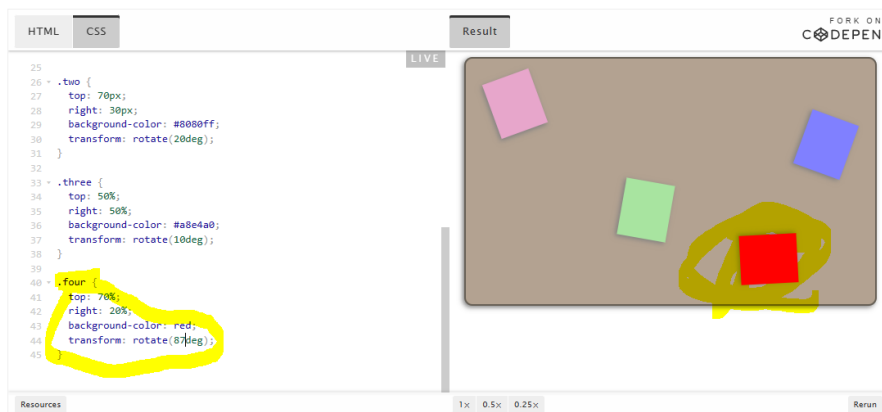
Одну карточку оставили наверху и немного оттолкнули слева. Вторую поместили справа и отодвинули сверху. А третью — приклеили почти по центру.

Без позиционирования у нас это вряд ли бы получилось.

## Упражнение 1

Попробуйте в примере с доской желаний создать ещё одну карточку. Задайте ей любой цвет и прикрепите куда хочется. Попрактикуйтесь в использовании позиционирования.

давайте создадим прототип такой доски в коде:



Давайте попробуем использовать на практике значения `fixed` и `sticky`. И разберёмся, в чем их сходство и отличие.

```

1  ∨ a {
2    text-decoration: none;
3    color: ■ white;
4  }
5  ∨ a: hover {
6    color: □ #25221f;
7  }
8  ∨ .wrap {
9    position: relative;
10   margin: 0 auto;
11   max-width: 600px;
12   height: 300px;
13   background-color: ■ #efece9;
14   overflow-y: scroll;
15 }
16 ∨ .nav {
17   position: sticky;
18   display: flex;
19   justify-content: end;
20   align-items: center;
21   gap: 2em;
22   padding-right: 2em;
23   top: 0;
24   width: auto;
25   height: 60px;
26   background-color: ■ #b3a291;
27 }
28
29 ∨ .content {
30   padding: 2em;
31 }
32
33 ∨ .fixed {
34   padding: 2em;
35   position: fixed;
36   right: 20px;
37   bottom: 0;
38   background-color: ■ #8b7b8b;
39   color: ■ white;
40   border-radius: 5px;
41 }

```

В этом примере мы сделали простенький каркас страницы. Здесь есть шапка с классом `nav`, блок с текстом и зависшая в уголке надпись.

Попробуйте пролистать текст и вы увидите, что **шапка приклеилась к верху страницы** и не меняет своё положение. Мы добились этого с помощью `position: sticky` и `top: 0`.

Такая шапка нужна для того, чтобы пользователь всегда мог перейти на другой раздел. Даже если пролистал далеко вниз.

Это очень полезный приём. Запомните его для дальнейшей работы.

А вот для надписи мы задали `position: fixed`. И приклеили её к низу экрана. На неё не влияют другие элементы. Только сам экран.

Такая вёрстка хорошо подходит для модальных окон и чатов. И надоедливой рекламы.

Но это уже другая история.

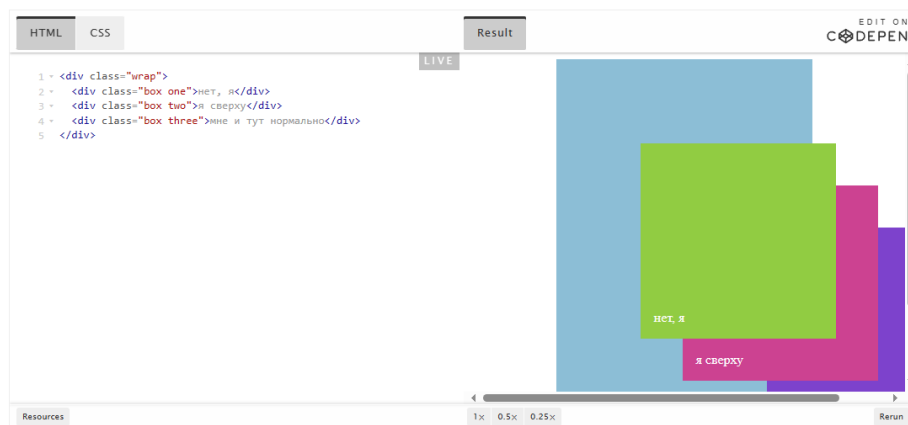
## **Z-index и слои**

**Z-index** — это свойство, которое позволяет управлять порядком слоёв на странице. Оно задаёт уровень, на котором отображается элемент. Элементы с большим значением `z-index` отображаются поверх элементов с меньшим значением `z-index`.

Например, если у элемента *A* значение `z-index` равно 2, а у элемента *B* значение `z-index` равно 1, то элемент *A* будет отображаться поверх элемента *B*.

Давайте рассмотрим пример использования `z-index`:





```
1  .wrap {
2    margin: 0 auto;
3    width: 60%;
4    height: 500px;
5    position: relative;
6    z-index: 0;
7    background-color: #8cbcd6;
8  }
9  .box {
10   display: flex;
11   align-items: end;
12   position: absolute;
13   padding: 1em;
14   left: 250px;
15   width: 200px;
16   height: 200px;
17   color: white;
18 }
19 .one {
20   top: 100px;
21   left: 100px;
22   background-color: #91cc42;
23   z-index: 3;
24 }
25
26 .two {
27   top: 150px;
28   left: 150px;
29   background-color: #cc4291;
30   z-index: 2;
31 }
32
33 .three {
34   top: 200px;
35   background-color: #7d42cc;
36   z-index: 1;
37 }
38
```

Мы создали родителя и три кубика. Дочерние элементы получили абсолютное позиционирование относительно блока с классом wrap. Но без z-index они встали бы в следующем порядке: внизу зелёный, над ним — розовый, и в самом верху — фиолетовый. Потому что в разметке они идут подряд. Но с помощью z-index мы поменяли порядок слоёв так, как нам захотелось.

## Упражнение 2

Поменяйте порядок кубиков в примере. Задайте им другие `z-index`-ы, в любом порядке. Можете создать ещё кубиков и также изменить их слой.

**Это очень полезный инструмент в арсенале разработчика. И он точно пригодится в создании этих элементов:**

- **Модальные окна.** Эти элементы всегда должны быть поверх всего остального контента на странице. Это работа для `z-index`.
- **Выпадающие меню.** Вы наверняка видели такие на платформах с музыкой и в социальных сетях. Чтобы они не провалились под фон, им можно задать высокий `z-index`.
- **Надписи на картинках.** Помните наш пример с затемнённой картинкой? Мы задали для надписи высокий `z-index`, чтобы градиент её не перекрыл.

А теперь перейдём к закреплению новых знаний.

Какое значение `position` используется элементами по умолчанию?

☐ `relative`

☒ `static`

☐ `sticky`

☐ `fixed`

☐ `absolute`

☐ Ни одно из вышеперечисленных



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

#### Упражнение 4

1/1 point (graded)

Какое значение нужно задать свойству `position`, чтобы сделать не исчезающую шапку-навигации?

