

Современный учебник JavaScript

© Илья Кантор

Сборка от 27 апреля 2014 для печати

Внимание, эта сборка может быть устаревшей и не соответствовать текущему тексту.

Актуальный онлайн-учебник, с интерактивными примерами, доступен по адресу <http://learn.javascript.ru>.

Вопросы по JavaScript можно задавать в комментариях на сайте или на форуме javascript.ru/forum.

Вопросы по сборке, предложения по её улучшению – можно писать мне, по адресу iliakan@javascript.ru .

Глава: CSS для JavaScript-разработчика

В файле находится только одна глава учебника. Это сделано в целях уменьшения размера файла, для удобного чтения с устройств.

Содержание

О чём пойдёт речь

- Чек-лист

- Почитать

Единицы измерения "px", "em", "%" и другие

- Расшифровка единиц измерения

- px и em/%

- % и em

- Неактуальная проблема: масштабирование

Свойства "font-size" и "line-height"

- font-size и line-height

- Синтаксис font: size/height family

- Итого

Свойство white-space

- pre / nowrap

- pre-wrap/pre-line

Свойство "outline"

Свойство "box-sizing"

- Значения box-sizing

- Пример: подстроить ширину к родителю

- Попытка width:100%

- Решение: дополнительный элемент

- Решение: box-sizing

Свойство "margin"

- Объединение отступов

- Отрицательные margin-top/left
- Пример: вынос заголовка
- Пример: вынос отчерка
- Отрицательные margin-right/bottom
- Итого

Свойство "display"

- Значение none
- Значение block
- Значение inline
- Значение inline-block
- Значения table-*
- Вертикальное центрирование с table-cell
- Значения list-item и run-in

Лишнее место под IMG

- Демонстрация проблемы
- Решение: сделать элемент блочным
- Решение: задать vertical-align
- Итого

Свойство "float"

- Как работает float
 - Текст с картинками
 - Блок с float
- Очистка под float
 - Шаг 1. Добавляем информацию
 - Шаг 2. Свойство clear
- Заполнение блока-родителя
 - Поставить родителю float
 - Добавить в родителя элемент с clear
 - Универсальный класс clearfix
 - overflow:auto/hidden
- Еще применения float
 - Галерея
 - Вёрстка в несколько колонок
 - float:left + float:right
 - float + margin

Свойство "overflow"

- visible
- hidden
- auto
- scroll
- overflow-x, overflow-y
- Итого

Свойство "position"

- position: static
- position: relative
 - Координаты
- position: absolute
- position: absolute в позиционированном родителе
- position: fixed

Итого

Почитать

Особенности свойства "height" в %

Пример

Попытка height:100% + float:left

Решение: height:100% + position:absolute

Итого

Знаете ли вы селекторы?

Основные виды селекторов

Отношения

Фильтр по месту среди соседей

Фильтр по месту среди соседей с тем же тегом

Селекторы атрибутов

Другие псевдофильтры

Псевдоэлементы ::before, ::after

Практика

CSS без IE6(7)

CSS-спрайты

Шаг 1. Использовать background

Шаг 2. Объединить изображения

Шаг 3. Показать часть спрайта в «окошке»

Шаг 4. Сдвинуть спрайт

Отступы

Итого

Центрирование горизонтальное и вертикальное

Горизонтальное

text-align

margin: auto

Вертикальное

position:absolute + margin

Одна строка: line-height

Свойство vertical-align

Таблица с vertical-align

Центрирование в строке с vertical-align

Центрирование с vertical-align без таблиц

Итого

Правила форматирования CSS

Каждое свойство — на отдельной строке

Каждый селектор — на отдельной строке

Свойства, сильнее влияющие на документ, идут первыми

Организация CSS-файлов проекта

Решения задач

О чём пойдёт речь

Неужели мы сейчас будем учить CSS? Ничего подобного. Предполагается, что вы уже знаете CSS.

Если нет, то обязательно подучите перед тем, как читать этот раздел. Мы будем останавливаться на особенностях CSS, понемногу, но очень конкретно.

Особенность квалификации JavaScript-разработчика заключается в том, что он не обязан выбирать цвета, рисовать иконки, «делать красиво». Он также не обязан верстать макет в HTML, разве что если является по совместительству специалистом-верстальщиком.

