**Разработка интерфейса для разных устройств. Введение**

Современные устройства позволяют моментально получать информацию. Но кроме устройства пользователю нужен удобный интерфейс. Задача разработчика — сделать интерфейс понятным и удобным на любом устройстве.

Пользователи теряют интерес к проекту, если блоки «едут», текст плохо читается, а кнопки не реагируют на нажатия. Поэтому, чтобы не терять посетителей, сайт должен быть отзывчивым и адаптивным.

Вы уже изучили технологии построения гибких сеток; знаете, что такое отзывчивость на уровне каркаса сайта. Поэтому первая часть задачи решена. Осталось научиться адаптировать блоки и элементы под различные устройства.

В этой теме вы обобщите знания о создании «резиновых» блоков. Многие концепции вам уже знакомы: например, работа с относительными величинами. Нужно только собрать всё вместе. Эти знания вы пополните новыми: работой с вычисляемыми значениями и особенностями отступов в относительных величинах. Так у вас появятся инструменты, позволяющие сделать гибким любой объект на сайте.

Поговорим также об изображениях и шрифтах. Чтобы графика и типографика смотрелись классно на экранах с высоким разрешением, приходится прибегать к хитростям.

В финале темы разберём самый важный инструмент создания адаптивного сайта — медиазапросы. Они позволяют менять CSS-стили, когда экран устройства уменьшается или увеличивается. Именно появление медиазапросов в своё время сделало веб адаптивным. После прохождения темы смело сможете добавлять в резюме пункт «адаптивная вёрстка».

Но изучить инструменты в тренажёре — это половина дела. В практической части спринта вам предстоит потрудиться, чтобы применить всё изученное в одном макете.

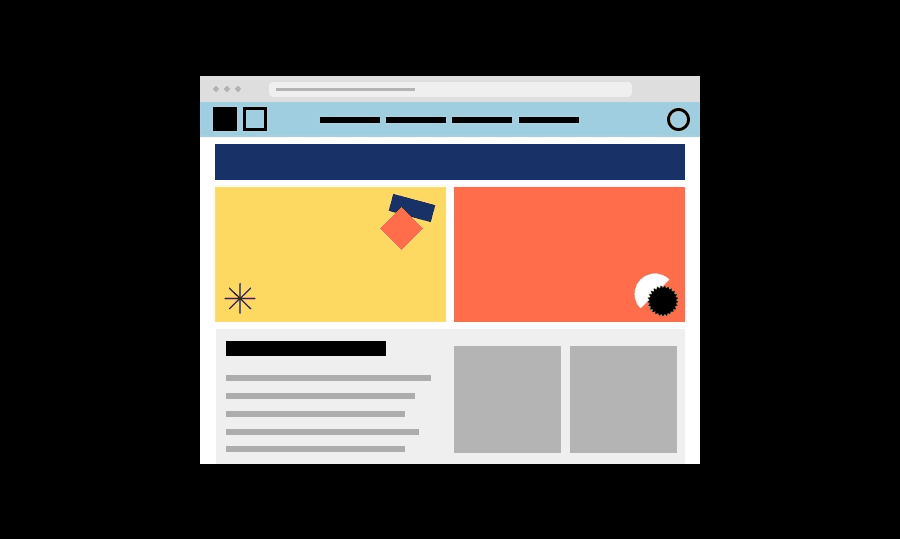
# Относительные размеры блоков

Обобщим сведения об уже знакомых вам единицах измерения, которые делают макет «резиновым».

### **Минимальные и максимальные значения размеров**

Свойства min-width, min-height, max-width и max-height — очень серьёзный инструмент «резиновой» вёрстки. Вот несколько вопросов, которые нужно обсудить с дизайнером, прежде чем приступать к вёрстке макета:

* Насколько будет сжиматься вся страница? В смысле, до какого предела сжатия окна браузера элементы будут уменьшаться пропорционально в ширину. После достижения этого предела элементы могут либо перестать сжиматься (и появится скролл) либо радикально изменить своё положение/поведение. В этом случае используют min-width.
* До какого предела могут расширяться страница или области контента? Есть много вариантов поведения. Обычно «резиновая» вёрстка предусматривает максимальную ширину, свыше которой элементы перестают растягиваться, а по бокам появляются поля (на веб-дизайнерском жаргоне — «уши»). Например, посмотрите, как ведёт себя страница сайта [Нобелевской премии](https://www.nobelprize.org/alfred-nobel/) — в шапке и в основной зоне контента на широких мониторах. Если изображение Альфреда Нобеля тянется бесконечно, то зона контента на определённой ширине растягиваться перестаёт. Подобное поведение обеспечивается свойством max-width в сочетании с горизонтальным выравниванием через margin: auto;
* Как должны менять свой размер текстовые блоки? Помните, что высота текстовых элементов может меняться при сжатии браузера или увеличении объёма текста. С одной стороны это значит, что самим текстовым элементам нужно задавать минимальную высоту вместо фиксированной, а с другой — родительские элементы тоже должны растягиваться при увеличении высоты контента. Вспоминайте это правило каждый раз, когда пишете свойство height, и задумывайтесь, не лучше ли использовать min-height.
* Как элементы будут смотреться на очень больших экранах? Там размеры могут оказаться слишком велики. Например, вы задали основной картинке статьи min-height: 80vh;. На большом экране или проекторе изображение станет гигантским, а вам это не подходит. Не стесняйтесь в нужном месте использовать max-height.



### **Проценты:**

Размеры элементов можно задавать в процентах от размера родительского элемента. Особенность размеров, указанных в процентах: их часто невозможно применить к высоте. Для высоты это сработает только во flex-контейнере или когда высота родителя явно задана. Не забывайте об этом правиле, иначе поначалу вы можете запутаться.

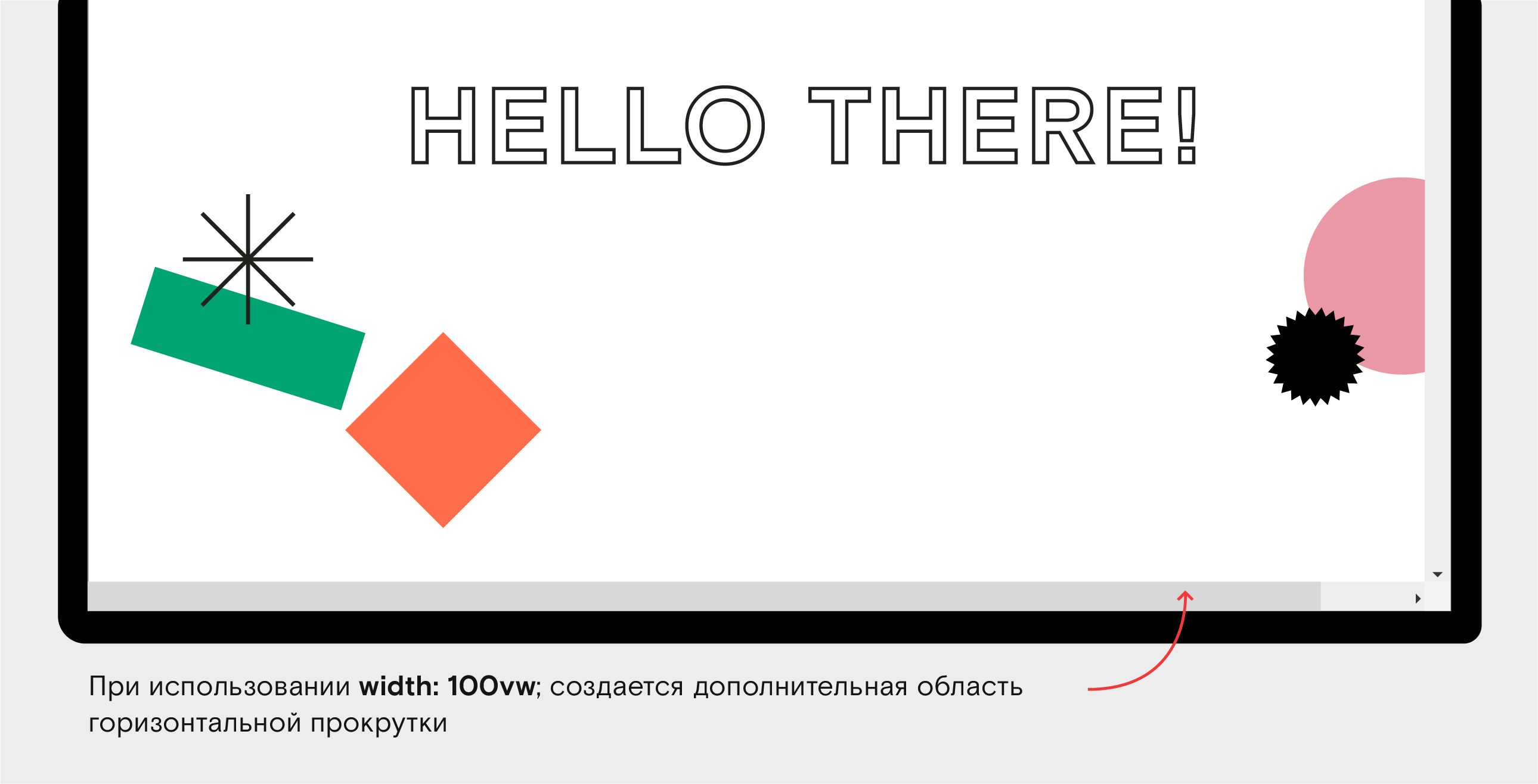
### **vh и vw:**

Эти единицы измерения привязаны к размерам окна браузера, а не родительским элементам. Измеряются в числах от 0 до 100.