☐ `fixed`

☐ `static`

☐ `absolute`

☒ `sticky`



[Show answer](#)

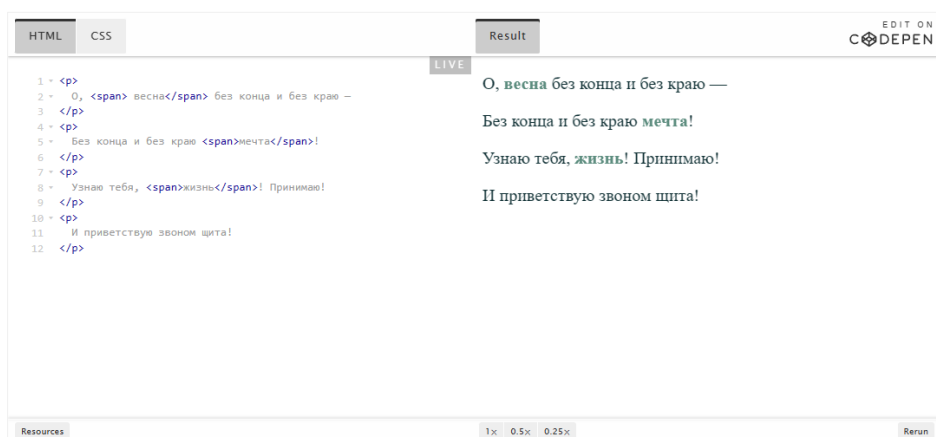
Отправить

✓ Верно (1/1 балл)

До этого мы стилизовали текст параграфами. Но если вы хотите выделить в тексте только одно слово или несколько, не обязательно задавать стили всему блоку.

Для этих целей существует специальный элемент `<span>`. В этот элемент можно обернуть нужную часть текста и задать ему стили.

Давайте попробуем:



С помощью обёртки слов в `<span>` мы смогли сделать текст живее и интереснее.

Конечно, этот способ подойдет не только для оживления текста, но и для смысловых выделений важных слов. Их можно встретить и в блогах, и в онлайн-магазинах. Полезный в работе метод.

## Упражнение 1

Посмотрите на картинку. У нас есть четыре параграфа (`<p>`) со строками стихотворения. Ключевые слова обёрнуты в особый элемент `<span>`, чтобы задать им стили. А ещё у нас есть две одинаковых кнопки `<button>`.

Заметался *пожар* голубой,  
Позабылись родимые дали.  
В первый раз я запел про *любовь*,  
В первый раз *отрекаюсь* скандалить.

Ваша первая задача: сделать правильную разметку. Расположить теги `<p>`, `<button>` и `<span>` в своей разметке так, как на картинке. Используйте `coderep`, чтобы контролировать работу.

#### Подсказка

Не усложняйте код. Используйте пример в юните как образец.

#### Посмотреть ответ

```
<p>Заметался <span>пожар</span> голубой,</p>
<p>Позабылись родимые дали.</p>
<p>В первый раз я запел про <span>любовь</span>,</p>
<p>В первый раз <span>отрекаюсь</span> скандалить.</p>

<button>Нравится</button>
<button>Не нравится</button>
```

Ваша вторая задача: написать нужные стили тегам `<p>`, `<button>` и `<span>`. В этом задании не нужны классы, используйте селектор тега. Ширина рамки — 2 пикселя.

#### Подсказка

Вам понадобится не так много свойств. Рамка, стиль шрифта и цвет текста.

#### Посмотреть ответ

```
button {
  border: 2px solid green;
}
span {
  font-style: italic;
  color: red;
}
```

За этот модуль вы проделали огромную работу. С чем мы вас и поздравляем. 😊

И предлагаем доработать **проект сайта-визитки**. Самое время применить полученные знания на практике.

## Вы можете добавить:

- анимации при наведении на ссылки и кнопки (→ «[Псевдоклассы](#)»);
- стилизацию отдельных слов или всего текста (→ «[Работа со стилями](#)»);
- меню при помощи якорей (→ «[Селектор по идентификатору](#)»);
- фон для всей страницы и цвета конкретных элементов (→ «[Цвета в CSS](#)»);
- отступы между разделами (→ «[Блочная модель документа](#)»).

**Давайте вспомним главные рекомендации по выполнению задания:**

6. Избегайте стилизаций элемента по тегу. Используйте классы. Они надёжнее и удобнее.
7. Не копируйте стили. Если у вас есть несколько блоков с одинаковыми стилями — задайте им общий класс. У разработчиков есть правило DRY — don't repeat yourself.
8. Попробуйте самостоятельно разобраться с анимацией элементов. Это можно сделать с помощью псевдокласса `:hover` и свойства [transition](#). Анимация добавит вашей странице интерактивности.

В самостоятельной работе вам предоставляется свобода для творчества и выбора технологий. Используйте полученные знания, чтобы создать красивый и функциональный сайт.

Продемонстрируйте ваши умения и навыки веб-разработки. Не забывайте следовать рекомендациям и избегать повторений в стилях. Удачи!

А вот такая [страница](#) получилась у нас после применения знаний из модуля «Основы CSS. Layouts».

В этом модуле вы получили фундаментальные знания CSS.

### **Давайте перечислим самое важное, чему вы научились в модуле:**

- Подключать внешние стили и задавать их внутри *HTML*-разметки.
- Задавать элементам классы и идентификаторы, а затем использовать их для стилизации.
- Отличать псевдоклассы от псевдоэлементов и работать с ними в *CSS*.
- Определять приоритет селекторов.
- Задавать цвета через *RGB(A)*, *HSL(A)* и *HEX*.
- Управлять размером элементов с помощью абсолютных и относительных единиц измерения.
- Задавать блочным элементам границу, а также внешние и внутренние отступы.
- Работать с кодом через *DevTools*.
- Менять позиционирование элементов и создавать из них сложные композиции.

Предлагаем вам пройти итоговый тест по модулю, чтобы закрепить полученные знания.

Как подключить внешние стили к HTML-разметке?

- ☐ Через атрибут `style` у тега элемента.
- ☒ Через тег `<link>` внутри `<head>` документа.
- ☐ Через тег `<style>` внутри `<head>` документа.
- ☐ Невозможно подключить внешние стили к HTML-разметке.



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

Каким будет результат применения этих стилей к элементу `h1` ?

```
h1 {  
  font-size: 36px;  
  color: blue;  
  font-family: "Helvetica Neue";  
}
```

- ☒ Текст заголовка `h1` будет иметь размер шрифта 36 пикселей, синий цвет и шрифт Helvetica Neue.
- ☐ Текст заголовка `h1` будет иметь размер шрифта 36 пикселей и синий цвет, но шрифт будет использоваться по умолчанию для браузера.
- ☐ Текст заголовка `h1` будет иметь размер шрифта 16 пикселей, красный цвет и шрифт Helvetica Neue.



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

Вопрос 3

1/1 point (graded)

Какой из перечисленных селекторов CSS выберет все элементы с классом `my-class`, которые являются прямыми потомками элемента с классом `parent-class` ?

- ☒ `.parent-class > .my-class`
- ☐ `.parent-class .my-class`
- ☐ `.my-class + .parent-class`
- ☐ `.parent-class ~ .my-class`



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

#### Вопрос 4

1/1 point (graded)

Что такое псевдоклассы в CSS?

- ☐ Это способ задания стилей для определенной группы элементов.
- ☒ Это способ задания стилей для определенного элемента в определённом состоянии.
- ☐ Это способ задания стилей для элементов, которые находятся в одном родительском контейнере.
- ☐ Это способ задания стилей для элементов, которые находятся на определенном уровне вложенности.



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

#### Вопрос 5

1/1 point (graded)

Какие из перечисленных селекторов являются псевдоэлементами в CSS? Выберите все подходящие варианты.

