

МЕДИА-ФУНКЦИИ



АНТОН ВАРНАВСКИЙ / BRAINIT



АНТОН ВАРНАВСКИЙ

co-основатель и frontend developer в BrainIT



anton.varnauski@gmail.com



[anton_varnavskiy](#)

ПЛАН ЗАНЯТИЯ

1. Взаимодействие с тач-устройствами
2. Медиафункции
3. Единица `em`
4. Flexbox
 - `margin: auto`
 - Отсутствие «margin collapsing»



ВЗАИМОДЕЙСТВИЕ С ТАЧ-УСТРОЙСТВАМИ

СПЕЦИФИКА УСТРОЙСТВ

Мобильные телефоны отличаются от компьютеров: нет видимого курсора — нет и hover-эффектов. В то же время сенсорные экраны мобильных устройств имеют целый набор новых взаимодействий: например, прикосновения (тап, tap), сдвиг (свайп, swipe) и смена масштаба (зум, zoom).

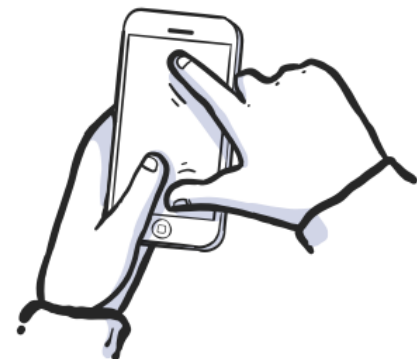
SWIPE



TAP



Zoom



ЭКСПЕРИМЕНТ С HOVER-ЭФФЕКТОМ

Рассмотрим, как по-разному будет выглядеть простой hover-эффект на разных устройствах.

Создадим разметку:

```
1 <div class="picture">
2   
3   <div class="picture__overlay">
4     <span class="picture__text">
5       LOREM IPSUM DOLOR SIT AMET
6     </span>
7   </div>
8 </div>
```

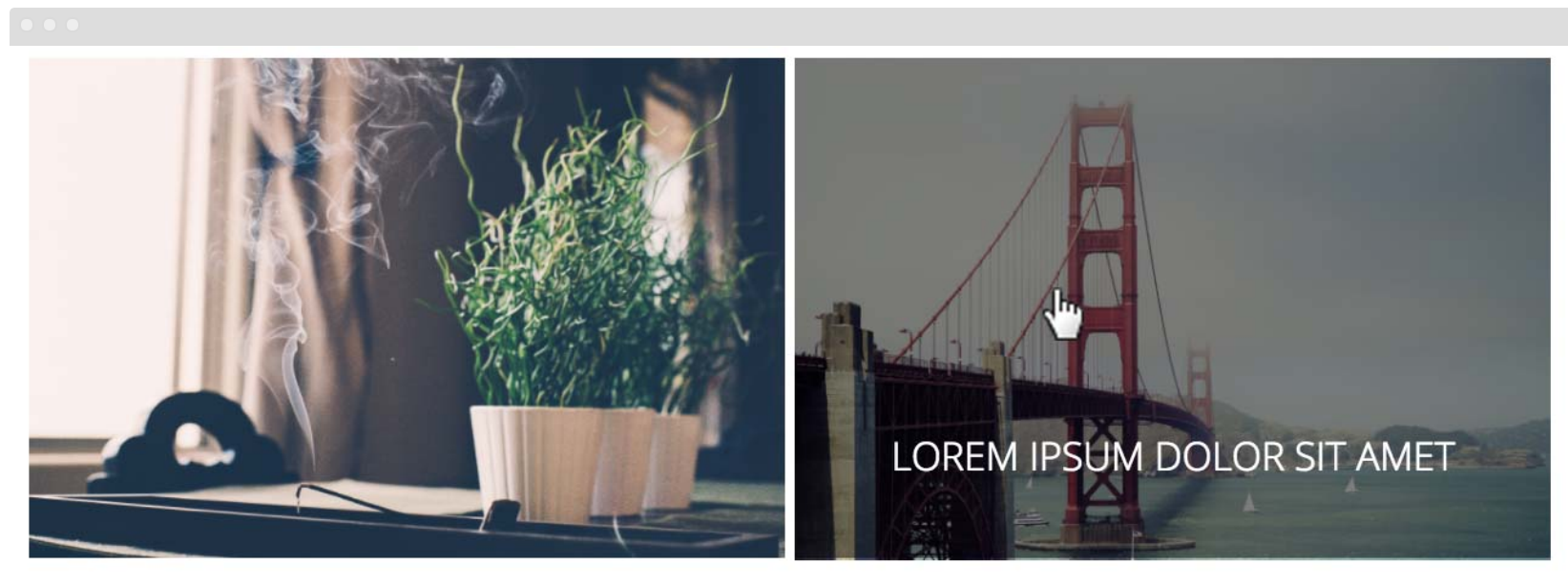
ДОБАВИМ СТИЛИ ДЛЯ `hover`

```
1  .picture__overlay {  
2      opacity: 0;  
3  }  
4  
5  .picture__overlay:hover {  
6      opacity: 1;  
7  }
```

[Live Demo](#)

НАВЕДЕМ КУРСОР МЫШИ

На десктопе hover-эффект появится ожидаемо — при наведении на блок с картинкой:





hover НА ANDROID

На Android для данного блока будет показан эффект, но он появится после тапа и останется, пока не произойдет следующий тап.

hover НА IPHONE

А вот на iPhone поведение будет отличаться. Для исходного примера тап на блочный элемент с картинкой не вызовет hover-эффект — ничего не произойдет.



ПОДГОНЯЕМ РАЗМЕТКУ ПОД IPHONE

Если же мы изменим код таким образом, чтобы текст hover-блока содержал интерактивный элемент — например, ссылку, — то мы добьемся срабатывания hover-эффекта на iPhone:

```
1 <div class="picture">
2   
3   <div class="picture_overlay">
4     <a class="picture_interactive">LOREM IPSUM</a>
5   </div>
6 </div>
```

[Live Demo](#)

НАДО ПОПАСТЬ ПО ССЫЛКЕ

Однако отображаться hover-блок будет лишь тогда, когда при тапе мы попадем пальцем на интерактивный элемент (ссылка `.picture__interactive`). При тапе на любую другую область блока по-прежнему ничего происходить не будет:

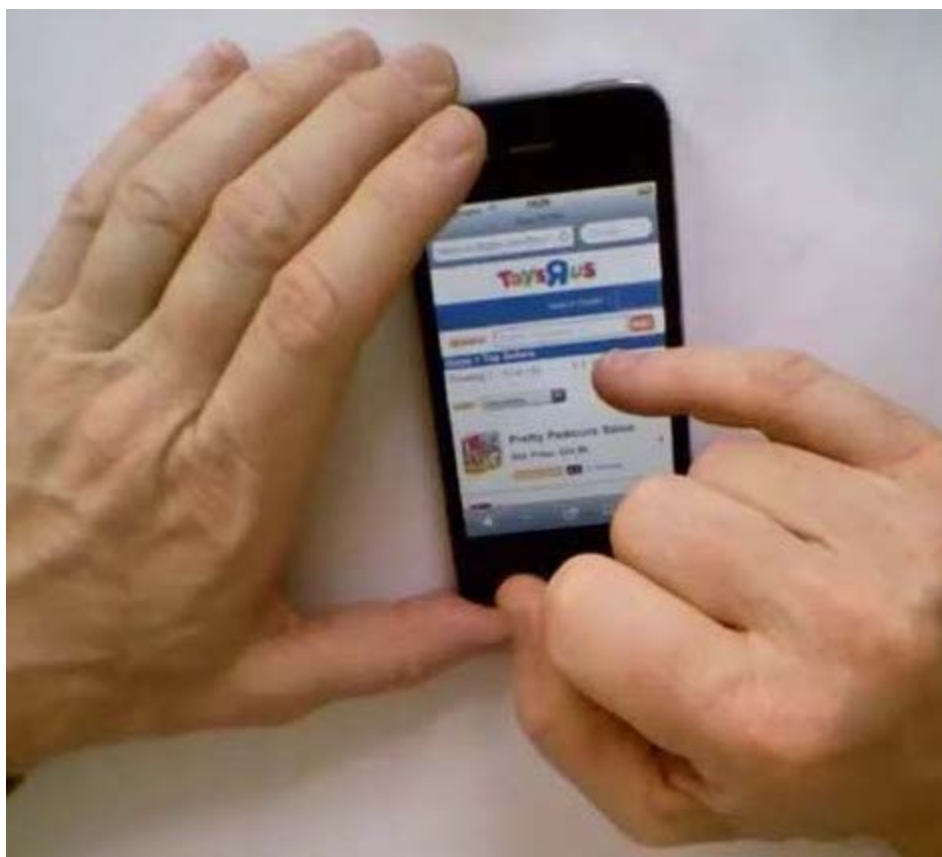


hover ТЕРЯЕТ СМЫСЛ

На сенсорных устройствах смысл hover-эффекта теряется, и интерфейс для тач-устройств должен проектироваться с учетом того, что взаимодействие осуществляется с помощью пальцев.

ОБЛАСТЬ КЛИКА ДЛЯ ПАЛЬЦА

Площадь касания любого пальца намного больше площади курсора мыши. А значит, размер области для клика у сенсорных устройств очень важен, ведь попасть пальцем в определенную точку экрана намного сложнее.