100vw — вся ширина окна браузера, включая полосу скролла, 1vw — это 1% от ширины окна браузера.

100vh — вся высота окна браузера, 1vh равно 1% высоты окна браузера.

Правила вроде width: 100vw; и height: 100vh; используйте очень осторожно. Это же размер всего окна, включая полосы скролла, а в мобильных устройствах ещё и вспомогательные элементы навигации. В одних браузерах скролл виден по умолчанию, в других — нет. Да и ширина самого скролла может быть разной. В некоторых браузерах значения 100vh или 100vw могут создать дополнительные области прокрутки, а значит, лучше предпочесть им величины в процентах.



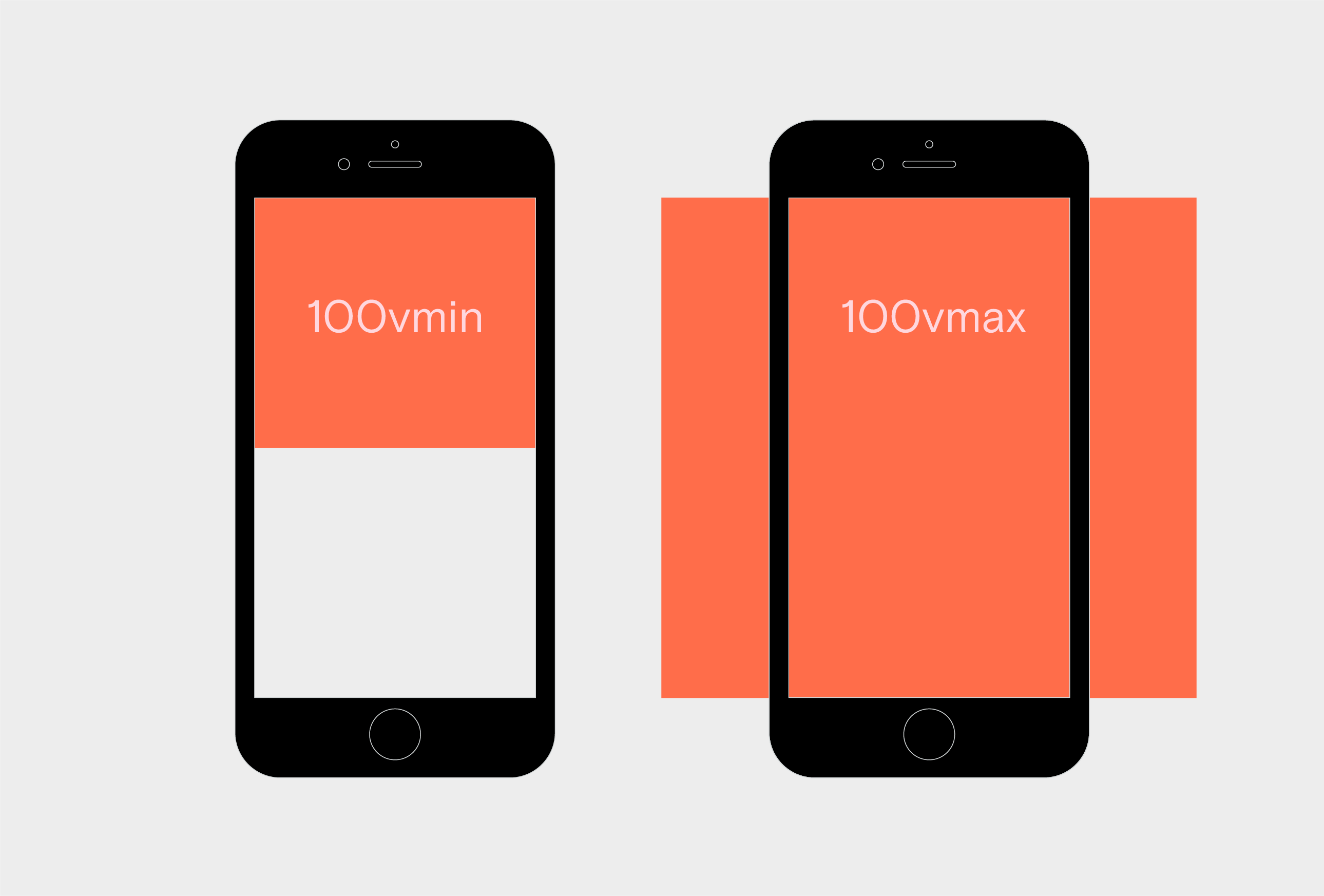
### vmin и vmax:

Фактически это проценты либо от высоты, либо от ширины браузера: в зависимости от того, какая из этих величин больше. Единицы vmin (от англ. viewport minimum, «минимальный размер окна просмотра») — это размер в процентах от меньшей величины, а vmax — от большей.

Если сайт открыт на компьютере, то есть ширина больше высоты, единицы vmin будут считаться от высоты, а vmax — от ширины. И наоборот: если сайт открыт с телефона, то есть больше высота, vmin будут взяты от ширины, а vmax — от высоты.

Так, 50vmin на горизонтально ориентированном устройстве задаёт размер в 50% высоты окна браузера, а на вертикально ориентированном — 50% ширины.

С величиной vmax всё наоборот: браузер выбирает из собственной ширины и высоты максимум, чтобы относительно него рассчитать размер. 50vmax на горизонтально ориентированном устройстве — это 50% ширины окна браузера, а на вертикально ориентированном — половина высоты.



Эти единицы измерения не слишком часто встретишь в реальном проекте, но они могут вам пригодиться.

**Вычисляемые значения, функция calc()**

До этого мы говорили о CSS только как о языке инструкций, описывающих, как отрисовывать элементы. Но современный CSS может больше — прямо как в JavaScript, он тоже имеет встроенные функции.

Функция calc() (от англ. calculation, «вычисление») служит для расчёта, как правило, размеров. Особенно полезна тем, что может производить вычисления с комбинациями относительных и абсолютных величин. Её синтаксис:

Скопировать кодCSS

selector { */\*любой селектор\*/*

width: calc(<выражение>);

}

Функция работает с любыми единицами измерения и способна выполнять 4 операции: сложение, вычитание, умножение и деление.

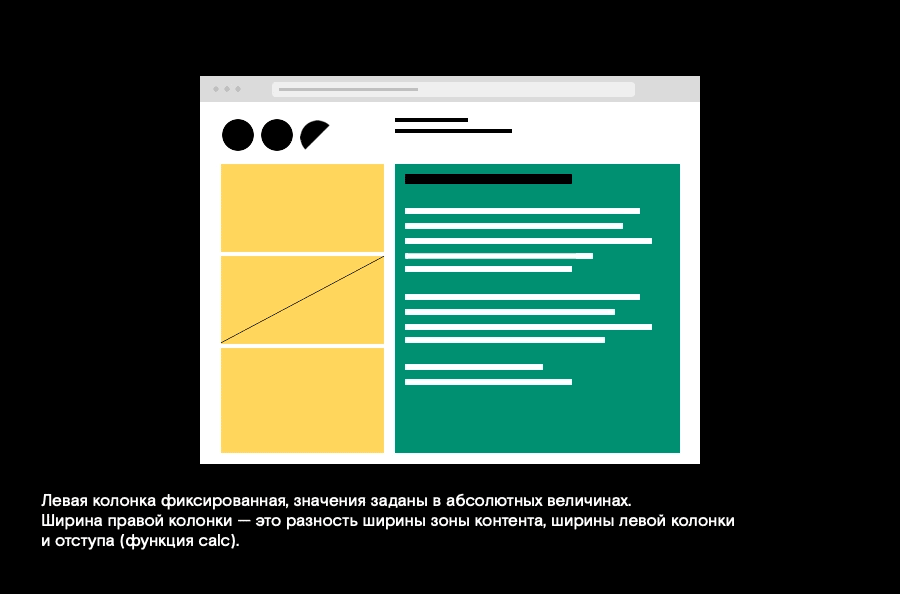
Скопировать кодCSS

selector {

width: calc(100%/3);

height: calc((100vh + 300px)/2);

}



Внимание: операторы ʻ+ʼ и ʻ-ʼ работают корректно, если отделены пробелами. Для умножения и деления это не обязательно.