Вот что он должен уметь абсолютно точно — так это и **разработать такую структуру HTML/CSS для элементов управления, которая не сломается, и с которой ему же потом удобно будет взаимодействовать**.

Это требует **отличного знания CSS в области позиционирования** элементов, включая тонкости работы `display`, `margin`, `border`, `outline`, `position`, `float`, `border-box` и остальных свойств, а также подходы к построению структуры компонент (CSS layouts).

Многое можно сделать при помощи JavaScript. И зачастую, не зная CSS, так и делают. Но мы на это ловиться не будем.

Если что-то можно сделать через CSS — лучше делать это через CSS.

Причина проста — как бы ни был сложен CSS, JavaScript ещё сложнее. С этим можно поспорить, особенно если достать из нафталина IE 6,7 (IE5.5 не заваялся?) и показать, что CSS там ну совсем никакой. Да и в IE8 тоже есть забавные баги.

Как правило, даже если CSS на вид сложнее — поддерживать и развивать его проще, чем JS. Поэтому овчинка стоит выделки.

Кроме того, есть ещё одно наблюдение.

Знание JavaScript не может заменить знание CSS.

Жить становится приятнее и проще, если есть хорошее знание и CSS и JavaScript.

Чек-лист

Ниже находится «чек-лист». Если хоть одно свойство незнакомо — это стоп-сигнал для дальнейшего чтения этого раздела.

- ➡ Блочная модель, включая:
 - ➡ `margin`, `padding`, `border`, `overflow`
 - ➡ а также `height/width` и `min-height/min-width`.
- ➡ Текст:
 - ➡ `font`
 - ➡ `line-height`.
- ➡ Различные курсоры `cursor`.
- ➡ Позиционирование:
 - ➡ `position`, `float`, `clear`, `display`, `visibility`
 - ➡ Центрирование при помощи CSS
 - ➡ Перекрытие `z-index` и прозрачность `opacity`

- Селекторы:
 - Приоритет селекторов
 - Селекторы `#id`, `.class`, `a > b`
- Сброс браузерных стилей, [reset.css](#) [1]

Почитать

Книжек много, но хороших — как всегда, мало.

С уверенностью могу рекомендовать следующие:

- [CSS. Каскадные таблицы стилей. Подробное руководство. Эрик Мейер.](#) [2]
- [Большая книга CSS, Дэвид Макфарланд](#) [3]
- [CSS ручной работы, Дэн Седерхольм](#) [4]

Дальнейшие статьи раздела не являются *учебником* CSS, поэтому пожалуйста, изучите эту технологию до них. Это *очерки для лучшей систематизации и дополнения* уже существующих знаний.

Единицы измерения "px", "em", "%" и другие

Есть три основных единицы измерения в CSS: px, em и %. Остальные — производные от них.

Расшифровка единиц измерения

Основные единицы измерения:

- **1px — один пиксель.**
- **1em — текущий размер шрифта.** В большинстве не-мобильных браузеров шрифт по умолчанию имеет размер 16px. Можно брать любые пропорции от текущего шрифта: 2em, 0.5em и т.п.
- **1% — процент от другого свойства, какого — зависит от того, размер чего ставим.** Например, при установке свойства `margin-left: 1%`, процент берётся от ширины родительского блока, т.е. `margin-left` будет размером в 1% ширины родителя. Лишь при установке `font-size` процент берётся от текущего размера шрифта.

Это отличается от em, который привязан к шрифту всегда.

Производные единицы измерения: mm, cm, pt и pc:

- **Производные от пикселя:**
 - 1mm(мм) = 3.8px
 - 1cm(см) = 38px
 - 1pt(пункт) = 4/3 px
 - 1pc(пика) = 16px

Такие единицы как сантиметр "cm" и миллиметр "mm" нам всем знакомы, а *пункт* "pt" и *пика* "pc" пришли из типографии.

Здесь мы не будем углубляться в детали, но один вопрос у вас, если вы задумываетесь в то, что читаете, наверняка возник: «Почему все эти единицы зависят от пикселя?»

В частности, почему в одном сантиметре cm содержится ровно 38 пикселей? Ведь [пиксель](#) [5] — это точка на мониторе, а

мониторы бывают разные, и размеры этого самого пикселя могут отличаться.