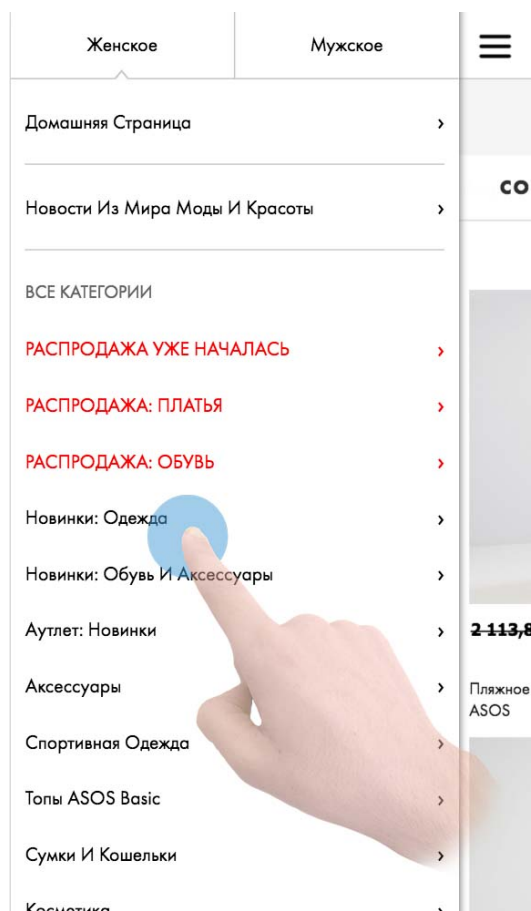
СРЕДНИЙ РАЗМЕР ПАЛЬЦА

Средняя ширина подушечки пальца взрослого человека составляет 10 мм. В рекомендациях по интерфейсу приложений рекомендуется задавать размер активных элементов не менее 7 мм или 48px и не размещать интерактивные элементы слишком близко друг к другу.

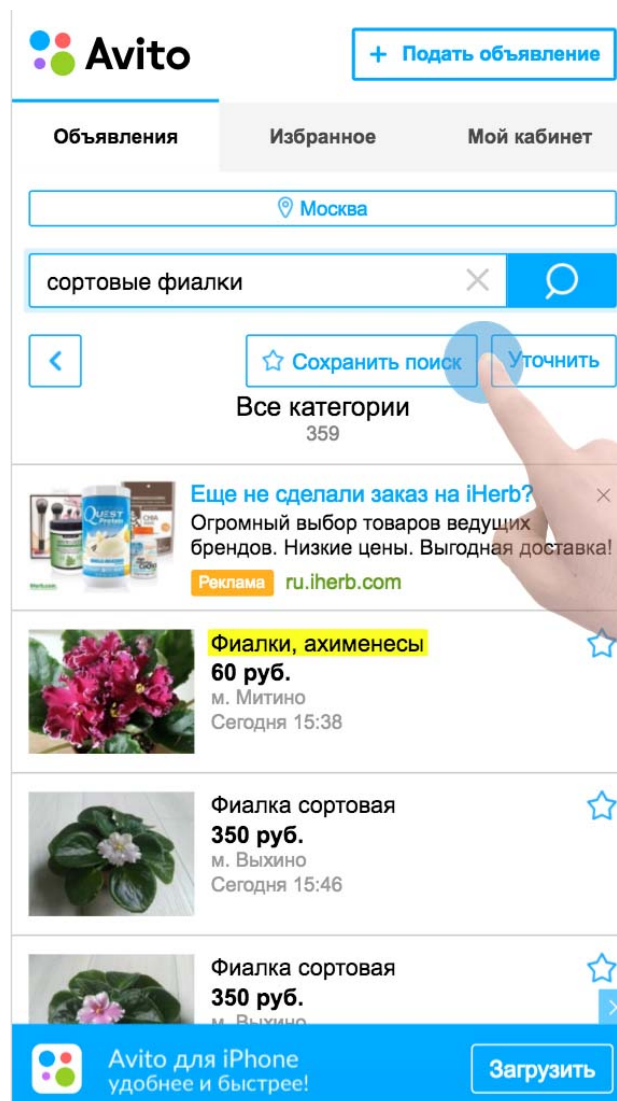


ВЫПАДАЮЩИЕ СПИСКИ НЕЖЕЛАТЕЛЬНЫ

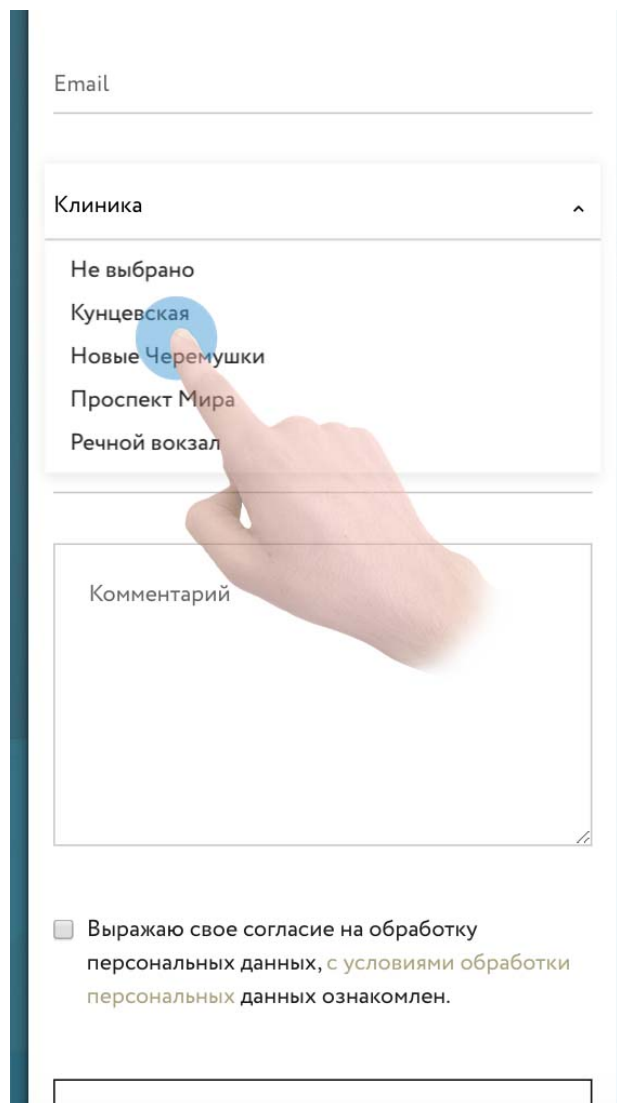
Очень неудобны на мобильных версиях сайтов выпадающие списки (dropdowns, дропдауны) — по ним сложно попасть пальцем, их неудобно прокручивать, и это часто очень раздражает:



БОЛЬШЕ ВЫПАДАЮЩИХ СПИСКОВ!



И СНОВА ДРОПДАУН



Email

Клиника ^

- Не выбрано
- Кунцевская
- Новые Черемушки
- Проспект Мира
- Речной вокзал

Комментарий

☐ Выражаю свое согласие на обработку персональных данных, с условиями обработки персональных данных ознакомлен.

ЧЕКБОКС – АДЕКВАТНАЯ ЗАМЕНА

Во многих случаях дропдауны лучше заменить другими элементами. В случае выпадающего списка из нескольких пунктов подойдет чекбокс:

ИЗБЕГАЙТЕ ЭТОГО

Вы хотите подписаться на уведомления?

Пожалуйста, выберите	▼
Да	
Нет	

ДЕЛАЙТЕ ТАК

☐ Я хочу подписаться на уведомления

или

Push-уведомления



МНОГО ВАРИАНТОВ? РАДИОКНОПКИ!

А если нужен выбор из большого количества вариантов — выпадающий список можно заменить, например, группой радиокнопок:

ИЗБЕГАЙТЕ ЭТОГО

Отрасль

Пожалуйста, выберите	▼
Банковское дело	
IT-технологии	
Медиа	
Телекоммуникации	

ДЕЛАЙТЕ ТАК

Отрасль

<input type="radio"/> Банковское дело
<input type="radio"/> IT-технологии
<input type="radio"/> Медиа
<input type="radio"/> Телекоммуникации
<input type="radio"/> Другое

СЧЕТЧИК ДЛЯ ЦИФР

В случае числового выбора более наглядным и намного более удобным для мобильных устройств будет вариант счетчика с кнопками «+» и «-».

ИЗБЕГАЙТЕ ЭТОГО

Количество пассажиров

Пожалуйста, выберите	▼
1	
2	
3	
4	

ДЕЛАЙТЕ ТАК

Количество пассажиров

-	4	+
---	---	---



ЧЕК-ЛИСТ ДЛЯ ПРИЕМКИ МАКЕТА

- Достаточное расстояние между интерактивными элементами.
- Достаточный размер элементов взаимодействия.
- Оптимальный размер текста и заголовков.
- Отсутствие в дизайне для мобильных устройств выпадающих элементов.



МЕДИАФУНКЦИИ

ДВУХКОЛОНОЧНЫЙ МАКЕТ

Сверстаем страницу вывода статьи с дополнительным блоком новостей для обычных мониторов и мобильных устройств:



[Live Demo](#)

ОТКРЫВАЕМ НА ТЕЛЕФОНЕ

Проверим верстку на телефоне:

Рекорды Мозякина, шоу Гаврилова и другие события «регулярки» КХЛ

Нападающий «Магнитки» сумел обновить несколько бомбардирских рекордов. Осенью он стал самым результативным отечественным снайпером, обогнав по этому показателю легендарного Бориса Михайлова. Затем он перебил рекорды Стива Мозеса и Александра Радулова по количеству забитых голов и набранных очков за «регулярку», а до 1000 баллов за результативность в чемпионатах России ему остался всего один шаг, который он сделает в плей-офф.