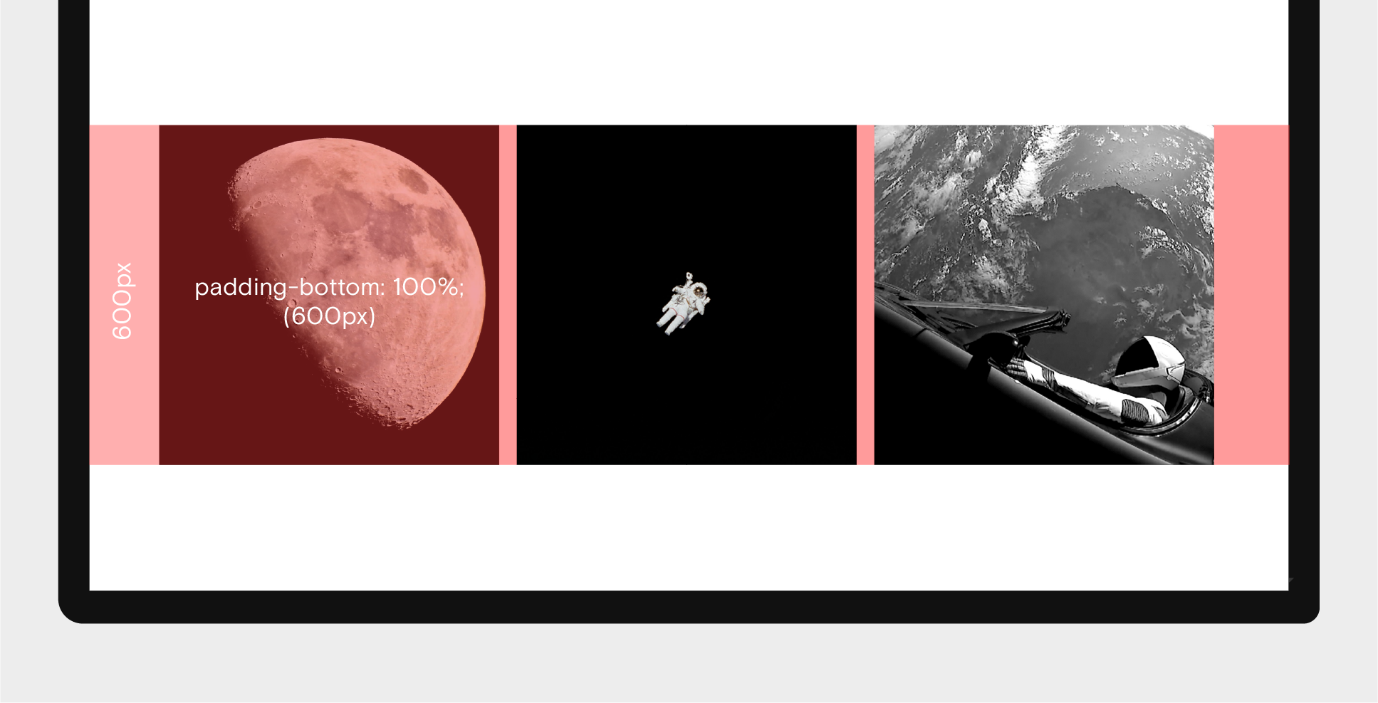
Функция calc() пригождается, например, когда:

* одна колонка имеет фиксированную ширину, а остальные — резиновые. Такое часто встречается на сайтах с рекламными баннерами на полях.
* сами колонки растягиваются, а расстояние между ними фиксированное.
* дочерний элемент ýже родительского на фиксированное количество пикселей.

# Размеры margin и padding в процентах

У внешних и внутренних отступов есть важная особенность, которую часто используют при создании «резиновых» макетов. Указав размер отступа в процентах, мы получим величину, рассчитанную от ширины родительского блока.

Если родительский блок имеет ширину 600px, а внутренний отступ дочернего элемента указан как 50%, то физически размер отступа составит 300px. Обратите внимание: все внутренние отступы — в том числе и вертикальные — рассчитываются от ширины родителя. Так же работает и внешний отступ, но именно особенность padding помогает сделать высоту блока «резиновой».



Рассмотрим на примере. Есть HTML-файл с двумя блоками:

Скопировать кодHTML

<div class="block">

<div class="block\_\_image"></div>

</div>

Каждому классу назначены правила CSS:

Скопировать кодCSS

.block {

width: 30%;

}

.block\_\_image {

width: 100%;

}

Предположим, нужно сделать элемент block\_\_image квадратным, причём с сохранением пропорций, когда окно браузера сжимается по горизонтали и вертикали. Установка высоты в 30% нам не поможет, ведь высота не наследуется или наследуется от высоты родителя. Другие единицы измерения здесь тоже бессильны.

Тут и выручает внутренний отступ:

Скопировать кодCSS

.block {

width: 30%;

}

.block\_\_image {

width: 100%;

padding-bottom: 100%;

}

Элемент block\_\_image будет растянут внутренним отступом на всю ширину своего родителя. Этот трюк позволил реализовать блок, резиновый и по горизонтали, и по вертикали.

# Относительные размеры текстовых элементов

У шрифтов тоже есть относительные размеры. С ними можно пропорционально масштабировать текстовые элементы в блоках или на странице. Но эти единицы измерения не делают шрифты резиновыми. Их размеры просто вычисляют относительно других размеров.

Когда-то давно браузеры не умели правильно масштабировать текст с размерами в пикселях. Поэтому появились относительные величины. Современные браузеры отлично работают с пикселями, но миф о более адаптивных величинах em и rem сохранился.

Попрактикуйтесь с этими размерами. Интернет заполнен мануалами с их использованием, а некоторые команды разработки принципиально не применяют пиксели. Вдруг вы попадёте в такую команду на собеседование. Но не используйте em и rem в наших проектных работах — жизнь программиста и так сложна.

Мы нашли для вас [перевод ироничной статьи о сложившейся ситуации](https://medium.com/devschacht/david-gilbertson-rems-and-ems-and-why-you-probably-dont-need-them-3b2b1e785787). Автор во многом прав, хоть и резок в суждениях. Прочтите её, когда пройдёте тему целиком: тогда вы лучше её поймёте.

### **Единица измерения em**

Единица измерения, пришедшая из типографского дела. В старых шрифтах семейства «антиква» самой широкой буквой латинского алфавита была заглавная «M». Слово em и есть английское название этой буквы. Её ширина равнялась высоте, что делало её удобной единицей измерения, и ширину остальных букв рассчитывали в долях «M».

В CSS значение размера шрифта в единицах em высчитывается относительно размера шрифта родительского элемента. Если у родителя это 18px, то 1em для дочерних элементов равен 18px, а 0.5em — 9px. Например:

Скопировать кодCSS

.parent {

font-size: 24px;

}

.child {

font-size: 2em;

}

Здесь размер шрифта у дочернего элемента .child вдвое больше, чем у родителя .parent, и составляет 48 пикселей.