- ☐ `hover`
- ☒ `before`
- ☐ `nth-child`
- ☒ `after`
- ☒ `first-letter`



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

#### Вопрос 6

1/1 point (graded)

Выберите правильные ответы:

- ☒ Идентификаторы имеют более высокий приоритет, чем классы.
- ☐ Селекторы псевдоэлементов имеют более высокий приоритет, чем селекторы псевдоклассов.
- ☐ Селекторы атрибутов имеют более высокий приоритет, чем селекторы классов.
- ☐ Все перечисленные утверждения верны.
- ☐ Ни одно из перечисленных утверждений не верно.



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

#### Вопрос 7



### Вопрос 7

1/1 point (graded)

Какой цвет соответствует HEX-коду #FFA500 ?

☐ Красный

☐ Зелёный

☐ Синий

☒ Оранжевый



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

✓ Верно (1/1 балл)

### Вопрос 8

1/1 point (graded)

Какая из перечисленных единиц измерения в CSS является относительной и зависит от размера шрифта родительского элемента?

☐ px

☐ rem

☒ em

☐ vw



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

### Вопрос 9

1/1 point (graded)

Запишите свойство, которое задаст элементу внешний отступ справа – на 20px .

margin-right: 20px; ✓

[Show answer](#)

Отправить

✓ Верно (1/1 балл)

### Вопрос 10

1/1 point (graded)

Какие задачи можно решить с помощью DevTools в браузере?

☒ Изменять стили элементов

☒ Анализировать и отлаживать JavaScript код

☐ Создавать новые файлы и проекты внутри DevTools

☒ Оценить производительность и оптимизировать код

☒ Разрабатывать и тестировать адаптивный дизайн

☒ Отладить ошибки в вёрстке и JavaScript коде

☒ Имитировать мобильные устройства и различные экраны

☐ Удалять файлы на компьютере пользователя



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

### Вопрос 11

1/1 point (graded)

Какие значения может принимать свойство z-index в CSS?

☒ auto, inherit, initial

☒ Числовые значения

☐ top, bottom

☐ relative, absolute, fixed



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

### Вопрос 12

1/1 point (graded)

Мы хотим стилизовать одно слово внутри параграфа. Какой тег пропущен? Пишите без кавычек.

`<p> Я пришёл к тебе с приветом, рассказать что <...>солнце</...> встало!</p>`

span



[Show answer](#)

Отправить

✓ Верно (1/1 балл)

## Дополнительно

Впереди — продвинутый CSS и ещё больше интересного. А пока что мы собрали для вас полезные материалы для самостоятельного изучения:


- [Сайт](#) с большим количеством информации по *HTML* и *CSS*.
- [Платформа](#) для самостоятельного изучения веб-разработки на английском языке.
- [Справочник](#) по *frontend*-разработке от команды *Mozilla*.
- И ещё один современный [справочник](#) на русском языке.
- [Учебник](#) по вёрстке на русском языке.

До новых встреч в следующем модуле!

В этом тренажёре вы найдете дополнительные задания для закрепления основных навыков, необходимых для стилизации веб-страниц, и попрактикуетесь преобразовывать внешний вид страницы — делать её более структурированной и визуально привлекательной. Для выполнения упражнений необходимо обязательно изучить теорию по данной теме в соответствующих модулях.

Задания в тренажёре расположены тематически и связаны между собой по принципу усложнения. Вам будет дан исходный код, в котором нужно произвести изменения. Результат вашей работы должен совпадать с примером. В случае затруднений, вы можете заглянуть в готовый код результата в конце этого модуля. Для работы вы можете использовать любой удобный для вас инструмент: онлайн-песочницу, редактор кода или IDE.

Для выполнения заданий можно применить несколько путей решения. Однако помните, что хорошее решение — в первую очередь *простое*.

В конце каждого юнита вы увидите кнопку . С её помощью вы можете отмечать юниты, с задачами в которых вы уже справились, чтобы вам было легче отследить свой прогресс.

☆ **Обратите внимание!** Задания в этом тренажёре не оцениваются и не влияют на общий процент успешного прохождения курса («Total»). Учебный тренажёр в первую очередь даёт вам возможность попрактиковаться, а не тестирует вас.

## Задание 1

Установите для страницы красный цвет фона (background-color: red;), используя глобальные стили.

**Исходный код:**

```
<!DOCTYPE html>
<html>
  <head>
    <style>

    </style>
  </head>

  <body>
    <h1>Тренируемся подключать стили.</h1>
  </body>
</html>
```

**Ожидаемый результат:**



**Тренируемся подключать стили.**

## Задание 2

Установите белый цвет заголовка (color: white;), используя внутренние стили.

**Исходный код:**

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background-color: red;
      }
    </style>
  </head>
  <body>
    <h1>Тренируемся подключать стили.</h1>
  </body>
</html>
```

```
    }
  </style>
</head>

<body>
  <h1>Тренируемся подключать стили.</h1>
</body>
</html>
```

**Ожидаемый результат:**



**Тренируемся подключать стили.**

### Задание 3

Теперь вынесите стили (background-color: red; и color: white;) в отдельный CSS-файл, удалив глобальное и внутреннее подключения.

Соберите все селекторы, свойства и значения в отдельный файл style.css и подключите его к вашей HTML-странице.

**Исходный код:**

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background-color: red;
      }
    </style>
  </head>

  <body>
    <h1 style="color: white;">Тренируемся подключать стили.</h1>
  </body>
</html>
```

Ожидаемый результат (ничего не меняется):

Тренируемся подключать стили.

I have completed this

## Задание 1

Оберните строки в теги так, чтобы они располагались одна над другой.

Исходный код:

```
<!DOCTYPE html>
<html>
  <head>
  </head>

  <body>
    <h1>Поток документа.</h1>
    строка 1
    строка 2
    строка 3
    строка 4
  </body>
</html>
```

Ожидаемый результат:

---

**Поток документа.**

строка 1  
строка 2  
строка 3  
строка 4

## Задание 2

Теперь замените тег, расположив все элементы в одной строке.

## Исходный код:

```
<!DOCTYPE html>
<html>
  <head>
  </head>

  <body>
    <h1>Поток документа.</h1>
    <p>строка 1</p>
    <p>строка 2</p>
    <p>строка 3</p>
    <p>строка 4</p>
  </body>
</html>
```

## Ожидаемый результат:

### Поток документа.

строка 1 строка 2 строка 3 строка 4

## Задание 1

Установите для заголовка красный цвет в шестнадцатеричной системе

## Исходный код:

```
<!DOCTYPE html>
<html>
  <head>
  </head>

  <body>
    <h1>Базовые свойства. Цвета</h1>
    <p>строка 1</p>
    <p>строка 2</p>
    <p>строка 3</p>
    <p>строка 4</p>
  </body>
</html>
```

Ожидаемый результат:

## Базовые свойства. Цвета

строка 1

строка 2

строка 3

строка 4

## Задание 2

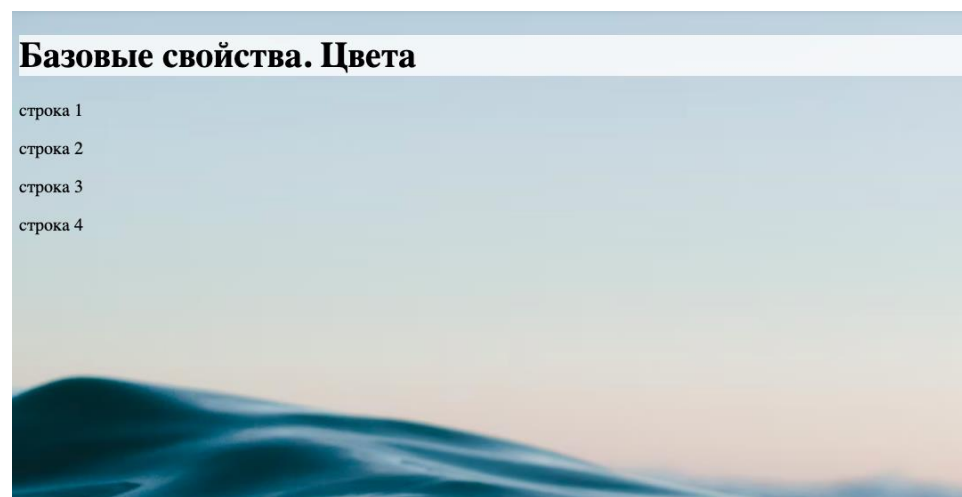
Установите для страницы [фоновое изображение](#), для заголовка сделайте белый цвет фона с прозрачностью 80%. Используйте систему RGBA.