Кто бы мог подумать, что в матче команд, которые на тот момент конкурировали между собой за выход в плей-офф, получится такое голевое изобилие! «Витязь» выиграл со счётом 9:6. К третьей минуте форвард хозяев Алексей Макеев уже оформил дубль, а Клинкхаммер и Стась ещё до конца первого периода сравняли счёт. Набросав друг другу ещё четыре шайбы во втором периоде (три в ворота «Динамо» и одну во владения «Витязя»), команды не успокоились и в заключительной двадцатиминутке организовали ещё пять голов. Зрители, которые в тот день до отказа заполнили трибуны дворца в Подольске, остались в приятном шоке от увиденного.

Новости спорта

19:37
«Спартак» из Юрмалы, в который сватали Р. Билялетдинова, возглавил поллак Зуб

19:34
Вальверде вышел на 1-е место по числу матчей в роли главного тренера «Атлетика»



МЕНЯЕМ ПОВЕДЕНИЕ

Очевидно, что на мобильных устройствах мы не можем оставить две колонки. Нам нужно, чтобы они растянулись на всю область просмотра. Здесь возникает потребность определить это мобильное устройство.

Но как это сделать?

ДОСТУПНЫЕ ЗНАЧЕНИЯ

Мобильное устройство отличается от настольного компьютера разрешением экрана. Именно так мы и определим устройство.

width — Описывает ширину области просмотра.

device-width — Определяет всю доступную ширину экрана устройства.

height — Описывает высоту области просмотра.

device-height — Определяет всю доступную высоту экрана устройства.

РАЗНИЦА МЕЖДУ ЗНАЧЕНИЯМИ

Параметры `device-width` и `device-height` определяют ширину и высоту экрана устройства, а `width` и `height` – ширину и высоту области просмотра браузера.

Например, на мониторе 1920x1080 мы можем сделать окно браузера размером 800x600, и, соответственно, область просмотра станет такой же, но разрешение самого экрана останется 1920x1080.

DEPRECATED! (НЕЖЕЛАТЕЛЬНЫЕ)

Кроме того, важно сказать, что параметры `device-width` и `device-height` теперь являются устаревшими и их использование осуждается спецификацией.

@media

Ранее мы рассматривали объявления медиазапроса через атрибут `media` у тега `<link>`. Но кроме такого способа существует CSS-функция `@media`, с помощью которой можно начать использовать запросы.

ВНЕДРЕНИЕ МЕДИАЗАПРОСОВ

В файле стилей объявим медиазапрос с типом `screen`:

```
1 @media screen {  
2     body {  
3         font-size: 14px;  
4     }  
5 }
```

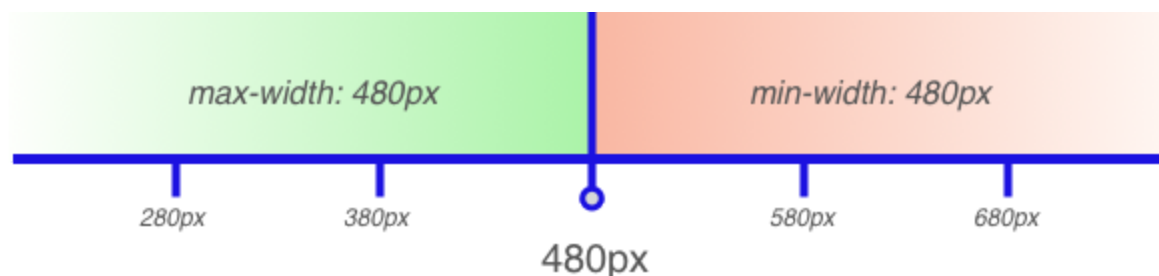
ИСПОЛЬЗУЕМ МЕДИАФУНКЦИЮ

Вместо медиатипа напишем круглые скобки, в которых укажем любой из ранее рассмотренных параметров, а через двоеточие – значение параметра. Например, параметр `width` со значением `480px`:

```
1  @media (max-width: 480px) {  
2      body {  
3          font-size: 14px;  
4      }  
5  }
```


СКРЫВАЕМ ОПИСАНИЕ

С помощью приставки `max-` мы сказали браузеру, что требуется применить CSS только для устройств, у которых ширина области просмотра меньше или равняется `480px`.



viewport

Для того, чтобы браузер корректно работал с параметрами `width`, `height`, `device-width` и `device-height`, нужно правильно настроить область просмотра, или, как называют в англоязычных источниках, `viewport`.

НАСТРОЙКА ОБЛАСТИ ПРОСМОТРА

Для этого у атрибута `name` тега `<meta>` есть специальное значение `viewport`:

```
<meta name="viewport">
```



ТОНКАЯ НАСТРОЙКА

`width` / `height` — С помощью этих параметров можно задать ширину и высоту области просмотра.

Для ширины допустимы значения от `200px` до `10000px` или ключевое слово `device-width`.

А для высоты допустимы значения от `223px` до `10000px` или ключевое слово `device-height`. Крайне важно, что на практике очень редко нужно задавать значение для высоты.

МАСШТАБИРОВАНИЕ

`initial-scale` — Используя `initial-scale`, можно указать начальное масштабирование страницы.

Допустимые значения: от `0.1` до `10`.

ПОЛЬЗОВАТЕЛЬСКИЙ КОНТРОЛЬ

`minimum-scale` / `maximum-scale` — Задавая `minimum-scale` и `maximum-scale`, можно указать минимальное и максимальное значения, в пределах которых пользователь может увеличивать или уменьшать страницу.

Допустимые значения от `0.1` до `10`.

ДАЕМ УВЕЛИЧИВАТЬ?

user-scalable — С помощью параметра можно запретить или разрешить масштабирование страницы.

Допустимые значения: **no** или **yes**.

СТАНДАРТНАЯ НАСТРОЙКА

```
<meta  
  name="viewport"  
  content="width=device-width, initial-scale=1.0">
```


ВЕРНЕМСЯ К ПРИМЕРУ

Ранее для позиционирования колонок мы использовали следующий CSS:

```
1  .page {  
2    max-width: 1200px;  
3    display: flex;  
4    flex-wrap: wrap;  
5  }  
6  
7  .content {  
8    width: 75%;  
9    padding: 28px;  
10 }  
11  
12 .sidebar {  
13   width: 23%;  
14   margin-left: 2%;  
15   padding: 25px;  
16 }
```

ВВОДИМ @media

Мы сохраним две колонки для устройств, у которых ширина экрана больше, чем 481px, а для остальных устройств они встанут друг под другом:

```
1  .page {
2    max-width: 1200px;
3    display: flex;
4    flex-wrap: wrap;
5  }
6
7  @media (min-width: 481px) {
8    .content {
9      width: 75%;
10     padding: 28px;
11   }
12
13   .sidebar {
14     width: 23%;
15     margin-left: 2%;
16     padding: 25px;
17   }
18 }
```

КОЛОНКИ ДРУГ ПОД ДРУГОМ

```
1  @media (max-width: 480px) {  
2    .content, .sidebar {  
3      width: 100%;  
4      padding: 20px;  
5    }  
6  
7    .sidebar {  
8      margin-top: 20px;  
9    }  
10 }
```

ДОБАВИМ viewport

В тег `<head>` добавим настройки `viewport`:

```
1 <head>
2   <meta name="viewport"
3     content="width=device-width, initial-scale=1.0">
4   <link href="style.css" rel="stylesheet">
5 </head>
```

ТЕПЕРЬ ОТЛИЧНО И НА МОБИЛЬНЫХ!

Вот и все, теперь страница корректно отображается как на десктопных устройствах, так и на смартфоне:

Рекорды Мозякина, шоу Гаврилова и другие события «регулярки» КХЛ

Нападающий «Магнитки» сумел обновить несколько бомбардирских рекордов. Осенью он стал самым результативным отечественным снайпером, обогнав по этому показателю легендарного Бориса Михайлова. Затем он перебил рекорды Стива Мозеса и Александра Радулова по количеству забитых голов и набранных очков за «регулярку», а до 1000 баллов за результативность в чемпионатах России ему остался всего один шаг, который он сделает в плей-офф.

Кто бы мог подумать, что в матче команд, которые на тот момент конкурировали между собой за выход в плей-офф, получится такое голевое изобилие! «Витязь»



ЕДИНИЦА **em**

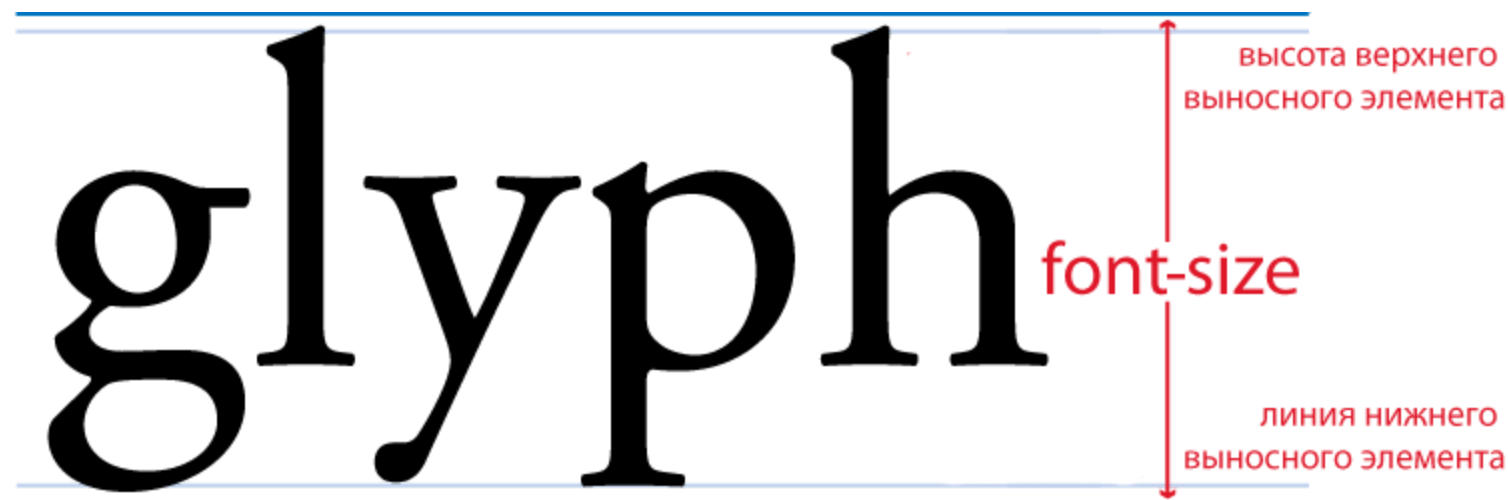


em – еще одна относительная единица, которая широко используется в верстке помимо процентов (%). Она позволяет задавать размеры в зависимости от размера шрифта элемента.



em ДЛЯ font-size

1em для элемента равен размеру шрифта этого самого элемента.