### **Единица измерения**rem

Rem расшифровывается как root em, «корневой» em. При установке размеров в rem за единицу принимают размер шрифта корневого объекта документа — тега <html>. Если размер шрифта для него не установлен явно, то во всех современных браузерах 1rem — это 16 пикселей, базовый размер шрифта. Если задать правило для тега html, единица 1rem станет равна установленному ему значению свойства font-size, а 0.5rem — половиной этого значения, независимо от размера шрифта родительского блока. Например:

Скопировать кодCSS

html {

font-size: 30px;

}

.parent {

font-size: 40px;

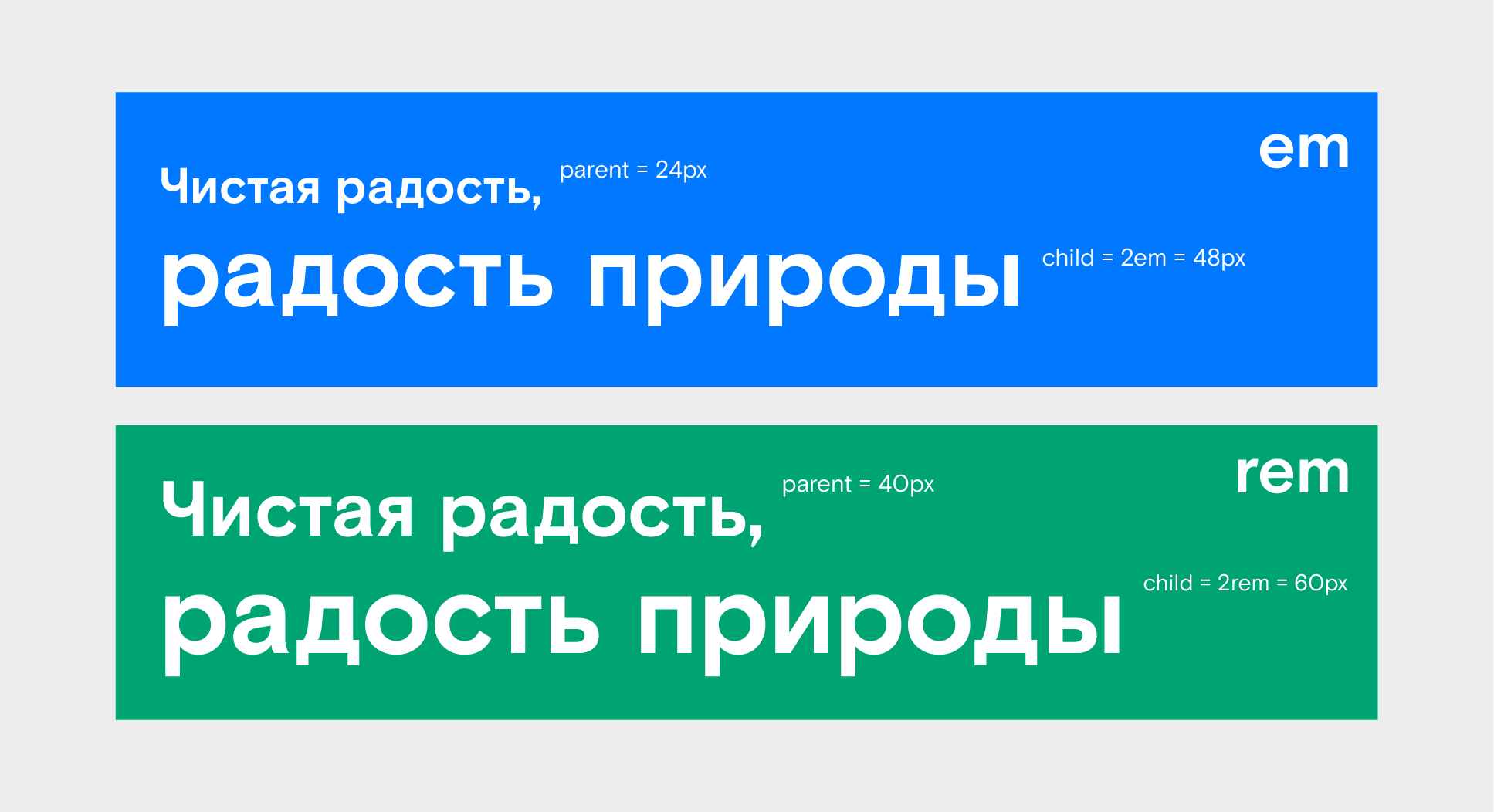
}

.child {

font-size: 2rem;

}

Здесь размер шрифта элемента .child — 60 пикселей.



Разница между размерами в em и rem

В единицах em и rem также измеряют отступы, межстрочные и межбуквенные расстояния.

Высота строки так тесно связана с размером шрифта, что для неё можно не писать слова em и rem, достаточно указать относительную величину дробным или целым числом. Например, line-height: 0.5; устанавливает интерлиньяж равным половине кегля (размера шрифта). Довольно часто эти значения равны 1.2 или 1.5. Вы уже встречали их в брифах домашней работы.

Придерживайтесь именно этого подхода. Интерлиньяж в относительных единицах — стандарт разработки.

Единицы измерения vh, vw, %, vmin и vmax тоже работают с текстовыми элементами, но применяются к ним редко.

**Особенности использования растровых изображений**

Существует два типа графических файлов, которые используют в веб-разработке: **растровые** и **векторные**.

Растровые изображения состоят из набора пикселей — цветных точек на мониторе компьютера. Описание цвета каждого пикселя называется **графической матрицей**.

Векторные изображения описаны математически в виде простейших геометрических объектов (примитивов) — точек, сплайнов, кривых Безье, кругов и окружностей, многоугольников.



Самые популярные форматы растровых изображений: .jpg, .png, .gif.

Большинство браузеров поддерживает единственный векторный формат — **.svg**

Векторные файлы описаны формулами, поэтому изображение выглядит чётким независимо от размера.

С растровыми изображениями две проблемы: 1) они плохо выглядят при простом увеличении и 2) размываются на мониторах более высокой чёткости.

1. Любой растровый файл состоит из определённого количества пикселей: увеличивая изображение, мы пропорционально увеличиваем каждый пиксель, так что при каком-то масштабе неминуемо увидим сетку из маленьких прямоугольников.
2. Разрешение современных мониторов при одинаковых размерах может различаться в 4 раза. Если два экрана имеют одинаковые размеры, но у одного из них разрешение больше, то размер пикселя этого монитора меньше, чем у менее чёткого дисплея. И если на экране с бóльшим разрешением отрисовать элементы в их реальном размере в пикселях, они станут очень мелкими.

Чтобы сохранить «физический» размер элементов, браузер увеличивает «пиксельный» размер каждого в 4 раза: вдвое по высоте и вдвое по ширине. То есть из каждого пикселя исходного изображения делает четыре. Для растровых изображений это критично: от подобного растягивания картинка неизбежно размоется.

Поэтому растровые изображения следует готовить сразу в двойном размере — так они будут хорошо смотреться на любом экране.

# Особенности использования векторных изображений

Для логотипов и иконок в современной вёрстке принято использовать векторные изображения в формате .svg. В первую очередь, это гарантирует чёткость на всех видах устройств. Помимо этого, svg-графикой можно управлять средствами CSS и JavaScript.

Обычно файл в формате .svg создают в векторных графических редакторах, таких как Sketch, Adobe Illustrator или Figma. Если открыть svg-файл в текстовом редакторе, виден его код. Примерно такой:

Скопировать кодHTML

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="98" height="39">

<g transform="matrix(1.25,0,0,-1.25,209.15146,959.834)">

<g transform="translate(-157.01488,752.9562)">

<path d="m 0,0 -1.535,0 c -2.412,0 -4.751,1.754 -4.751,6.14 0,4.568 2.193,6.432 4.422,6.432 L 0,12.572 0,0 z m 2.412,14.911 -4.166,0 c -4.094,0 -7.565,-3.107 -7.565,-9.137 0,-3.618 1.681,-6.286 4.678,-7.601 l -5.592,-10.087 c -0.183,-0.329 0,-0.585 0.292,-0.585 l 2.595,0 c 0.219,0 0.366,0.073 0.439,0.256 l 5.08,9.904 1.827,0 0,-9.904 c 0,-0.11 0.11,-0.256 0.256,-0.256 l 2.266,0 c 0.219,0 0.292,0.11 0.292,0.292 l 0,26.753 c 0,0.255 -0.146,0.365 -0.402,0.365" style="fill:#ed1e24;"/></g>

<g transform="translate(-140.86568,758.7815)"><path d="m 0,0 -2.266,0 c -0.183,0 -0.292,-0.146 -0.292,-0.256 l 0,-6.98 -4.203,0 0,6.98 C -6.761,-0.073 -6.834,0 -7.017,0 l -2.266,0 c -0.182,0 -0.292,-0.11 -0.292,-0.256 l 0,-17.762 c 0,-0.182 0.073,-0.292 0.255,-0.292 l 2.303,0 c 0.146,0 0.256,0.11 0.256,0.292 l 0,8.479 4.203,0 0,-8.479 c 0,-0.182 0.109,-0.292 0.292,-0.292 l 2.266,0 c 0.183,0 0.256,0.147 0.256,0.292 l 0,17.762 C 0.256,-0.11 0.146,0 0,0" style="fill:#231f20;"/></g>