Исходный код:

```
<!DOCTYPE html>
<html>
  <head>
  </head>

  <body>
    <h1>Базовые свойства. Цвета</h1>
    <p>строка 1</p>
    <p>строка 2</p>
    <p>строка 3</p>
    <p>строка 4</p>
  </body>
</html>
```

Ожидаемый результат:





## Задание 3

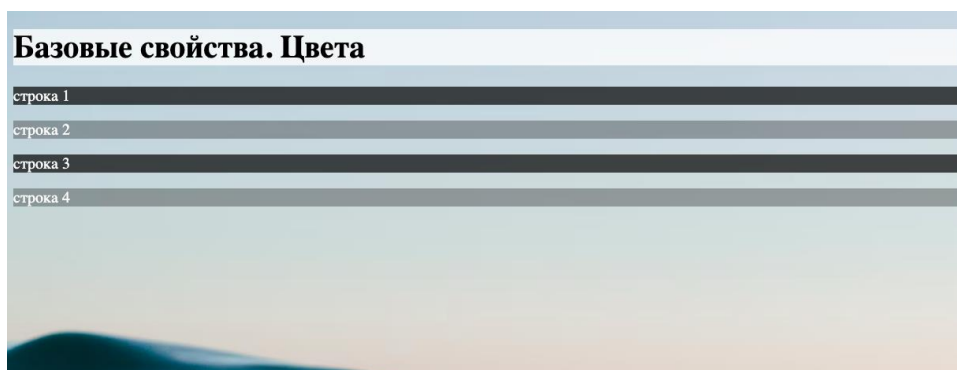
Установите для нечётных строк фоновый цвет — 70% серый, для чётных — 30% серый. Цвет шрифта сделайте белым. Используйте систему HSLA и селекторы :nth-child.

### Исходный код:

```
<!DOCTYPE html>
<html>
  <head>
  </head>

  <body style="background-image:
url('https://images.unsplash.com/photo-1518837695005-
2083093ee35b?ixlib=rb-
1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&
fit=crop&w=1470&q=80');">
    <h1 style="background-color: rgba(255, 255, 255, .8);">Базовые
свойства. Цвета</h1>
    <p>строка 1</p>
    <p>строка 2</p>
    <p>строка 3</p>
    <p>строка 4</p>
  </body>
</html>
```

### Ожидаемый результат:



I have completed this

## Задание 1

Установите для html-элемента размер шрифта 16px, а для заголовка h1 — 40px. Для величины шрифта заголовка используйте относительные величины.

### Исходный код:

```
<!DOCTYPE html>
<html>
  <head>
  </head>

  <body>
    <div class="title-block">
      <h1>Базовые свойства. Размеры</h1>
      <h2>Занимательный подзаголовок</h2>
    </div>
  </body>
</html>
```

### Ожидаемый результат:

# Базовые свойства. Размеры

## Занимательный подзаголовок

## Задание 2

Напишите стили для .title-block, h1 и h2 так, чтобы размер шрифта h1 остался 40px, а размер шрифта подзаголовка был в два раза меньше. Для размеров heading-элементов используйте величины em.

### Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
```

```
    html {
      font-size: 16px;
    }

    h1 {
      font-size: 2.5rem;
    }

  </style>
</head>

<body>
  <div class="title-block">
    <h1>Базовые свойства. Размеры</h1>
    <h2>Занимательный подзаголовок</h2>
  </div>
</body>

</html>
```

Ожидаемый результат:

## Базовые свойства. Размеры

Занимательный подзаголовок

### Задание 3

Установите стиль для ul так, чтобы каждая следующая строка была в два раза больше предыдущей.

Исходный код:

```
<!DOCTYPE html>
<html>
  <head>
  </head>

  <body>
    <ul>
      <li>Строка 1</li>
```

```
<ul>
  <li>Строка 2</li>
  <ul>
    <li>Строка 3</li>
  </ul>
</ul>
</ul>
</body>
</html>
```

**Ожидаемый результат:**

- Строка 1
  - Строка 2

■ Строка 3

## Задание 4

Установите синему блоку ширину, равную ширине страницы, а красный растяните на всю высоту.

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      body {
        margin: 0;
      }

      .blue {
        background-color: blue;
        width: 20px;
        height: 20px;
      }

      .red {
```

```
        background-color: red;
        width: 40px;
        height: 40px;
    }

</style>
</head>

<body>
    <div class="red">
        <div class="blue"></div>
    </div>
</body>

</html>
```

**Ожидаемый результат:**



## Задание 5

В прошлом задании вы увидели, что синий блок вышел далеко за границы родительского красного блока. Замените свойство `width` на `min-` и `max-width` так, чтобы красный блок растянулся вслед за синим, но не сжимался меньше 40px.

**Исходный код:**

```
<!DOCTYPE html>
<html>

    <head>
        <style>
            body {
                margin: 0;
            }

            .blue {
```

```
        background-color: blue;
        width: 100vw;
        height: 20px;
    }

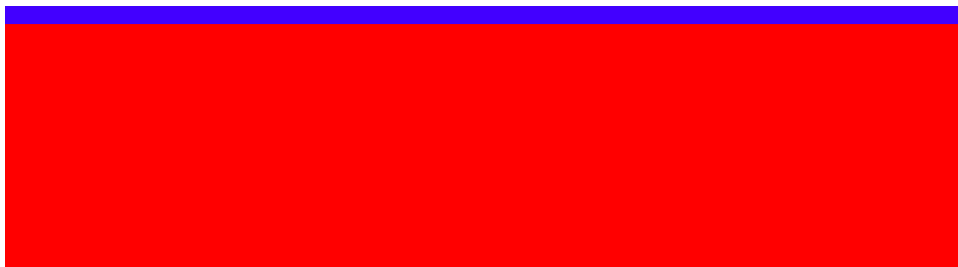
    .red {
        background-color: red;
        width: 40px;
        height: 100vh;
    }

</style>
</head>

<body>
    <div class="red">
        <div class="blue"></div>
    </div>
</body>

</html>
```

**Ожидаемый результат:**



## Задание 6

Задайте синему блоку высоту, равную половине высоты родительского элемента.

**Исходный код:**

```
<!DOCTYPE html>
<html>

    <head>
        <style>
            body {
```

```
        margin: 0;
    }

    .blue {
        background-color: blue;
        width: 100vw;
        height: 20px;
    }

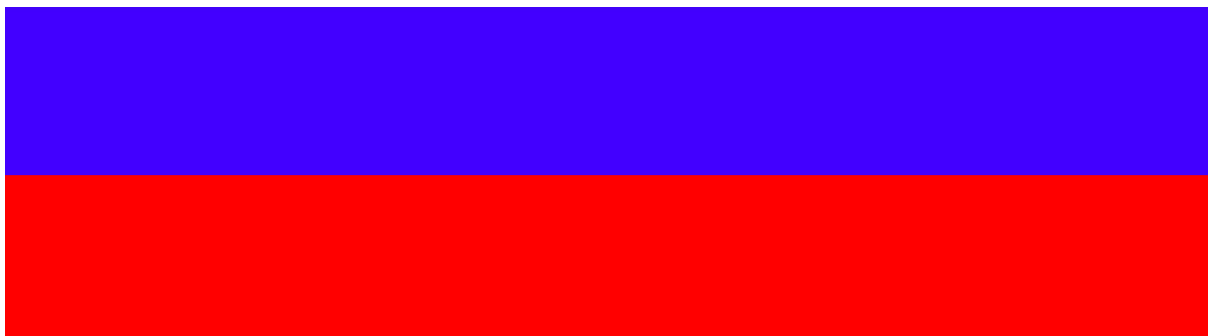
    .red {
        background-color: red;
        min-width: 40px;
        max-width: 100vw;
        height: 100vh;
    }

</style>
</head>

<body>
    <div class="red">
        <div class="blue"></div>
    </div>
</body>

</html>
```

**Ожидаемый результат:**



## Задание 1

Задайте заголовку h2 горизонтальные margin так, чтоб он всегда был по центру страницы. Обратите внимание, что для центрирования элемента таким способом он должен быть блочным и иметь заданную ширину.