Поэтому в первую очередь `em` очень удобно использовать тогда, когда необходимо, чтобы размер шрифта нескольких элементов находился в зависимости от размера шрифта их общего родительского элемента.

КОРОТКАЯ ЗАМЕТКА

Рассмотрим пример — блок услуги с заголовком и детальным описанием:



Диагностика заболеваний зубов

Комплексная диагностика — обязательное условие любого лечения заболеваний зубов и полости рта. Только на основании тщательных исследований зубочелюстной системы (а возможно и расчетов — в случае ортодонтического и ортопедического лечения) врач сможет ознакомить пациента с оптимальным планом лечения, его длительностью и прогнозируемым конечным результатом.

HTML-РАЗМЕТКА

```
1 <section class="service">
2   <h1 class="service_heading">
3     Диагностика заболеваний зубов
4   </h1>
5   <p class="service_detail">
6     Комплексная диагностика – обязательное условие ...
7   </p>
8 </section>
```

РАЗМЕРЫ ШРИФТОВ

Дизайнер в макете отразил, что при размере шрифта для обычного текста **16px** заголовок должен иметь размер **40px**. Укажем эти значения в файле `style.css`:

```
1  .service {  
2    font-size: 16px;  
3  }  
4  
5  .service_heading {  
6    font-size: 40px;  
7  }
```

ШРИФТ НА МОБИЛЬНЫХ

Далее для устройств с шириной экрана до 480px размер шрифта текста уменьшается до 14px, а заголовок должен иметь размер шрифта 35px:

```
1  .service {
2    font-size: 16px;
3  }
4
5  .service__heading {
6    font-size: 40px;
7  }
8
9  @media (max-width: 480px) {
10    .service {
11      font-size: 14px;
12    }
13
14    .service__heading {
15      font-size: 35px;
16    }
17  }
```

[Live Demo](#)

МЫСЛИ ОБ ОПТИМИЗАЦИИ

Смотря на код, можно прийти к выводу, что он какой-то повторяющийся. Мы несколько раз задаем размеры одним и тем же блокам. Можно ли как-то избавиться от дублирования? 🤔

Конечно, да! Посмотрим, как упростится задача, если для задания размера шрифта мы используем `em`.

СВЯЖЕМ РАЗМЕРЫ ПРИ ПОМОЩИ **em**

Для блока `.service` оставим первоначальный размер шрифта из макета (16px для обычного текста), тогда для заголовка `.service__heading` размер шрифта получится $40 \div 16 = 2.5em$.

```
1  .service {  
2    font-size: 16px;  
3  }  
4  
5  .service__heading {  
6    font-size: 2.5em;  
7  }
```

em ДЛ Я ЭКРАНОВ МЕНЬШЕ 480px

Для устройств с шириной экрана до 480px размер текста в `.service` 14px, а у `.service__heading` – 35px. Разделим размер заголовка на основной шрифт $35 \div 14 = 2.5em$:

```
1  .service {  
2    font-size: 16px;  
3  }  
4  
5  .service__heading {  
6    font-size: 2.5em;  
7  }  
8  
9  @media (max-width: 480px) {  
10   .service {  
11     font-size: 14px;  
12   }  
13 }
```

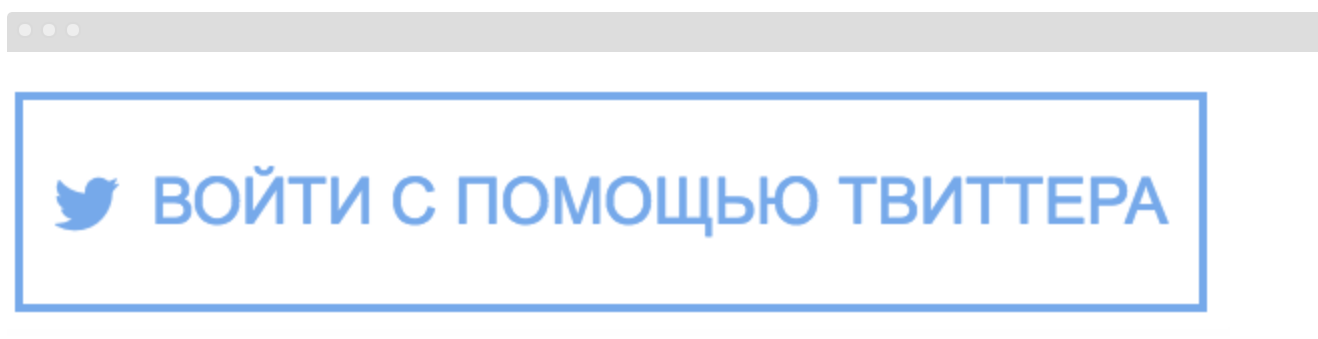
[Live Demo](#)

em ДЛ Я РАЗМЕРОВ

Размер шрифта — не единственное свойство, которое удобно задавать в `em`. Если их задавать в качестве значения для свойств `width`, `height`, `padding`, `margin` и `background-size`, то можно добиться более универсальных решений.

ВЕРСТАЕМ КНОПКУ

В качестве примера сверстаем кнопку входа через твиттер для двух версий — десктопной и мобильной.



РАЗМЕТКА КНОПКИ

```
1 <a href="#" class="button">
2   <span class="button_label">
3     Войти с помощью Твиттера
4   </span>
5 </a>
```

СТИЛИ КНОПКИ ДЛЯ БОЛЬШИХ ЭКРАНОВ

```
1  @media (min-width: 481px) {  
2    .button {  
3      padding: 14px 7px;  
4      font-size: 14px;  
5    }  
6  
7    .button::before {  
8      width: 14px;  
9      height: 14px;  
10     margin-right: 7px;  
11     background-size: 14px;  
12   }  
13 }
```

СТИЛИ КНОПКИ ДЛЯ МАЛЕНЬКИХ ЭКРАНОВ

```
1  @media (max-width: 480px) {  
2    .button {  
3      padding: 16px 8px;  
4      font-size: 16px;  
5    }  
6  
7    .button::before {  
8      width: 16px;  
9      height: 16px;  
10     margin-right: 8px;  
11     background-size: 16px;  
12   }  
13 }
```

УПРОЩАЕМ КОД

Вроде бы мы выполнили нашу задачу. Но давайте подумаем, оптимальное ли это решение? Мы два раза указали значения для одних и тех же свойств, а именно: `width`, `height`, `margin-right`, `background-size`. Можно ли как-то это сделать проще?

Ответ прост — да.

ШИРИНА И ВЫСОТА В `em`

Чтобы перевести `px` в `em` для свойств `width` и `height`, нужно значения свойств поделить на значение `font-size` элемента.

Например, для `.button::before` ширина рассчитывается по формуле:

$$W_{em} = W_{px} \div \text{font-size} = 14px \div 14px = 1$$

Получилось `1em`.

Точно такая же формула будет и для свойства `height`.

СОКРАЩАЕМ КОД

```
1  .button::before {  
2      width: 1em;  
3      height: 1em;  
4  }  
5  
6  @media (min-width: 481px) {  
7      .button {  
8          font-size: 14px;  
9      }  
10 }  
11 @media (max-width: 480px) {  
12     .button {  
13         font-size: 16px;  
14     }  
15 }
```

ПЕРЕСЧИТАЕМ ОСТАЛЬНЫЕ РАЗМЕРЫ

Перерассчитаем свойства: `padding`, `margin` и `background-size`.