<g transform="translate(-89.129977,741.6412)"><path d="m 0,0 -0.292,1.242 c -0.037,0.11 -0.073,0.147 -0.183,0.147 -0.183,0 -1.169,-0.402 -2.595,-0.402 -2.777,0 -4.13,2.668 -4.13,6.907 0,3.728 1.133,7.419 3.034,7.419 0.511,0 1.133,-0.146 2.119,-1.06 0.11,-0.11 0.183,-0.146 0.293,-0.146 0.11,0 0.219,0.073 0.329,0.256 l 0.987,1.608 c 0.073,0.109 0.109,0.219 0.109,0.329 0,0.146 -0.146,0.255 -0.365,0.365 -1.499,0.768 -2.522,0.914 -3.399,0.914 -3.435,0 -6.103,-4.093 -6.103,-10.489 0,-5.518 2.558,-8.698 6.213,-8.698 1.535,0 3.179,0.731 3.837,1.206 C 0,-0.292 0.037,-0.146 0,0" style="fill:#231f20;"/></g>

<g transform="translate(-105.01378,750.522)"><path d="M 0,0 4.386,8.004 C 4.459,8.113 4.422,8.26 4.276,8.26 l -2.156,0 C 1.498,8.26 1.279,8.04 1.06,7.565 L -2.229,0.731 C -2.412,0.329 -2.595,0.073 -2.778,-0.037 l 0,8.041 c 0,0.146 -0.073,0.256 -0.219,0.256 l -2.376,0 c -0.109,0 -0.219,-0.11 -0.219,-0.256 l 0,-17.798 c 0,-0.11 0.073,-0.256 0.219,-0.256 l 2.376,0 c 0.146,0 0.219,0.11 0.219,0.256 l 0,9.173 c 0.11,-0.073 0.183,-0.183 0.256,-0.329 l 4.166,-8.552 c 0.22,-0.438 0.549,-0.548 1.024,-0.548 l 2.339,0 c 0.146,0 0.183,0.146 0.109,0.256 L 0,0" style="fill:#231f20;"/></g>

<g transform="translate(-129.90198,742.7742)"><path d="m 0,0 -5.299,0 c 1.864,3.179 2.595,6.761 2.851,11.147 0.073,0.95 0.109,1.936 0.109,2.558 L 0,13.705 0,0 z M 3.947,0 3.509,0 C 2.96,0 2.814,0.219 2.814,0.658 l 0,15.093 c 0,0.183 -0.073,0.256 -0.219,0.256 l -7.309,0 c -0.183,0 -0.293,-0.109 -0.293,-0.256 l 0,-1.315 C -5.007,10.014 -5.262,4.312 -7.601,0 l -0.805,0 c -0.219,0 -0.292,-0.11 -0.292,-0.292 l 0,-5.336 c 0,-0.183 0.146,-0.256 0.256,-0.256 l 1.754,0 c 0.402,0 0.548,0.292 0.548,0.548 l 0,3.034 7.821,0 0,-3.326 c 0,-0.146 0.11,-0.256 0.256,-0.256 l 1.864,0 c 0.219,0 0.439,0.183 0.439,0.548 l 0,5.044 C 4.24,-0.11 4.13,0 3.947,0" style="fill:#231f20;"/></g>

<g transform="translate(-118.90158,751.2895)"><path d="m 0,0 c -0.84,0 -2.156,-0.146 -2.156,-0.146 0,0 0,0.548 0.073,1.133 0.329,3.106 1.096,4.678 2.376,4.678 1.169,0 1.9,-1.608 2.083,-4.423 C 2.412,0.658 2.412,0 2.412,0 L 0,0 z m 0.402,7.93 c -3.947,0 -5.701,-4.97 -5.701,-10.379 0,-5.518 2.485,-8.807 6.14,-8.807 1.79,0 3.362,0.694 4.166,1.206 0.146,0.073 0.183,0.182 0.146,0.329 l -0.292,1.242 c -0.037,0.146 -0.11,0.219 -0.293,0.219 -0.146,0 -1.206,-0.402 -2.704,-0.402 -2.814,0 -4.13,2.449 -4.13,5.665 l 0,0.865 c 0,0 1.426,-0.171 2.266,-0.171 l 5.336,0 0,1.535 c 0,5.665 -2.01,8.698 -4.934,8.698" style="fill:#231f20;"/></g></g></svg>

Графические редакторы обычно формируют не самый оптимальный svg-код. Он часто содержит множественные метаданные, повторяющиеся строчки, встроенные ненужные стили и другую дополнительную информацию. Поэтому svg-файлы перед использованием на сайте стоит оптимизировать. Для этого разработаны вспомогательные онлайн-инструменты. Один из таких — **SVGOMG**, который разработал Jake Archibald. Зайдите по ссылке <https://jakearchibald.github.io/svgomg/> , выберите файл на компьютере или вставьте код вашего svg-файла, потом отключите всё лишнее в визуальном редакторе. Вы заметите, как уменьшится вес вашего svg-файла. Скачайте его или скопируйте его код.

Файл в формате .svg можно поместить на страницу так:

1. через атрибут src тега img;
2. через свойство background-image в CSS;
3. как iframe;
4. через атрибут src тега <embed>;
5. через атрибут data тега <object>;
6. вставив код svg-файла в HTML.

Первые два способа вам знакомы. Их минус — невозможность преобразования формул изображения внешними CSS- и JS-файлами. Обычно это требуется, чтобы делать сложную анимацию, управляющую геометрией svg-изображения.

Третий и четвертый способы на практике не используются, а тег object не поддерживается ни одним современным браузером, кроме Edge от компании Microsoft.

Самый оптимальный путь работы с svg — вставлять его код прямо в код HTML-страницы. Если задачи не предполагают преобразования формул, создающих svg, можно работать с этим форматом, как с любым другим изображением.

# Оптимизация шрифтов под устройства с различными разрешениями

В различных браузерах и на разных операционных системах вид одного и того же шрифта может отличаться. Вы не раз обнаружите, что шрифты из макета в финальной вёрстке выглядят иначе. Это связано с тем, что производители программного обеспечения прибегают к разным способам сглаживания шрифтов.



Многие разработчики резонно считают, что работа со шрифтами должна производиться на стороне браузера и операционной системы, без вторжения CSS. Каким шрифт станет после обработки софтом компьютера, таким он и будет на сайте. Несмотря на это, существует ряд способов сделать шрифты более приятными глазу и сгладить разницу отображения на разных устройствах.

В первую очередь отметим, что использовать нужно только качественные шрифты, предназначенные для веба. Самый поддерживаемый формат на сегодня — это woff. Если вы не можете его нигде добыть, нужен конвертер.

Один из лучших — <https://www.fontsquirrel.com/tools/webfont-generator>. Рекомендуем использовать его.

Вот ещё несколько свойств, которые позволят вам улучшить отображение в разных браузерах:

### Сглаживание

**Сглаживание** (англ. font smoothing, «сглаживание шрифта»). **Вендорное** свойство (нестандартное, работает только с префиксами) для сглаживания **кастомных** (несистемных, нестандартных) шрифтов.

В Safari и во всех браузерах на платформе Chromium (например, Google Chrome, Яндекс Браузер или Opera):

Скопировать кодCSS

-webkit-font-smoothing: antialiased;

В Mozilla Firefox:

Скопировать кодCSS

-moz-osx-font-smoothing: grayscale;

### Подгонка размера текста

**Подгонка размера текста** (англ. text size adjust):

Скопировать кодCSS

-webkit-text-size-adjust: 100%;

-ms-text-size-adjust: 100%;

-moz-text-size-adjust: 100%;

Также вендорное свойство. Отвечает за то, чтобы при изменении масштаба страницы размер шрифта изменялся пропорционально.