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      h1 {
        background-color: #F0BA7D;
        margin: 25px;
      }

      h2 {
        background-color: #CADADD;
        width: 50vw;
      }

    </style>
  </head>

  <body>
    <h1>Базовые свойства. Расстояния, границы и отступы</h1>
    <h2>Занимательный подзаголовок</h2>
  </body>

</html>
```

Ожидаемый результат:

**Базовые свойства. Расстояния, границы и отступы**

**Занимательный подзаголовок**

## Задание 2

Задайте заголовку h2 отступ по вертикали таким образом, чтоб расстояние между h1 и h2 было 50px. Учтите, что h1 уже имеет вертикальный отступ в 25px.

Несмотря на то, что первый заголовок уже имеет отступ в 25px, мы должны задать второму заголовку отступ в 50px, если нам требуется такое



расстояние между элементами. Это явление называется [схлопыванием внешних отступов](#).

### Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      h1 {
        background-color: #F0BA7D;
        margin: 25px;
      }

      h2 {
        background-color: #CADADD;
        width: 50vw;
        margin: 0 auto;
      }

    </style>
  </head>

  <body>
    <h1>Базовые свойства. Расстояния, границы и отступы</h1>
    <h2>Занимательный подзаголовок</h2>
  </body>

</html>
```

### Ожидаемый результат:

**Базовые свойства. Расстояния, границы и отступы**

Занимательный подзаголовок

## Задание 3

Задайте подзаголовку padding: 25px по горизонтали, 10px сверху и 20px снизу.

## Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      h1 {
        background-color: #F0BA7D;
        margin: 25px;
      }

      h2 {
        background-color: #CADADD;
        width: 50vw;
        margin: 50px auto;
      }

    </style>
  </head>

  <body>
    <h1>Базовые свойства. Расстояния, границы и отступы</h1>
    <h2>Занимательный подзаголовок</h2>
  </body>

</html>
```

## Ожидаемый результат:

**Базовые свойства. Расстояния, границы и отступы**

Занимательный подзаголовок

## Задание 1

Сместите все блоки с нечётными цифрами на 50px вправо.

## Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        background-color: lightgray;
        padding: 15px 5px;
      }

      .item {
        margin: 5px;
        padding: 10px;
        background-color: blue;
        color: white;
        display: block;
      }

    </style>
  </head>

  <body>
    <div class="wrapper">
      <span class="item">1</span>
      <span class="item">2</span>
      <span class="item">3</span>
      <span class="item">4</span>
      <span class="item">5</span>
      <span class="item">6</span>
      <span class="item">7</span>
      <span class="item">8</span>
      <span class="item">9</span>
      <span class="item">10</span>
    </div>
  </body>

</html>
```

**Ожидаемый результат:**



## Задание 2

Зафиксируйте последний элемент `.item` в самом низу viewport так, чтобы при прокрутке он не менял своего положения.

Вы можете заметить, что как только блочный элемент получает `position: fixed`, он сразу же перестает занимать все отведённое ему место. Это происходит потому, что элемент покидает нормальный поток документа и в этом случае ширина (если она явно не указана), определяется его контентом. То же касается и `position: absolute`.

### Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        background-color: lightgray;
        padding: 15px 5px;
      }

      .item {
        margin: 5px;
        padding: 10px;
        background-color: blue;
        color: white;
        display: block;
      }
    </style>
```

```
</head>

<body>
  <div class="wrapper">
    <span class="item">1</span>
    <span class="item">2</span>
    <span class="item">3</span>
    <span class="item">4</span>
    <span class="item">5</span>
    <span class="item">6</span>
    <span class="item">7</span>
    <span class="item">8</span>
    <span class="item">9</span>
    <span class="item">10</span>
  </div>
</body>

</html>
```

**Ожидаемый результат:**

### Задание 3

Укажите первому элементу .item стили так, чтоб при прокрутке он «прилипал» к верхней части viewport.

Эту задачу нельзя решить, просто указав элементу position: sticky, поскольку такое позиционирование начинает работать, только если у элемента указано хотя бы одно из свойств: top, bottom, left или right.

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        background-color: lightgray;
        padding: 15px 5px;
        height: 200vh;
      }
    </style>
  </head>
</html>
```

```
.item {
  margin: 5px;
  padding: 10px;
  background-color: blue;
  color: white;
  display: block;
}

.item:first-child {
  background: red;
}

</style>
</head>

<body>
  <div class="wrapper">
    <span class="item">1</span>
    <span class="item">2</span>
    <span class="item">3</span>
    <span class="item">4</span>
    <span class="item">5</span>
    <span class="item">6</span>
    <span class="item">7</span>
    <span class="item">8</span>
    <span class="item">9</span>
    <span class="item">10</span>
  </div>
</body>

</html>
```

**Ожидаемый результат:**

## Задание 4

Расположите заголовок поверх розового фона. Эту задачу можно решить двумя способами, изменив стили либо фона, либо заголовка. Попробуйте оба варианта.

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .pink {
        width: 100%;
        height: 200px;
        background: pink;
      }

      h1 {
        font-size: 100px;
        padding: 50px;
        margin: 0;
      }
    </style>
  </head>

  <body>
    <div class="pink"></div>
    <h1>Position</h1>
  </body>

</html>
```

**Ожидаемый результат:**



# Position

## Задание 5

Поместите красный круг внутрь синего квадрата, изменив стили только у синего квадрата.

**Исходный код:**

```
<!DOCTYPE html>
<html>

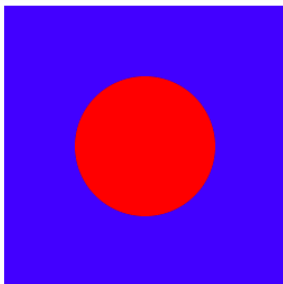
  <head>
    <style>
      .blue {
        width: 200px;
        height: 200px;
        background: blue;
      }

      .red {
        width: 100px;
        height: 100px;
        background: red;
        border-radius: 50%;
        position: absolute;
        top: 50px;
        right: 50px;
      }
    </style>
  </head>

  <body>
    <div class="blue">
      <div class="red"></div>
    </div>
  </body>

</html>
```

**Ожидаемый результат:**



## Задание 6

Выведите заголовок «Position» поверх геометрических фигур.



## Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .blue {
        width: 200px;
        height: 200px;
        background: blue;
        position: absolute;
      }

      .red {
        width: 100px;
        height: 100px;
        background: red;
        border-radius: 50%;
        position: absolute;
        top: 50px;
        right: 50px;
      }

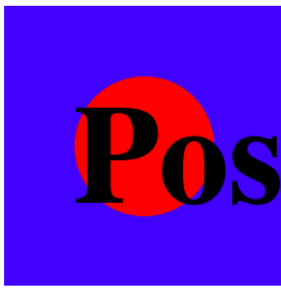
      h1 {
        font-size: 100px;
        padding: 50px;
        margin: 0;
      }

    </style>
  </head>

  <body>
    <div class="blue">
      <div class="red"></div>
    </div>
    <h1>Position</h1>
  </body>

</html>
```

## Ожидаемый результат:



# Position

## Задание 1

Измените стиль элемента `.item` так, чтоб цифры выстроились в колонку.

### Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        background-color: lightgray;
        padding: 15px 5px;
      }

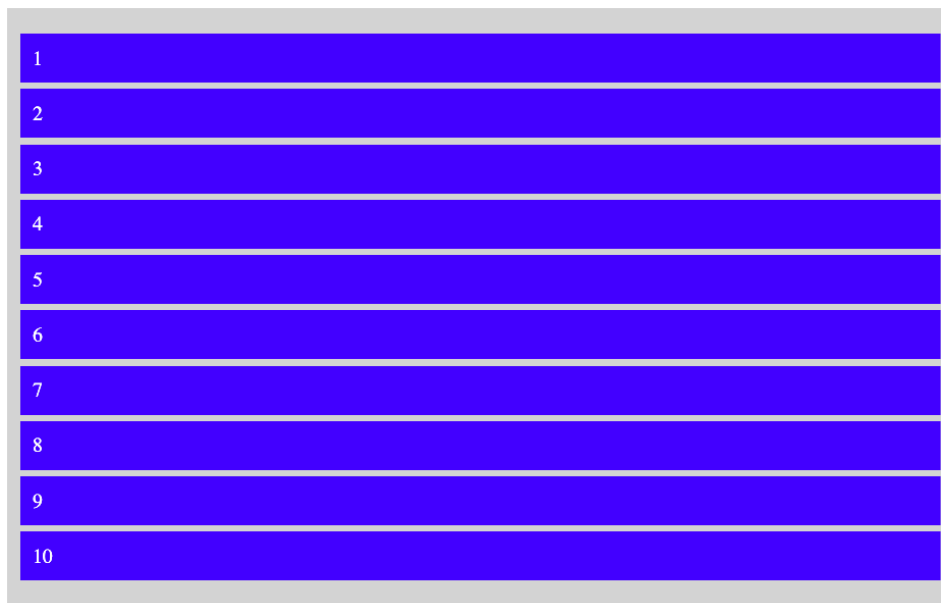
      .item {
        margin: 5px;
        padding: 10px;
        background-color: blue;
        color: white;
      }
    </style>
  </head>

  <body>
    <div class="wrapper">
      <span class="item">1</span>
      <span class="item">2</span>
      <span class="item">3</span>
      <span class="item">4</span>
      <span class="item">5</span>
      <span class="item">6</span>
      <span class="item">7</span>
      <span class="item">8</span>
      <span class="item">9</span>
      <span class="item">10</span>
    </div>
  </body>
</html>
```

```
        </div>
    </body>

</html>
```

**Ожидаемый результат:**



## Задание 2

Теперь снова выстройте цифры в ряд, но при этом каждый элемент `.item` должен иметь ширину 50px. Изменяйте стили только у класса `.item`.

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        background-color: lightgray;
        padding: 15px 5px;
      }

      .item {
        margin: 5px;
        padding: 10px;
        background-color: blue;
        color: white;
      }
    </style>
  </head>
  <body>
```

```

        display: block;
    }

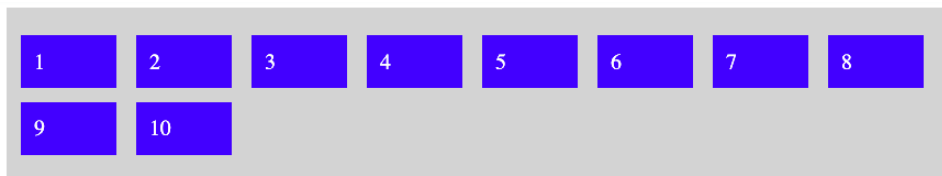
</style>
</head>

<body>
    <div class="wrapper">
        <span class="item">1</span>
        <span class="item">2</span>
        <span class="item">3</span>
        <span class="item">4</span>
        <span class="item">5</span>
        <span class="item">6</span>
        <span class="item">7</span>
        <span class="item">8</span>
        <span class="item">9</span>
        <span class="item">10</span>
    </div>
</body>

</html>

```

**Ожидаемый результат:**



### Задание 3

Скройте все блоки с чётными цифрами.

**Исходный код:**

```

<!DOCTYPE html>
<html>

    <head>
        <style>
            .wrapper {
                background-color: lightgray;
                padding: 15px 5px;
            }
        </style>
    </head>
    <body>
        <div class="wrapper">
            <span class="item">1</span>
            <span class="item">2</span>
            <span class="item">3</span>
            <span class="item">4</span>
            <span class="item">5</span>
            <span class="item">6</span>
            <span class="item">7</span>
            <span class="item">8</span>
            <span class="item">9</span>
            <span class="item">10</span>
        </div>
    </body>
</html>

```

```
}

.item {
  margin: 5px;
  padding: 10px;
  background-color: blue;
  color: white;
  display: inline-block;
  width: 50px;
}

</style>
</head>

<body>
  <div class="wrapper">
    <span class="item">1</span>
    <span class="item">2</span>
    <span class="item">3</span>
    <span class="item">4</span>
    <span class="item">5</span>
    <span class="item">6</span>
    <span class="item">7</span>
    <span class="item">8</span>
    <span class="item">9</span>
    <span class="item">10</span>
  </div>
</body>

</html>
```

**Ожидаемый результат:**



## Задание 1

Измените стиль родительского блока так, чтоб цифры выстроились в колонку в обратном порядке.

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        display: flex;
        background-color: lightgray;
        align-items: flex-start;
      }

      .item {
        margin: 5px;
        padding: 10px;
        background-color: blue;
        color: white;
      }
    </style>
  </head>

  <body>
    <div class="wrapper">
      <span class="item">1</span>
      <span class="item">2</span>
      <span class="item">3</span>
      <span class="item">4</span>
      <span class="item">5</span>
      <span class="item">6</span>
      <span class="item">7</span>
      <span class="item">8</span>
      <span class="item">9</span>
      <span class="item">10</span>
    </div>
  </body>

</html>
```

**Ожидаемый результат:**



## Задание 2

Измените стиль родительского блока, разрешив перенос блоков и разместив их в несколько строк.

### Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        display: flex;
        background-color: lightgray;
        align-items: flex-start;
      }

      .item {
        margin: 5px;
        padding: 10px;
        background-color: blue;
        color: white;
        min-width: 25%;
      }
    </style>
  </head>

  <body>
```

```
<div class="wrapper">
  <span class="item">1</span>
  <span class="item">2</span>
  <span class="item">3</span>
  <span class="item">4</span>
  <span class="item">5</span>
  <span class="item">6</span>
  <span class="item">7</span>
  <span class="item">8</span>
  <span class="item">9</span>
  <span class="item">10</span>
</div>
</body>

</html>
```

**Ожидаемый результат:**

1	2	3
4	5	6
7	8	9
10		

### Задание 3

Измените стиль родительского блока так, чтобы все блоки поместились в контейнер и выстроились в обратном порядке от 10 до 1. Используйте свойство flex-flow.