Формула не отличается от предыдущего примера:

```
1  .button {
2    padding: 1em 0.5em;
3  }
4
5  .button::before {
6    margin-right: 0.5em;
7    background-size: 1em;
8  }
9
10 @media (min-width: 481px) {
11   .button {
12     font-size: 14px;
13   }
14 }
15
16 @media (max-width: 480px) {
17   .button {
18     font-size: 16px;
19   }
20 }
```


ИТОГОВЫЕ СТИЛИ

```
1  .button {
2    padding: 1em 0.5em;
3  }
4
5  .button::before {
6    margin-right: 0.5em;
7    background-size: 1em;
8    width: 1em;
9    height: 1em;
10 }
11
12 @media (min-width: 481px) {
13   .button {
14     font-size: 14px;
15   }
16 }
17
18 @media (max-width: 480px) {
19   .button {
20     font-size: 16px;
21   }
22 }
```

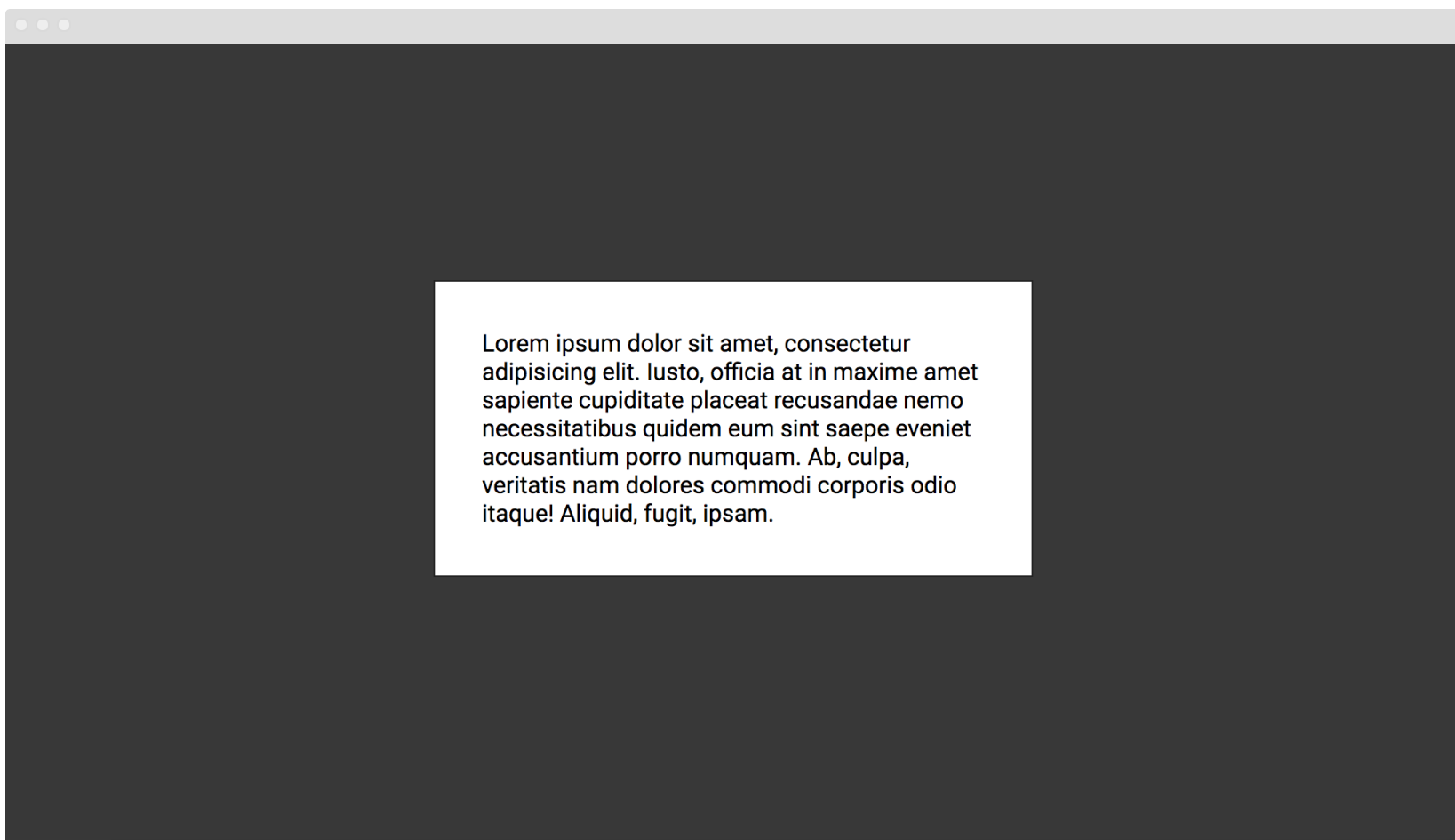
[Live Demo](#)

FLEXBOX:

```
margin: auto;
```

МОДАЛЬНОЕ ОКНО ПО ЦЕНТРУ

Давайте рассмотрим пример центрирования всплывающего окна по горизонтали и вертикали.



ПИШЕМ РАЗМЕТКУ

```
1 <div class="popup-wrapper">  
2   <div class="popup">...</div>  
3 </div>
```

ВЫРАВНИВАЕМ ПО ЦЕНТРУ

Так как окно должно находиться строго по центру, то мы будем использовать фиксированное позиционирование и свойство `transform`:

```
1  .popup-wrapper {  
2    width: 100%;  
3    height: 100%;  
4    position: fixed;  
5    top: 0;  
6    left: 0;  
7    background-color: rgba(0, 0, 0, 0.9);  
8  }  
9  
10 .popup {  
11   max-width: 400px;  
12   position: fixed;  
13   top: 50%;  
14   left: 50%;  
15   transform: translate(-50%, -50%);  
16 }
```

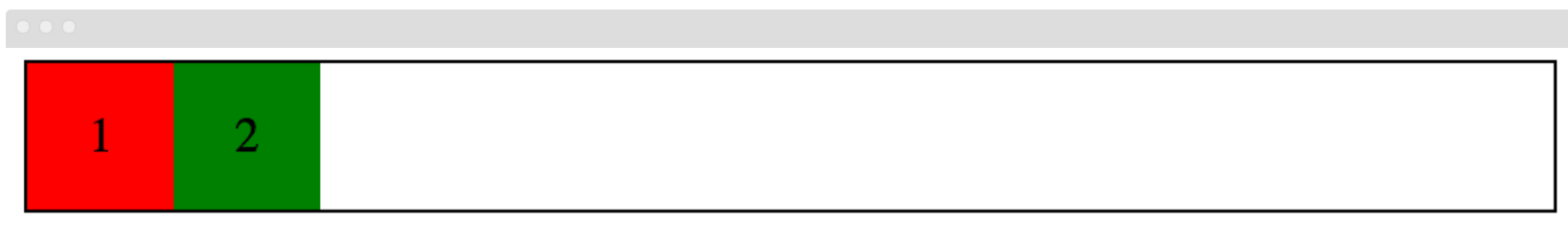
МОЖНО ПРОЩЕ

Вроде бы хороший способ, но он имеет один недостаток. Это использование свойства `transform`.

Но используя флексбоксы, мы можем использовать `margin` для центрирования по всем осям.

ПОТРЕНИРУЕМСЯ «НА КОШКАХ»

Для примера представим, что у нас есть два элемента.



СВОЙСТВА РОДИТЕЛЯ

У их родителя заданы следующие свойства:

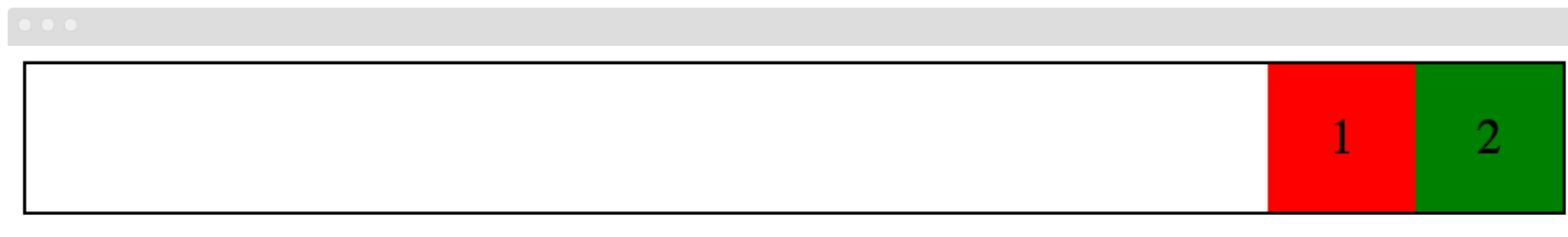
```
1  .container {  
2    display: flex;  
3    width: 500px;  
4    border: 1px solid;  
5  }
```


ОТСТУП ДЛЯ ПЕРВОГО ЭЛЕМЕНТА

Теперь укажем `margin-left: auto` первому элементу:

```
1 | .block:first-child {  
2 |     margin-left: auto;  
3 | }
```

МАКСИМАЛЬНО ВПРАВО



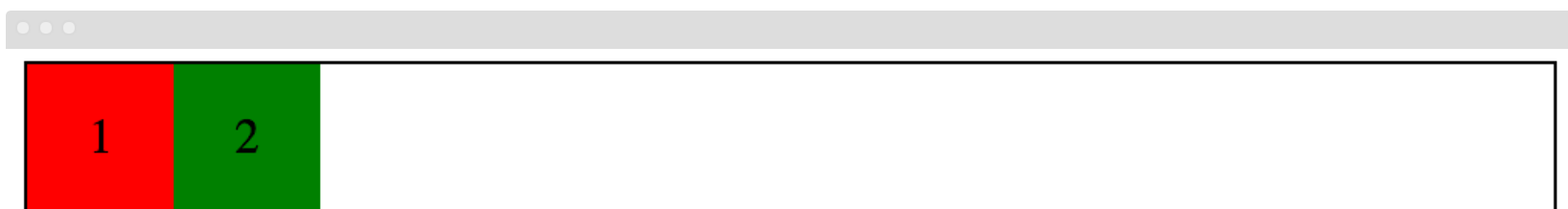
ДВИГАЕМ ВТОРОЙ ЭЛЕМЕНТ

Теперь попробуем указать `margin-right` второму элементу:

```
1 | .block:nth-child(2) {  
2 |     margin-right: auto;  
3 | }
```

ВЕРНУЛИСЬ К НАЧАЛУ

И наши элементы сдвинутся до предела влево.