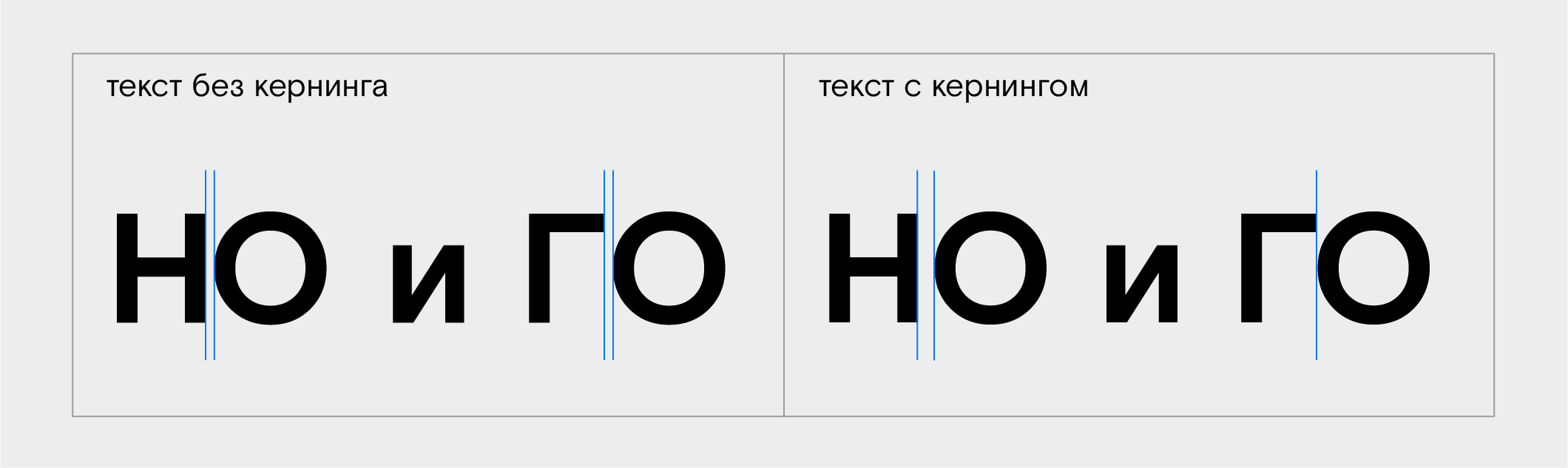
С погружением в типографику вы узнаете ещё массу необычных свойств. Мы описали те, что по умолчанию применяются в популярных библиотеках для упрощения вёрстки.

### Рендеринг

Свойство, которое задействует важные инструменты улучшения читаемости текста, в том числе **кёрнинг** и **лигатуры**.

Кёрнинг — это выборочное изменение расстояния между буквами, чтобы они лучше смотрелись рядом.

Для некоторых пар символов выигрышна меньшая дистанция. Например, буква «Г» из-за своей формы будет казаться дальше от следующей за ней «О», чем, скажем, буква «Н». Сравните:



Буква «О» будто уехала куда-то в сторону от «Г». Чтобы решить эту проблему, в шрифт добавляют таблицу кёрнинговых пар: между какими буквами расстояние нужно сократить или увеличить.

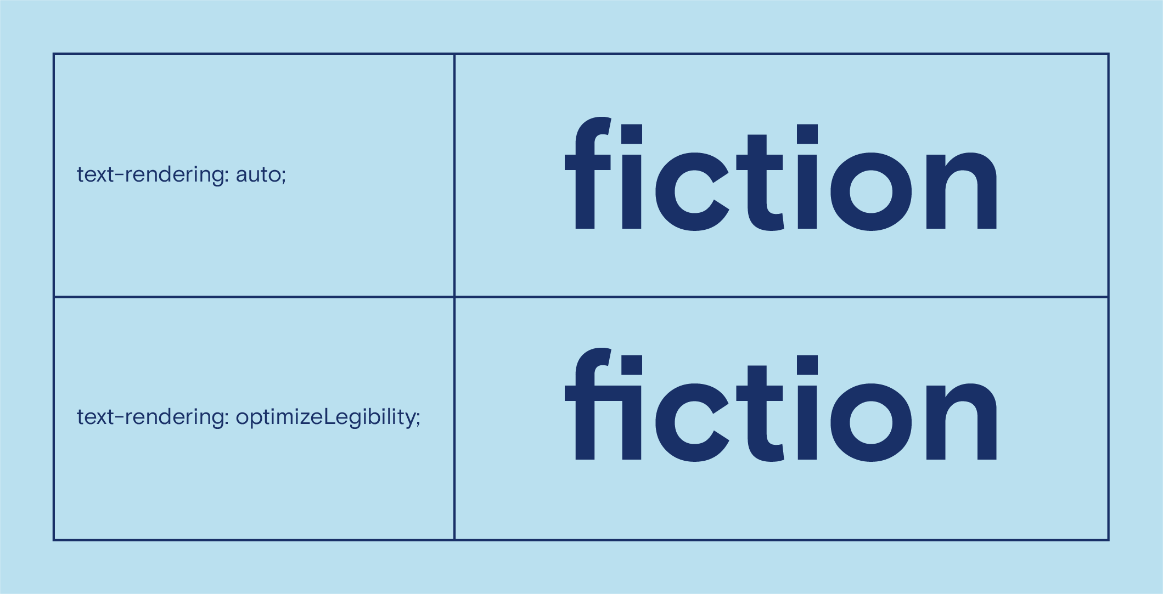
Лигатуры — знаки, образованные слиянием символов, например: њ, љ, œ, æ. Редко попадаются в русских текстах, но, к примеру, французские без них смотрятся неважно. Файлы шрифтов содержат указания, какие пары букв или символов должны образовывать лигатуры.

**Рендеринг** (англ. rendering, в значении «отрисовка») — это процесс отрисовки чего угодно на экране, в том числе и шрифта. Настройка рендеринга шрифта определяет, нужно ли применять к тексту кёрнинг и лигатуры. В CSS за это отвечает свойство text-rendering:

Скопировать кодCSS

text-rendering: optimizeLegibility;

Значение optimizeLegibility (от англ. optimize legibility, в значении «оптимизировать читаемость») говорит браузеру, что кёрнинг и лигатуры должны быть отрисованы. Учтите, что такая настройка сильно замедлит рендеринг страницы в целом.



Двигая и объединяя буквы, браузер сильнее нагружает устройство клиента-пользователя. Поэтому применяйте лигатуры и кёрнинг там, где это действительно нужно. Например, в заголовках.

Итого: если текста на сайте не много, то присваивай эти стили всем элементам.

page {

max-width: 1440px;

margin: auto;

font-family: 'Inter';

-moz-osx-font-smoothing: grayscale;

-webkit-font-smoothing: antialiased;

-webkit-text-size-adjust: 100%;

-ms-text-size-adjust: 100%;

-moz-text-size-adjust: 100%;

text-rendering: optimizeLegibility;

}

# Метатеги корректного масштабирования страницы

В контексте адаптивной вёрстки есть один очень важный метатег — viewport. С ним вы уже встречались в базовом курсе:

Скопировать кодHTML

<meta name="viewport" content="width=device-width, initial-scale=1">

Этот метатег определяет отношения между размерами веб-страницы и устройством, на котором открыт сайт. Правила этих отношений определяет значение атрибута content. У него есть 6 свойств:

* ширина (width);
* высота (height);
* масштабирование при загрузке (initial-scale);
* нижняя граница масштабирования (minimum-scale);
* верхняя граница масштабирования (maximum-scale);
* возможность пользовательского масштабирования (user-scalable).

### Ширина и высота

Свойство width определяет размер окна просмотра. Можно либо установить значение в пикселях (width=600), либо привязать к ширине экрана, на котором открыт сайт (width=device-width).

Аналогично, свойство height позволяет связать высоты экрана и макета. Для этого устанавливают значение height=device-height. Это свойство нужно довольно редко. Но когда оно нужно, обычно используется такая связка:

Скопировать кодHTML

<meta name="viewport" content="width=device-width, height=device-height, initial-scale=1">

### Настройки масштабирования

Атрибутом content можно также настроить масштаб страницы при загрузке и правила изменения размера страницы пользователем.

Сначала про масштаб. За него отвечает свойство initial-scale. Если ширина шапки вашего сайта составляет 100% от окна браузера, то при значении свойства initial-scale=1 в режиме полного экрана шапка отобразится как раз от края до края устройства. Если же установить значение меньше единицы (initial-scale=0.5), то шапка при загрузке займёт уже долю (в нашем примере — половину).

Теперь про пользовательское масштабирование. Его регулируют три свойства:

* minimum-scale;
* maximum-scale;
* user-scalable.

Первые два свойства задают границы, в которых можно масштабировать: minimum-scale определяет, насколько сильно страницу можно уменьшить, а maximum-scale — увеличить.

Последнее свойство user-scalable может принимать значения yes и no, т.е. «да» и «нет». Первый случай (yes) — это поведение страницы по умолчанию: пользователь может увеличивать и уменьшать страницу как хочет. Второй случай (no) запрещает пользователю масштабировать страницу вообще.

В общих случаях эти атрибуты не используют, но существуют частные. Например, некоторые мобильные браузеры увеличивают масштаб страницы при смене ориентации с вертикальной на горизонтальную. Такое поведение можно предотвратить, задав максимальное масштабирование.