..Всё дело в том, что здесь имеются в виду пиксели на «стандартизованном экране», характеристики которого описаны в [спецификации \[6\]](#), а вовсе не на мониторе посетителя. Поэтому и верны указанные выше равенства.

→ **Производные от шрифта:** `1ex` — высота буквы "x" в текущем шрифте. Эта единица измерения определена для любых шрифтов, включая те, в которых буквы "x" нет.

Производные единицы редко можно увидеть в проектах, так как использовать их, вообще говоря, нет необходимости. Достаточно базовых размеров `px`, `em`, `%`.



`vw`, `vh` и `vmin`

Webkit также поддерживает единицы из черновика стандарта [CSS Values and Units 3 \[7\]](#):

- `vw` — 1% ширины окна
- `vh` — 1% высоты окна
- `vmin` — наименьшее из (`vw`, `vh`)

`px` и `em/%`

Теперь поговорим об основной разнице между `px` (и производными от него) и значениями `em/%`.

Размеры в пикселях — абсолютные. Они не зависят от того, в каком месте находится элемент:

```
1 <div style="font-size:24px">
2   24px
3 <div style="font-size:24px">24px</div>
4 </div>
```

24px

24px

24 пикселей — и в Африке 24 пикселей, поэтому две строчки выше одинаковы.

Размеры в `em/%` — относительные. Они определяются по текущему контексту:

```
1 <div style="font-size:1.5em">  
2 1.5em  
3 <div style="font-size:1.5em">1.5em</div>  
4 </div>
```

1.5em
1.5em

В примере выше первая строчка при настройках браузера по умолчанию равна $16 * 1.5 = 24\text{px}$. А вторая в 1.5 раза больше текущего, уже увеличенного, размера шрифта, т.е. $24 * 1.5 = 36\text{px}$.

Аналогичная ситуация с процентами:

```
1 <div style="font-size:150%">  
2 150%  
3 <div style="font-size:150%">150%</div>  
4 </div>
```

150%
150%

% и em

Разница между % и em появляется при установке других свойств, кроме размера шрифта.

В этом случае смысл процента % зависит от свойства, например, для width/height это ширина/высота родителя. А em всегда привязан к шрифту.

Неактуальная проблема: масштабирование

Когда-то давно стандартное масштабирование браузеров некорректно работало с абсолютными размерами px.

Иначе говоря, если посетитель нажимал Ctrl+ или ещё как-то увеличивал масштаб страницы, то размеры em/% увеличивались корректно, а пиксели px оставались «как есть». Было некрасиво.

До сих пор в интернет существует множество статей на тему этих проблем и как их обходить. Но времена изменились, и встроенное масштабирование сейчас работает нормально. **Поэтому эта проблема не актуальна.**

С другой стороны, для масштабирования средствами JavaScript/CSS по-прежнему удобны относительные единицы: em и %.

Свойства "font-size" и "line-height"

Здесь мы рассмотрим, из каких компонентов состоит размер строки и шрифта, и как их удобно задавать.

font-size и line-height

- font-size — *размер шрифта*, в частности, определяющий высоту букв.
- line-height — *высота строки*.

Как соотносятся размер шрифта и высота строки?

Посмотрим это на примере:

```
1 <style>
2   body {
3     font-size: 40px;
4     line-height: 1em;
5   }
6 </style>
7
8 <div style="border: 1px dotted red">TTTppp</div>
```

Хвосты букв будут из такой строки вылезать, это видно в примере выше. Поэтому стандартные значения в большинстве браузеров таковы:

- font-size: 16px
- line-height: 1.25

Обратите внимание: **line-height** указан без единицы измерения. Это очень важно и обозначает *множитель*.

При указании line-height множителем — оно увеличивается относительно своего текущего значения. Для стандартного стиля браузера (line-height: 1.25) значению line-height: 2 будет соответствовать line-height: 2.5em, так как $1.25 * 2 = 2.5$.

Но, тем не менее, между тем, как они применяются, будет одна существенная разница.