Вроде бы получилось начальное поведение, но на самом деле второй блок «отталкивается» от правой границы родительского блока, находясь от него на максимально возможном расстоянии.

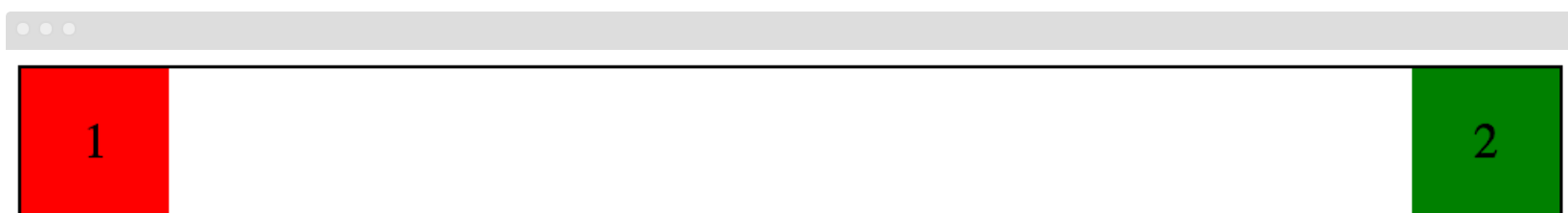
ВЫВОД

Исходя из двух экспериментов, мы можем сделать следующий вывод. Когда мы указываем значение `auto` для `margin`, то браузер вычитает из ширины родителя сумму ширин дочерних элементов и полученную разницу применяет к `margin`.

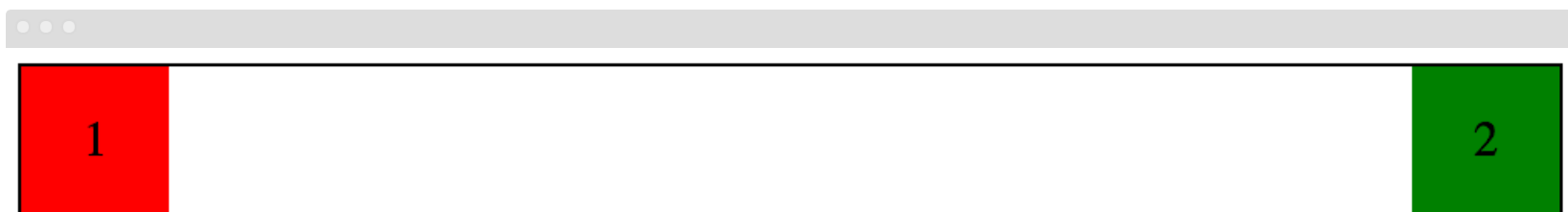
Как мы можем это использовать?

БЛОКИ ПО КРАЯМ

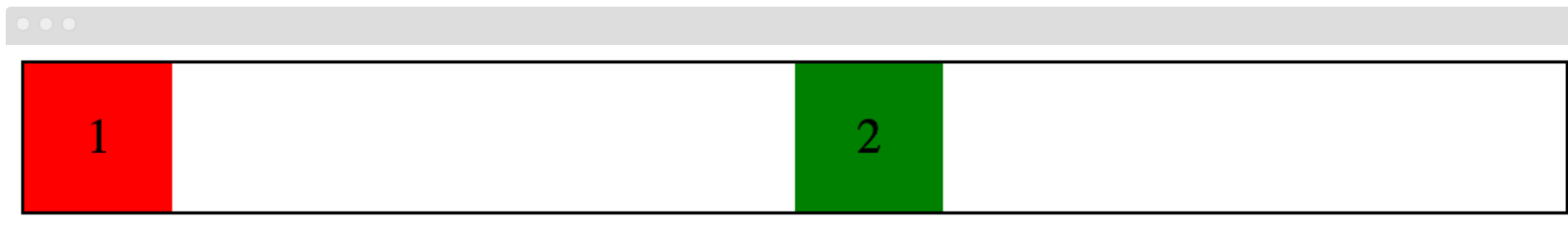
Например, мы можем разнести блоки по краям, добавив первому блоку `margin-right: auto`.



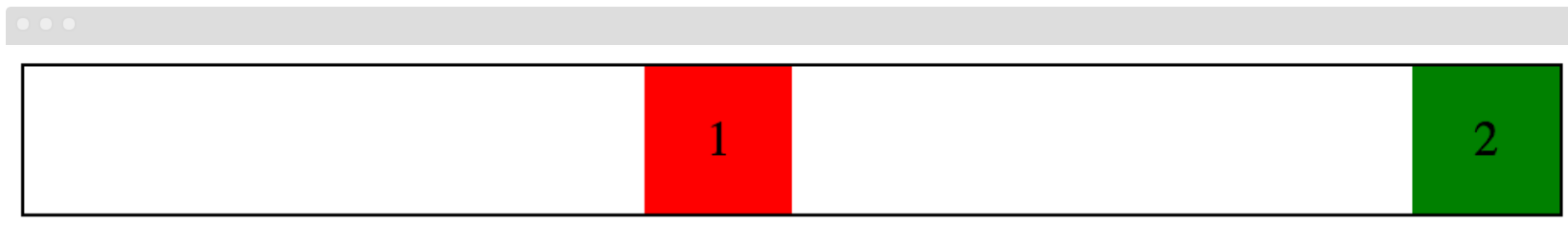
Или `margin-left: auto` второму блоку.



`margin-right: auto` ДЛЯ ОБОИХ

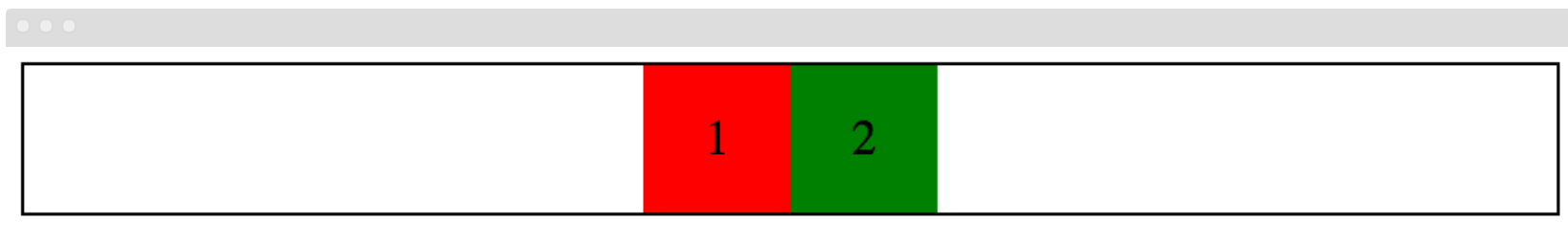


`margin-left: auto` ДЛЯ ОБОИХ



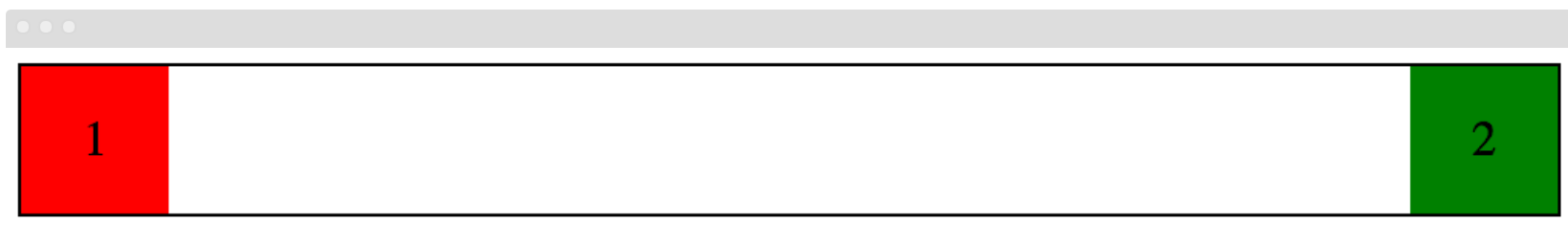
БЛОКИ ПО ЦЕНТРУ

А если указать первому блоку `margin-left: auto` а второму – `margin-right: auto`, мы получим блоки, расположенные по центру их родителя:



ПОСЛЕДНЯЯ КОМБИНАЦИЯ

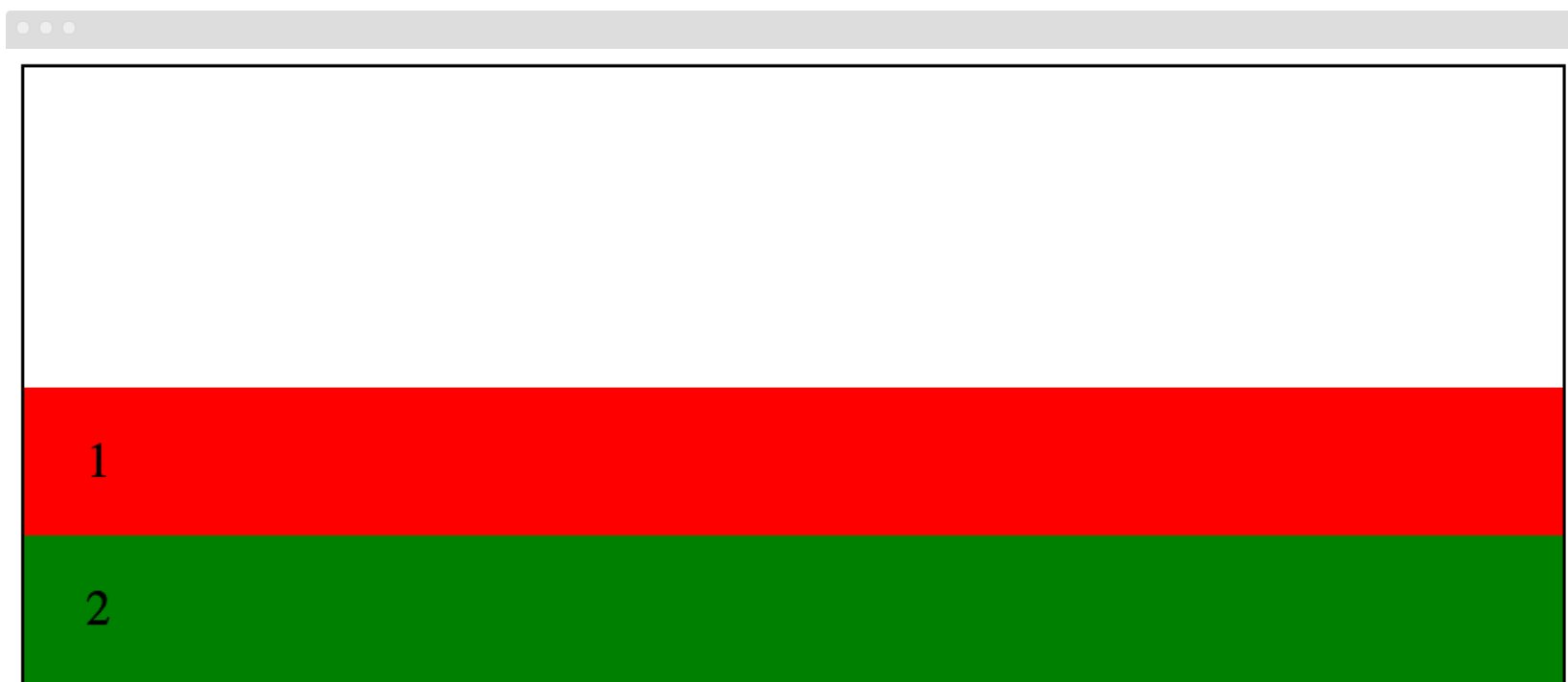
И последний из возможных вариантов — `margin-right: auto` для первого блока и `margin-left: auto` для второго — будет выглядеть так:



ВЕРТИКАЛЬНЫЕ ОТСТУПЫ

Но мы совсем забыли про `margin-top` и `margin-bottom`. Если мы установим `margin-top` и `margin-bottom` значение `auto`, то блок точно так же будет стремиться «оттолкнуться» от верхней или нижней границы родителя или соседнего блока по вертикали.

`margin-top: auto` ДЛЯ ПЕРВОГО
БЛОКА



РАЗНЫЕ ОТСТУПЫ У БЛОКОВ



`margin-bottom: auto` для
ВТОРОГО БЛОКА

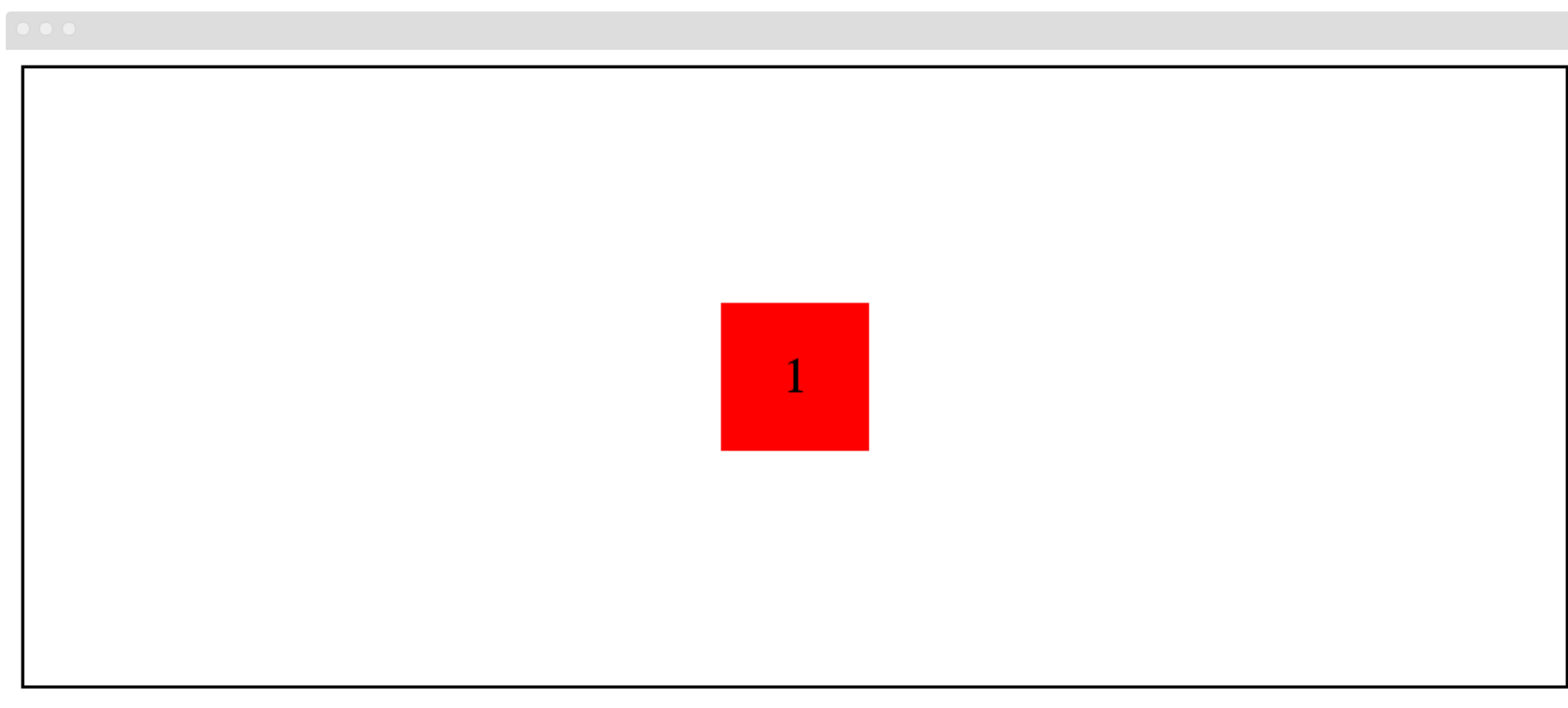


ИТОГ ЭКСПЕРИМЕНТА

Обобщив эти примеры, мы можем сделать вывод, что значение `auto` для `margin` у flex-элемента отодвигает его от ближайшего соседнего блока или границы родителя на максимально возможное расстояние.

ОТСТУПЫ СО ВСЕХ СТОРОН

А вот что будет, если установить `margin: auto` для единственного блока.





ВАЖНЫЙ МОМЕНТ

При использовании этого метода обратите внимание на особенность: браузер должен знать ширину и высоту родительского блока.

[Интерактивная песочница](#)

РЕШАЕМ ЗАДАЧУ БЕЗ transform

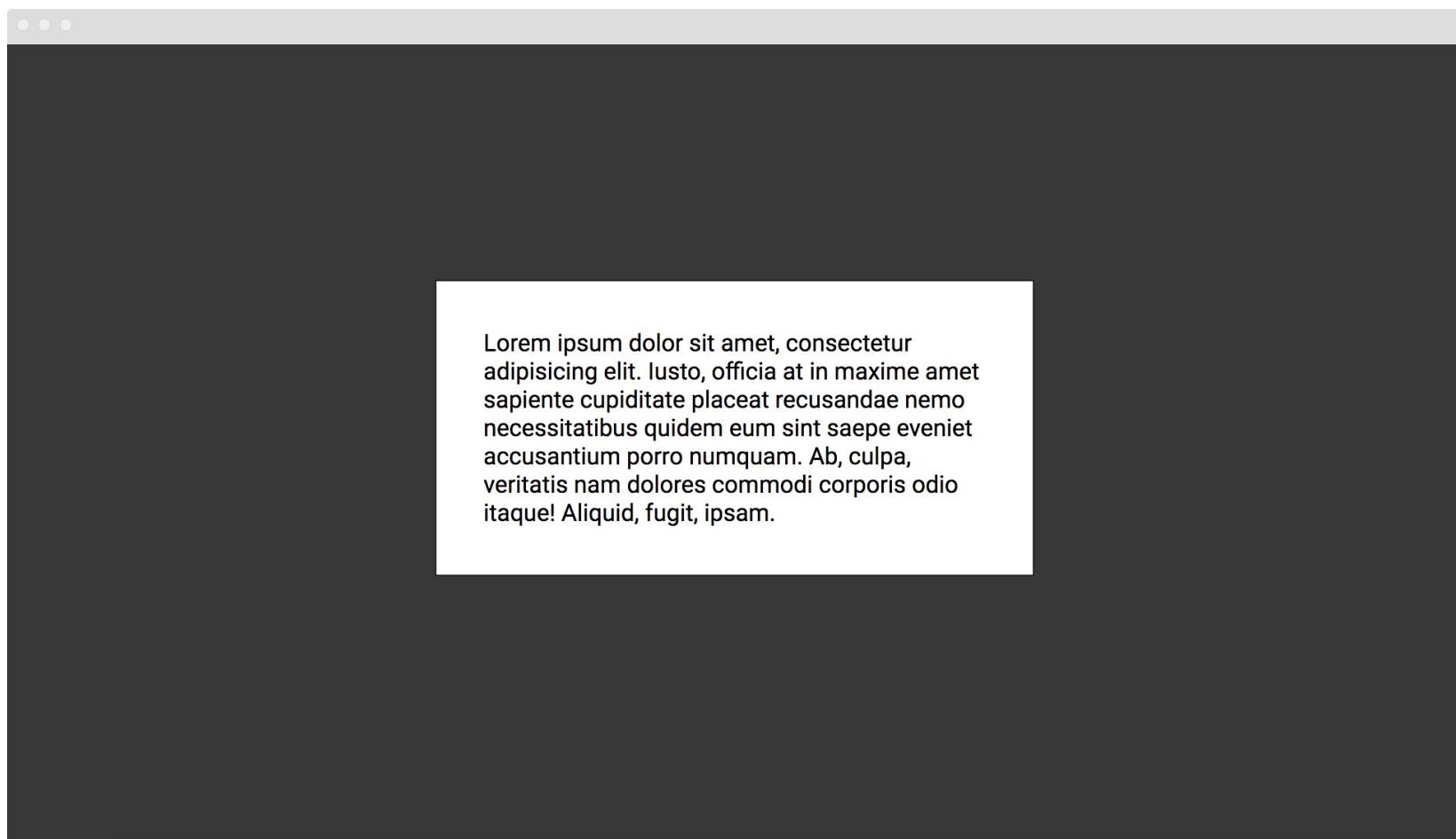
Теперь, зная, как ведет себя flex-элемент с `margin: auto`, мы можем решить задачу с универсальным центрированием всплывающего окна:

```
1  .popup-wrapper {  
2    width: 100%;  
3    height: 100%;  
4    position: fixed;  
5    top: 0;  
6    left: 0;  
7    background-color: rgba(0, 0, 0, 0.9);  
8    display: flex;  
9  }  
10  
11  .popup {  
12    max-width: 400px;  
13    margin: auto;  
14  }
```