Скопировать кодHTML

<meta name="viewport" content="initial-scale=1, maximum-scale=1">

# Медиазапросы

Устройства отличаются не только размером экрана. Так, у настольного компьютера есть клавиатура и мышь (или трекпад), которых нет у смартфона и планшета. Отсюда специфика взаимодействия с сайтом на мобильных устройствах:

* нет состояния наведения мыши;
* вместо щелчка мышью пользователь касается экрана;
* мелкий текст сложно разобрать;
* в малые элементы сложно попасть пальцем;
* дизайн в целом воспринимается иначе — отличный десктопный интерфейс может быть плохим для телефона.

Не всегда можно решить проблемы, сделав элементы тянущимися. Иногда для удобства пользования нужно совершенно поменять внешний вид элемента или перенести в другое место. Для этого мало операций с относительными единицами измерения или границами масштабирования.

Тут на помощь приходят **медиазапросы** (англ. media queries). Это директивы CSS, которые позволяют прописать элементу сразу несколько вариантов отображения. И выбирать внешний вид в зависимости от выполнения заданных условий.

Как раз медиазапросы позволяют создавать адаптивные сайты — ведь так можно прописать внешний вид элементов в зависимости от размеров окна браузера. Вот синтаксис медиазапроса:

Скопировать кодCSS

@media <условие\_1> and <условие\_2> {

<селектор> {

<свойство>: <значение>;

}

}

Размещают медиазапросы обычно в конце CSS-кода. Каждому режиму отображения — свой медиазапрос. Если проект верстается по БЭМ, медиазапросы прописываются по отдельности — для каждого компонента в конце его файла стилей, для блоков, элементов и модификаторов в их CSS-файлах. Если же стили группируются по типу, то в конце общего CSS-файла.

Условий в медиазапросах множество. Рассмотрим самые нужные.

Один из ключевых видов условий — определение типа устройства, на котором должны меняться стили. Вариантов много, от экранов монитора до устройств с поддержкой шрифтов Брайля. Регулярно используют два:

* all — все типы устройств;
* screen — экраны мониторов.

Например:

Скопировать кодCSS

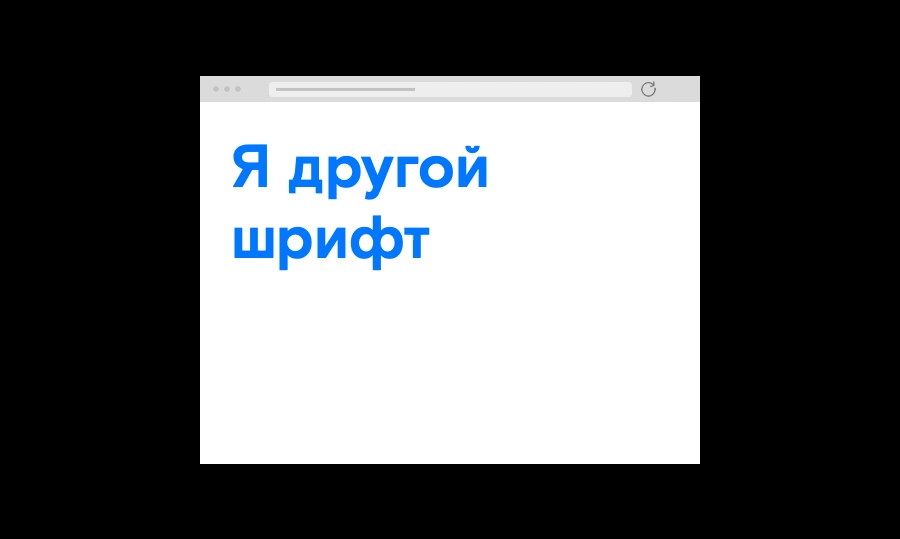
@media screen {

body {

font-family: 'Times New Roman';

}

}



Обычно также прописывают минимальные ширину и высоту. Дополнительные условия присоединяются ключевым словом and (англ. and, «и»). Стили такого запроса будут применяться, если сайт открыт с экрана, чьё разрешение находится в этой «вилке».

Скопировать кодCSS

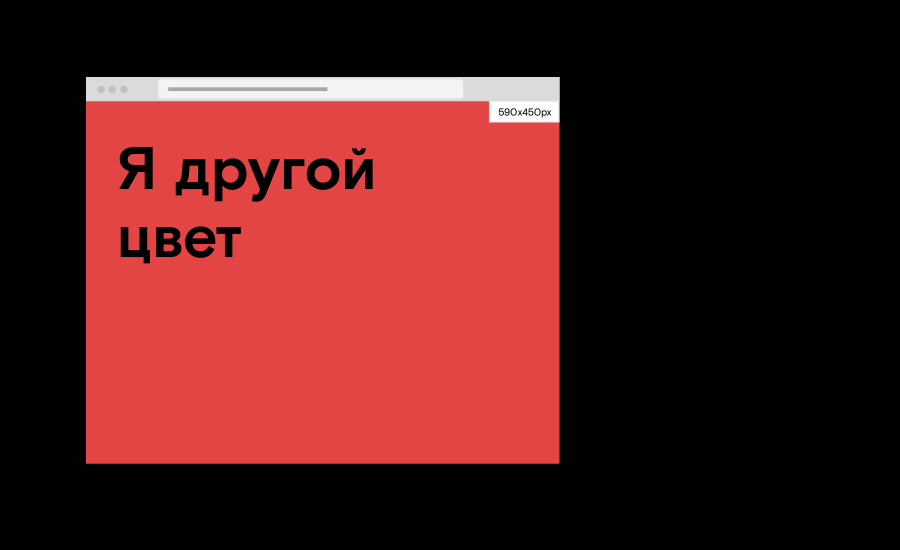
@media screen and (max-width: 720px) {

body {

background-color: red;

}

}



Фон страницы будет красным, пока размер окна браузера на мониторе не превышает 720 пикселей в ширину. Если же экран больше, цвет фона будет зависеть от стилей вне этого медиазапроса.

Минимальная ширина работает наоборот. Медиазапрос будет применён, если ширина окна браузера превысила заданный предел:

Скопировать кодCSS

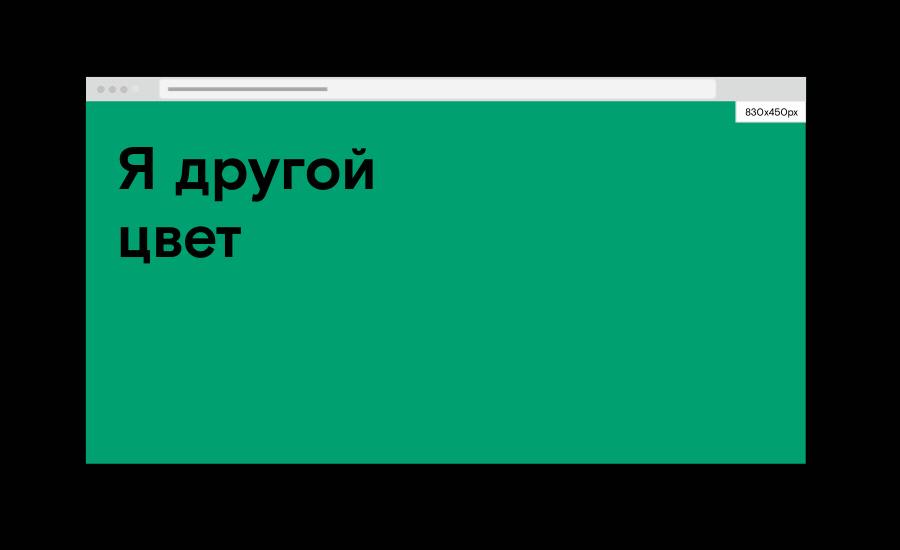
@media screen and (min-width: 720px) {

body {

background-color: green;

}

}



Когда экран шире 720 пикселей, фон страницы зелёный. Когда меньше — тот, что задан вне медиазапроса.

Важно научиться комбинировать медиазапросы под разные задачи. Кроме наших уроков, в этом вам помогут [документация](https://developer.mozilla.org/en-US/docs/Web/CSS/@media#Media_types) и [спецификация языка CSS](https://www.w3.org/TR/css3-mediaqueries/).

**Подходы к построению медиазапросов**

Обычно медиазапросов пишут много, чтобы определить поведение элементов для большинства современных устройств. Их много и появляются всё новые, поэтому нельзя написать универсальный набор медиазапросов для каждого существующего устройства.

Их сортируют по ширине и относят к одной из групп:

* маленькие смартфоны,
* широкие смартфоны,
* планшеты,
* малые настольные компьютеры и ноутбуки,
* средние настольные компьютеры и ноутбуки,
* большие мониторы,
* очень большие мониторы.