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        display: flex;
        background-color: lightgray;
        align-items: flex-start;
```



```

    }

    .item {
      margin: 5px;
      padding: 10px;
      background-color: blue;
      color: white;
      min-width: 25%;
    }

</style>
</head>

<body>
  <div class="wrapper">
    <span class="item">1</span>
    <span class="item">2</span>
    <span class="item">3</span>
    <span class="item">4</span>
    <span class="item">5</span>
    <span class="item">6</span>
    <span class="item">7</span>
    <span class="item">8</span>
    <span class="item">9</span>
    <span class="item">10</span>
  </div>
</body>

</html>

```

**Ожидаемый результат:**

			10
9	8	7	
6	5	4	
3	2	1	

## Задание 4

Распределите дочерние элементы так, чтоб они заполняли всё пространство по вертикали и имели равные межблочные расстояния.

## Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        height: 100vh;
        min-height: 1000px;
        display: flex;
        background-color: lightgray;
        flex-flow: column;
      }

      .item {
        margin: 5px;
        padding: 10px;
        background-color: blue;
        color: white;
        min-width: 25%;
      }

    </style>
  </head>

  <body>
    <div class="wrapper">
      <span class="item">1</span>
      <span class="item">2</span>
      <span class="item">3</span>
      <span class="item">4</span>
      <span class="item">5</span>
      <span class="item">6</span>
      <span class="item">7</span>
      <span class="item">8</span>
      <span class="item">9</span>
      <span class="item">10</span>
    </div>
  </body>

</html>
```

## Ожидаемый результат:

## Задание 5

Теперь сгруппируйте дочерние элементы в центре родительского блока по вертикали и по горизонтали. Ширина каждого блока должна быть равно 30%.

### Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        height: 100vh;
        display: flex;
        background-color: lightgray;
        flex-flow: column;
        justify-content: space-between;
      }

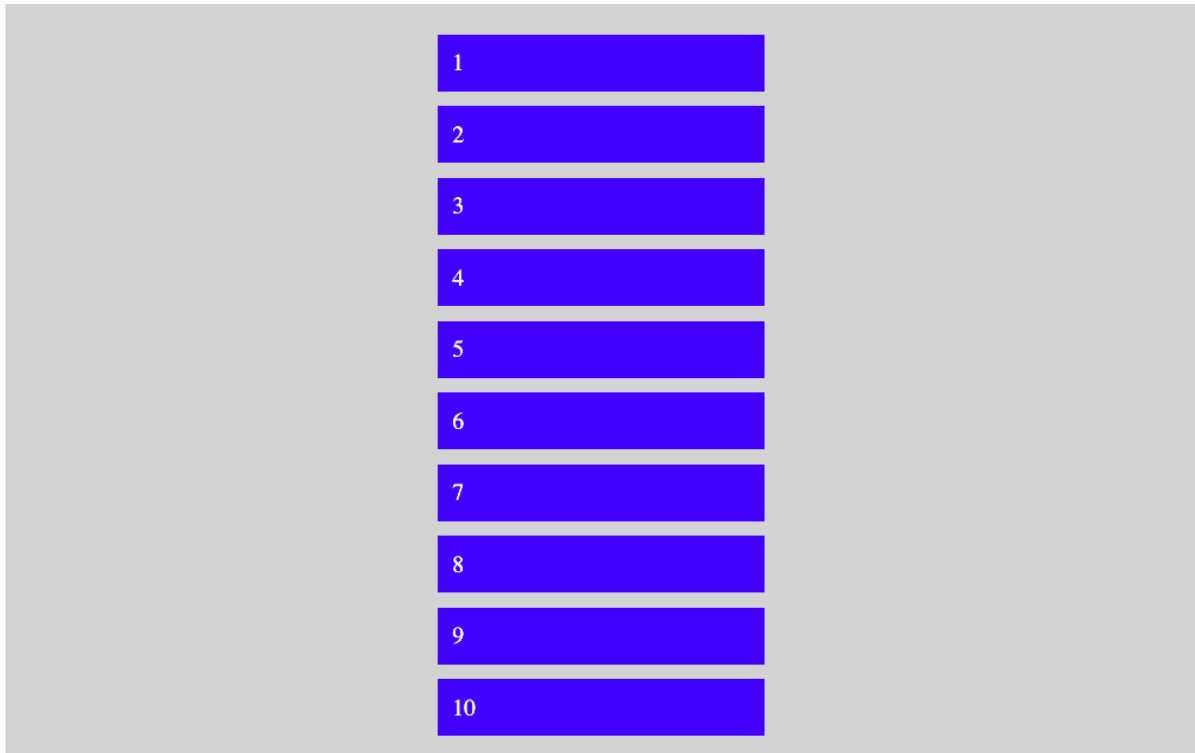
      .item {
        margin: 5px;
        padding: 10px;
        background-color: blue;
        color: white;
        width: 30%;
      }
    </style>
  </head>

  <body>
    <div class="wrapper">
      <span class="item">1</span>
      <span class="item">2</span>
      <span class="item">3</span>
      <span class="item">4</span>
      <span class="item">5</span>
      <span class="item">6</span>
      <span class="item">7</span>
      <span class="item">8</span>
      <span class="item">9</span>
      <span class="item">10</span>
    </div>
```

```
</body>

</html>
```

**Ожидаемый результат:**



## Задание 6

Сгруппируйте все дочерние элементы в центре родительского блока по горизонтали и по вертикали.

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        height: 100vh;
        display: flex;
        background-color: lightgray;
        align-items: flex-start;
        flex-wrap: wrap;
      }
    </style>
  </head>
  <body>
```

```
.item {  
  margin: 5px;  
  padding: 10px;  
  background-color: blue;  
  color: white;  
  min-width: 25%;  
}  
  
</style>  
</head>  
  
<body>  
  <div class="wrapper">  
    <span class="item">1</span>  
    <span class="item">2</span>  
    <span class="item">3</span>  
    <span class="item">4</span>  
    <span class="item">5</span>  
    <span class="item">6</span>  
    <span class="item">7</span>  
    <span class="item">8</span>  
    <span class="item">9</span>  
    <span class="item">10</span>  
  </div>  
</body>  
  
</html>
```

**Ожидаемый результат:**



## Задание 7

Установите порядок дочерних элементов так, чтоб сначала шли нечётные, а потом чётные числа. Используйте свойство `order` и псевдокласс `:nth-child`.

### Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        height: 100vh;
        display: flex;
        background-color: lightgray;
        align-items: flex-start;
        flex-wrap: wrap;
        justify-content: center;
        align-content: center;
      }

      .item {
        margin: 5px;
        padding: 10px;
        background-color: blue;
        color: white;
        min-width: 25%;
      }
    </style>
  </head>
</html>
```

```

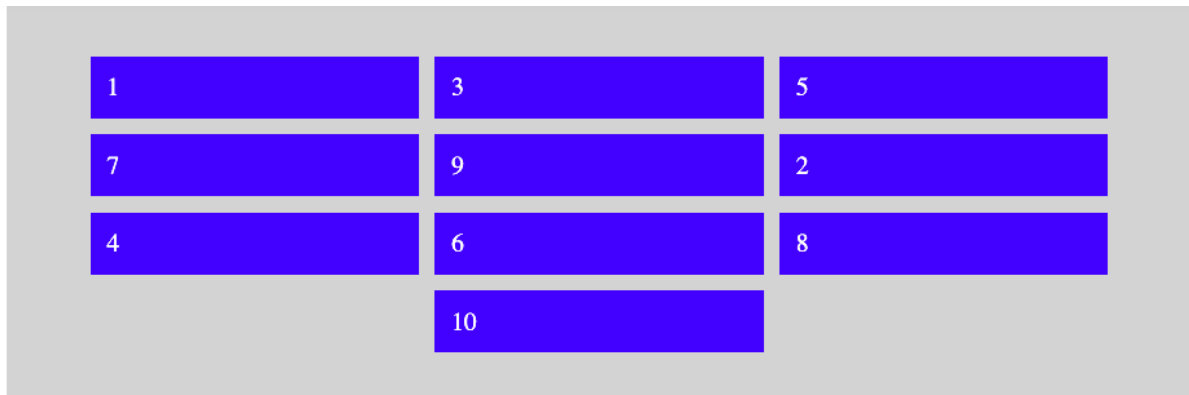
    </style>
</head>

<body>
  <div class="wrapper">
    <span class="item">1</span>
    <span class="item">2</span>
    <span class="item">3</span>
    <span class="item">4</span>
    <span class="item">5</span>
    <span class="item">6</span>
    <span class="item">7</span>
    <span class="item">8</span>
    <span class="item">9</span>
    <span class="item">10</span>
  </div>
</body>