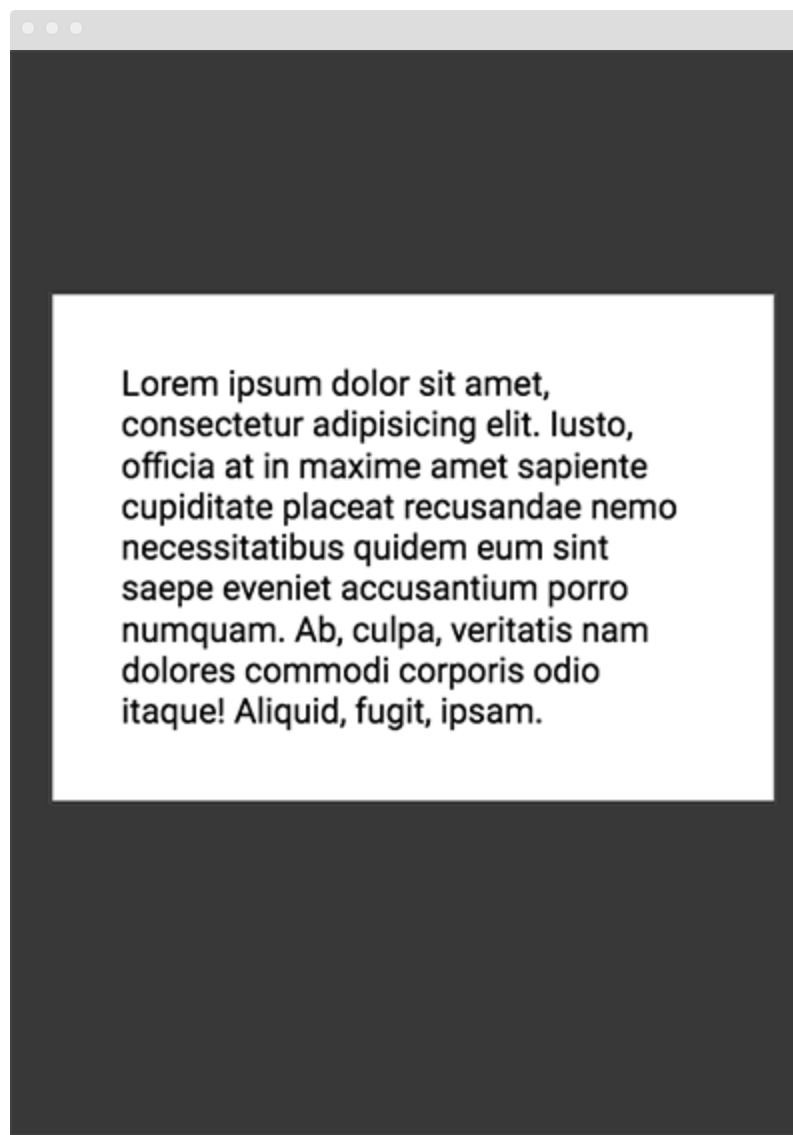
[Live Demo](#)

ПРОВЕРЯЕМ НА ДЕСКТОПЕ

Посмотрим, как верстка будет выглядеть на большом мониторе:



ПРОВЕРЯЕМ НА МОБИЛЬНОМ



«MARGIN COLLAPSING» И ФЛЕКСБОКСЫ

Если мы создадим разметку:

```
1 <div class="container">
2   <p>Нижний отступ этого абзаца схлопнут...</p>
3   <p>...с верхним отступом этого абзаца.</p>
4 </div>
```

И напишем стили:

```
1 p {
2   font-size: 14px;
3   margin-bottom: 28px;
4 }
5 p:nth-child(2) {
6   margin-top: 42px;
7 }
```

...то заметим, что браузер не будет суммировать значения отступов, а просто выберет наибольший (42px).

НАГЛЯДНО



Нижний отступ этого абзаца схлопнут...

...с верхним отступом этого абзаца.

ОТСТУПЫ И FLEX-ЭЛЕМЕНТЫ

Однако, давайте посмотрим, что произойдет, если абзацы будут flex-элементами:

```
1 | .container {  
2 |   display: flex;  
3 |   flex-direction: column;  
4 | }
```

Нижний отступ этого абзаца схлопнут...

...с верхним отступом этого абзаца.



ИТОГИ

ВЗАИМОДЕЙСТВИЕ С ТАЧ-УСТРОЙСТВАМИ

- Специфическая особенность мобильных устройств: нет эффектов наведения (ховер-эффектов). Однако сенсорные экраны имеют иные новые взаимодействия: прикосновение (тап), сдвиг (свайп), смена масштаба (зум).
- При проектировании мобильных интерфейсов рекомендуется задавать размер активных элементов не менее 7 мм или 48px и не размещать интерактивные элементы слишком близко друг к другу.
- Выпадающие меню для мобильных устройств неудобны, поэтому лучше заменить их другими элементами (чекбоксами, группами радиокнопок, счетчиками с кнопками «+» и «-»).

МЕДИАФУНКЦИИ

- Параметры `device-width` и `device-height` определяют ширину и высоту экрана устройства, но являются устаревшими. Параметры `width` и `height` определяют ширину и высоту области просмотра браузера.
- Медиазапросы могут быть объявлены через атрибут `media` у тега `<link>`. Существует и второй способ — CSS-функция `@media`.
- Медиазапрос можно объявить с помощью функции `@media`, используя медиатип либо любой из параметров `width` и `height` с записанным через двоеточие значением параметра.
- Для задания диапазона значений разрешений к параметрам `width` или `height` можно добавлять приставки `max-` и `min-`.

viewport

- Для корректной работы браузера с параметрами `width`, `height`, `device-width` и `device-height` нужно правильно настроить область просмотра – `viewport`. Для этого у атрибута `name` тега `<meta>` есть специальное значение `viewport`.
- Параметры `width` / `height` позволяют задать ширину и высоту области просмотра.
- `initial-scale` позволяет указать начальное масштабирование страницы.
- `minimum-scale` и `maximum-scale` позволяют указать минимальное и максимальное значения, в пределах которых пользователь может масштабировать страницу.
- `user-scalable` позволяет запретить или разрешить масштабирование страницы.

ЕДИНИЦА `em`

- `em` — относительная единица измерения, используемая в верстке помимо процентов (%). Позволяет задавать размеры в зависимости от размера шрифта элемента.
- `1em` для элемента равен `font-size` этого самого элемента. Поэтому `em` удобно использовать, когда нужно, чтобы размер шрифта нескольких элементов зависел от размера шрифта их общего родительского элемента.
- Если в `em` задавать значения для свойств `width`, `height`, `padding`, `margin` и `background-size`, то можно добиться более универсальных решений.
- Для перевода `px` в `em` для свойств `width` и `height` нужно значения свойств разделить на значение `font-size` элемента.

FLEXBOX: `margin: auto`, «MARGIN COLLAPSING»

- Используя флексбоксы, мы можем использовать `margin` для центрирования по всем осям.
- При значении `auto` для `margin` браузер вычитает из ширины родителя сумму ширин дочерних элементов и полученную разницу применяет к `margin`.
- Таким образом, значение `auto` для `margin` у flex-элемента отодвигает его от ближайшего соседнего блока или границы родителя на максимально возможное расстояние.
- Важно: при использовании этого метода браузер должен знать ширину и высоту родительского блока.
- Другая особенность: если абзацы являются flex-элементами, то браузер будет суммировать значения внешних отступов вместо их «схлопывания».



НЕТОЛОГИЯ
университет интернет-профессий

Задавайте вопросы и напишите отзыв о лекции!

АНТОН ВАРНАВСКИЙ



anton.varnauski@gmail.com



[anton_varnavskiy](https://t.me/anton_varnavskiy)