Точки перехода от одной группы к другой вам подскажут браузеры в инструментах разработчика:

Обычно базовый макет ориентирован на средний ноутбук с разрешением дисплея от 1024 до 1440 пикселей. Основные стили (до применения медиазапросов) пишут для таких устройств. Тогда по рекомендациям браузера необходимый набор медиа-запросов будет следующий:

Скопировать кодCSS

*/\* Основной код для разрешения 1024px \*/*

@media screen and (min-width: 2560px) {

*/\* стили для больших мониторов с разрешением 4K \*/*

}

@media screen and (min-width: 1440px) {

*/\* стили для больших настольных компьютеров и ноутбуков \*/*

}

@media screen and (max-width: 1024px) {

*/\* стили для планшета в альбомной ориентации \*/*

}

@media screen and (max-width: 768px) {

*/\* стили для планшета \*/*

}

@media screen and (max-width: 425px) {

*/\* стили для широкого смартфона \*/*

}

@media screen and (max-width: 375px) {

*/\* стили для среднего смартфона \*/*

}

@media screen and (max-width: 320px) {

*/\* стили для малого смартфона \*/*

}

На практике не все эти медиазапросы могут понадобиться: иногда дизайн предполагает изменение стилей только на некоторых разрешениях из списка. Часто вам придётся придумывать свои условия, исходя из специфики макета. Нужно обсудить точки перелома и поведение страницы с дизайнером.

Помните, что условия ограничения ширины можно комбинировать с другими:

Скопировать кодCSS

@media screen and (max-width: 320px) {

*/\* стили для мониторов с максимальной шириной 320 пикселей \*/*

}

@media all and (max-width: 320px) {

*/\* стили для любых типов устройств с максимальной шириной 320 пикселей \*/*

}

Есть важное правило, которое нужно соблюдать при написании медиазапросов: **пишите стили от более общего к более частному, от большего к меньшему,** это обезопасит вас от ситуаций, когда одни стили перебивают другие.

Другой интересный момент — вёрстка макетов под ландшафтную ориентацию устройств. В списке условий, применимых к медиазапросам, есть orientation со значениями portrait (англ. «портретная») и landscape (англ. «ландшафтная»). На деле поддержка браузерами этого свойства оставляет желать лучшего. Поэтому приходится комбинировать условия минимальных ширины и высоты. Вот например, условие, которое описывает горизонтальный смартфон модели iPhone 5:

Скопировать кодCSS

@media screen and (max-width: 568px) and (max-height: 320px) {

*/\* стили для iphone 5 \*/*

}

Построить сетку медиазапросов правильно — творческая задача. Нет единого рецепта, подходящего для любого дизайна. Зная основной принцип, обсуждайте частные случаи, размышляйте, экспериментируйте.

Напишем ещё одну зону медиазапросов, чтобы на горизонтальных смартфонах наша страница вела себя иначе.

Напишите медиазапрос для всех мониторов, максимальная ширина которых 812px (это ширина iPhone X в ландшафтной ориентации), а максимальная высота — 425px (рекомендация браузера по ширине больших смартфонов в портретной ориентации)

@media screen and (max-width: 812px) and (max-height: 425px) {

}

Вот код, который нужно вставить в тело медиазапроса. Делая это, объясните себе смысл каждого правила.

.content\_\_column {

width: 100%; */\* колонку растянем во всю ширину родителя \*/*

margin-bottom: 60px; */\* добавим отступ между какждой историей \*/*

display: flex; */\* пусть изображение и текст встанут рядом друг с другом \*/*

justify-content: space-between;

align-items: flex-start; */\* отменим состояние stretch, установленное по умолчанию. Изображение не будет растягиваться во всю высоту \*/*

}

.content\_\_column-image {

width: 30%; */\* теперь изображение будет размером в 30% родителя \*/*

padding-bottom: 30%;

}

.column\_\_description {

width: calc(70% - 40px); */\* учтем отступ в 40 пикселей между картинкой и текстом \*/*

}

*/\* далее стили для лучшего восприятия текстов \*/*

.content\_\_column-title {

margin-top: 0;

font-size: 1.5em;

margin-bottom: 18px;

}

.column\_\_quote {

font-size: 0.9em;

}

.column\_\_quote-cite {

font-size: 0.7em;

line-height: 1.5;

}

.share {

margin-top: 18px;

}

.share\_\_text {

font-size: 1em;

}

**Заключение**

Работа с тренажёром в этом спринте позади. Вы серьёзно потрудились! Прошли темы, составляющие основу профессии верстальщика. Теперь вас не напугать адаптивным дизайном, ведь вы знаете, как сделать гибкими сетку и блоки сайта. Даже полностью поменять внешний вид элемента под определённый размер экрана вам под силу.

Любой навык приходит с практикой. Вы закрепите полученные знания в новой проектной работе. Будет нелегко, ведь это ваш первый адаптивный сайт. К тому же придётся работать с макетом в «Фигме». Всё как в реальной работе.

Сначала делать адаптивные сайты сложно: на одном из разрешений всё время что-то ломается. Это случается даже с самыми лучшими разработчиками. Не паникуйте! Нет единых рецептов создания «идеального респонсив-макета». Рецепт каждый раз новый, как и макет, полученный от дизайнера. Ваша сила — в знании инструментов. Если экспериментировать и осознанно их применять, вы всегда добьётесь результатов.

У вас впереди:

1. несколько советов, что делать, если сверстать адаптивно никак не выходит,
2. очередная порция информации про Git
3. и новый проект.

Эта проектная работа не простая. Выполнив её, вы не просто положите проект в портфолио. После неё вы сможете гордо назвать себя верстальщиком.

# Что делать, если не получается сделать страницу адаптивной

Для многих студентов этот спринт станет первым большим «aha-моментом» (моментом осознания чего-то важного). Раньше вы верстали макет только для одной ширины экрана. В этот раз разрешений экрана будет много, вам придётся думать «в динамике». Представлять блок и решать, какие CSS-свойства к нему применить, чтобы он корректно работал на разных экранах.

Что делать, если блок нужно сделать адаптивным, но вы не помните, как:

1. Не напрягайтесь, очень многие студенты проходят через эту фазу. Думайте об этом как о «прокачке» абстрактного мышления и мыслительных способностей.
2. Откройте в соседней вкладке конспекты:
   * по «адаптиву»: [разработка интерфейса для разных устройств](https://code.s3.yandex.net/web-developer/cheat-sheet/cheat-sheet-developing-an-Interface-for-different-devices.pdf);
   * по позиционированию: [позиционирование элементов](https://code.s3.yandex.net/web-developer/cheat-sheet/cheat-sheet-positioning-elements.pdf);
   * о флексах: [Flexbox-вёрстка](https://code.s3.yandex.net/web-developer/cheat-sheet/cheat-sheet-flexbox-layout.pdf" \t "_blank).
3. Пройдитесь по ним заново. Попробуйте использовать конспекты от css-tricks:
   * [о свойстве display](https://css-tricks.com/almanac/properties/d/display/);
   * [позиционировании](https://css-tricks.com/almanac/properties/p/position/);
   * [флексбоксах](https://css-tricks.com/snippets/css/a-guide-to-flexbox/);
   * [гридах](https://css-tricks.com/snippets/css/complete-guide-grid/);
   * [медиа-запросах](https://css-tricks.com/snippets/css/media-queries-for-standard-devices/).
4. Откройте [онлайновый редактор кода CodePen](https://codepen.io/) и воспроизведите один из блоков, над которым сейчас работаете, в упрощённом виде: только сам элемент и родитель. Если у родителя несколько элементов, добавьте их. Не копируйте код из редактора, пишите самостоятельно с нуля. Так вы сможете переосмыслить код и избежать ошибок и опечаток.
5. Старайтесь понимать, как работают все свойства, которые вы используете в стилях. Если что-то не понимаете — разберитесь. В крайнем случае попробуйте избавиться от этого свойства.
6. Порастягивайте и посжимайте окно браузера, посмотрите, как ведёт себя вёрстка.
7. Определите, какой из элементов вываливается и «не хочет» вставать на место. Обратите внимание на CSS-свойства, которые влияют на поведение элемента. В первую очередь смотрите на display и position элемента и его родителя.

Скорее всего, к этому моменту вам удастся разобраться в чём проблема. Если нет, и вы сидите над проблемой больше 3-х часов, напишите в общий чат. Другие студенты или наставник вам помогут. Как просить о помощи, чтобы вам помогли, мы обсуждали в первом спринте (<https://praktikum.yandex.ru/trainer/web/lesson/471d902d-0c44-4180-824d-a0e6302f16b7>).