</html>

```

Ожидаемый результат:



## Задание 8

Теперь растяните последний элемент на всю доступную ширину.

Исходный код:

```

<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {

```

```
    height: 100vh;
    display: flex;
    background-color: lightgray;
    align-items: flex-start;
    flex-wrap: wrap;
    justify-content: center;
    align-content: center;
}

.item {
    margin: 5px;
    padding: 10px;
    background-color: blue;
    color: white;
    min-width: 25%;
}

.item:nth-child(even) {
    order: 1;
}

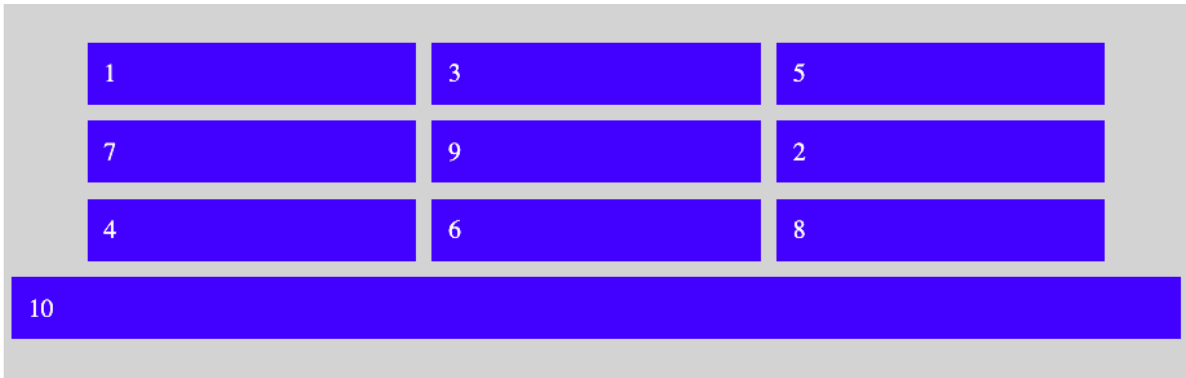
</style>
</head>

<body>
    <div class="wrapper">
        <span class="item">1</span>
        <span class="item">2</span>
        <span class="item">3</span>
        <span class="item">4</span>
        <span class="item">5</span>
        <span class="item">6</span>
        <span class="item">7</span>
        <span class="item">8</span>
        <span class="item">9</span>
        <span class="item">10</span>
    </div>
</body>

</html>
```

**Ожидаемый результат:**





## Задание 9

Установите синему блоку стили так, чтоб его ширина не могла быть меньше заданной.

### Исходный код:

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        height: 20vh;
        display: flex;
        background-color: lightgray;
      }

      .red {
        width: 70%;
        height: 20%;
        background: red;
      }

      .blue {
        width: 70%;
        height: 20%;
        background: blue;
      }

    </style>
  </head>

  <body>
    <div class="wrapper">
      <span class="red"></span>
```

```
        <span class="blue"></span>
    </div>
</body>

</html>
```

**Ожидаемый результат:**



## Задание 10

Задайте синему блоку высоту в 20px и расположите его относительно родительского контейнера в центре по вертикали.

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        height: 20vh;
        display: flex;
        background-color: lightgray;
      }

      .red {
        width: 70%;
        background: red;
      }

      .blue {
        width: 70%;
        background: blue;
        flex-shrink: 0;
      }
    </style>
  </head>
</html>
```

```
    </style>
</head>

<body>
  <div class="wrapper">
    <span class="red"></span>
    <span class="blue"></span>
  </div>
</body>

</html>
```

**Ожидаемый результат:**



## Задание 1

Дополните стили так, чтобы на ширине экрана  $\geq 600\text{px}$  навигация выстраивалась в строку. Ширина блока навигации при этом должна быть 100%, а расстояние между соседними пунктами меню — 50px.

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      ul {
        list-style: none;
        margin: 0;
        padding: 20px;
        background: gray;
        color: white;
        width: 30%;
      }
    </style>
  </head>
</html>
```

```
        </style>
    </head>

    <body>
        <ul>
            <li>0 компании</li>
            <li>Наши услуги</li>
            <li>Ваши отзывы</li>
            <li>Контакты</li>
        </ul>
    </body>

</html>
```

**Ожидаемый результат:**

## Задание 2

Теперь дополните условие так, чтоб на ширине экрана  $\geq 1000\text{px}$  навигация возвращалась к прежнему виду. Нужно именно дополнить существующее условие.

```
<!DOCTYPE html>
<html>

    <head>
        <style>
            ul {
                list-style: none;
                margin: 0;
                padding: 20px;
                background: gray;
                color: white;
                width: 30%;
            }

            @media screen and (min-width: 600px) {
                ul {
                    width: 100%;
                }

                li {
                    display: inline;
```

```

        margin-right: 50px;
    }
}

</style>
</head>

<body>
    <ul>
        <li>0 компании</li>
        <li>Наши услуги</li>
        <li>Ваши отзывы</li>
        <li>Контакты</li>
    </ul>
</body>

</html>

```

**Ожидаемый результат:**

### Задание 3

Задайте условие для блока `.wrapper` так, чтобы при ширине экрана  $\geq 600\text{px}$  блоки с числами выстроились в ряд.

**Исходный код:**

```

<!DOCTYPE html>
<html>

    <head>
        <style>
            .wrapper {
                display: flex;
                background-color: lightgray;
                flex-direction: column;
            }

            .item {
                margin: 5px;
                padding: 10px;
                background-color: blue;
                color: white;
            }

```

```
    </style>
</head>

<body>
  <div class="wrapper">
    <span class="item">1</span>
    <span class="item">2</span>
    <span class="item">3</span>
    <span class="item">4</span>
    <span class="item">5</span>
    <span class="item">6</span>
    <span class="item">7</span>
    <span class="item">8</span>
    <span class="item">9</span>
    <span class="item">10</span>
  </div>
</body>

</html>
```

**Ожидаемый результат:**

## Задание 4

Теперь скройте все блоки с чётными числами при разрешении  $\geq 600\text{px}$  и все с нечётными — при разрешении  $< 600\text{px}$ .

**Исходный код:**

```
<!DOCTYPE html>
<html>

  <head>
    <style>
      .wrapper {
        display: flex;
        background-color: lightgray;
        flex-direction: column;
      }

      .item {
        margin: 5px;
        padding: 10px;
      }
    </style>
  </head>
</html>
```

```

        background-color: blue;
        color: white;
    }

    @media screen and (min-width: 600px) {
        .wrapper {
            flex-direction: row;
        }
    }
}

</style>
</head>

<body>
    <div class="wrapper">
        <span class="item">1</span>
        <span class="item">2</span>
        <span class="item">3</span>
        <span class="item">4</span>
        <span class="item">5</span>
        <span class="item">6</span>
        <span class="item">7</span>
        <span class="item">8</span>
        <span class="item">9</span>
        <span class="item">10</span>
    </div>
</body>

</html>

```

**Ожидаемый результат:**

## Задание 5

Задайте условие для синего квадрата, чтобы при ширине экрана < 600px его высота менялась на 400px. Сделайте изменение плавным, пользуясь свойством transition. Время анимации поставьте .2s.

**Исходный код:**

```

<!DOCTYPE html>
<html>

    <head>

```

```
<style>
  .blue {
    width: 200px;
    height: 200px;
    background: blue;
  }

</style>
</head>

<body>
  <div class="blue">
  </div>
</body>

</html>
```

**Ожидаемый результат:**

I have completed this