- Значение, заданное множителем, наследуется и применяется в каждом элементе относительно его размера шрифта.
- Значение, заданное в единицах измерения, запоминается и наследуется «как есть»

Сравним эти способы:

Множитель	Конкретное значение


```

01 <ul>
02 <li style="line-height:1.5;font-size:10px">
03   шрифт 10px<br>строка 15px
04 <ul>
05   <li style="font-size:20px">
06     шрифт 20px<br>строка 30px
07   </li>
08 </ul>
09 </li>
10 </ul>

```

- шрифт 10px
строка 15px
 - шрифт 20px
строка 30px

Высота строки внутреннего списка берётся относительно текущего шрифта $20\text{px} \times 1.5 = 30\text{px}$.

```

01 <ul>
02 <li style="line-height:1.5em; font-size:10px">
03   шрифт 10px<br>строка 15px
04 <ul>
05   <li style="font-size:20px">
06     шрифт 20px<br>строка 15px
07   </li>
08 </ul>
09 </li>
10 </ul>

```

- шрифт 10px
строка 15px
 - шрифт 20px
строка 15px

Высота строки внутреннего списка просто наследуется от внешнего, где она равна 15px.

Как видно, слева высота строки привязана к размеру шрифта. Шрифт больше — высота больше. А справа она фиксирована.

В обычных ситуациях рекомендуется использовать именно множитель, за исключением особых случаев, когда вы действительно знаете что делаете.

Синтаксис `font: size/height family`

Есть синтаксис для одновременной установки `font-size` и `line-height`. Выглядит он так:

```
font: 20px/1.5 Arial,sans-serif;
```

При этом нужно обязательно указать сам шрифт, например `Arial,sans-serif`. Укороченный вариант `font: 20px/1.5` работать не будет.

Дополнительно можно задать и свойства `font-style`, `font-weight`:

```
font: italic bold 20px/1.5 Arial,sans-serif;
```

Итого

`line-height`

Размер строки, обычно он больше размера шрифта. При установке множителем рассчитывается каждый раз относительно текущего шрифта, при установке в единицах измерения — фиксируется.

font-size

Размер шрифта. Если сделать блок такой же высоты, как шрифт, то хвосты букв будут вылезать из-под него.

font: 125%/1.5 FontFamily

Даёт возможность одновременно задать размер, высоту строки и, собственно, сам шрифт.

Свойство white-space

Свойство white-space позволяет сохранять пробелы и переносы строк.

У него есть два известных значения:

- `white-space: normal` — обычное поведение
- `white-space: pre` — текст ведёт себя, как будто оформлен в тег `pre`.

Но браузеры поддерживают ещё 3, два из которых — только в IE8+ и очень полезны.

pre / nowrap

Встречаем первую «сладкую парочку» — `pre` и `nowrap`.

Оба этих значения меняют стандартное поведение HTML при работе с текстом:

pre:

- **сохраняет пробелы**
- **переносит текст при явном разрыве строки.**

nowrap

- **не сохраняет пробелы**
- **игнорирует явные разрывы строки (не переносит текст).**

Оба значения считают, что если текст слишком длинный и вылезает из контейнера — что ж, так тому и быть.

Для примера, рассмотрим следующий фрагмент кода:

```
if (hours > 18) {  
    // Пойти поиграть в теннис  
}
```

- **white-space: pre:**

```
if (hours > 18) {  
    // Пойти поиграть в теннис  
}
```

Здесь пробелы и переводы строк сохранены. В HTML этому значению white-space соответствует тег `PRE`.

- **white-space: nowrap:**

```
if (hours > 18) { // Пойти поиграть в теннис }
```

Здесь переводы строки проигнорированы, а подряд идущие пробелы, если присмотреться — сжаты в один (например, перед комментарием //).

Оба этих значения поддерживаются кросс-браузерно.

Их основной недостаток — текст вылезает из контейнера.

Например, мы хотим разрешить посетителям публиковать код на сайте, с сохранением разметки. Но тег `PRE` и описанные выше значения `white-space` для этого не подойдут!

Злой Василий Пупкин может написать такой текст, который вылезет из контейнера и поломает вёрстку страницы. В лучшем случае — появится горизонтальная прокрутка.

Можно, конечно, поставить на контейнере `overflow-x: hidden` и забыть. Но есть и другие значения `white-space`, специально для таких случаев.

pre-wrap/pre-line

Эти значения не поддерживаются в IE до версии 8.

pre-wrap

То же самое, что `pre`, но переводит строку, если текст вылезает из контейнера.

pre-line

То же самое, что `pre`, но переводит строку, если текст вылезает из контейнера и не сохраняет пробелы.

Оба поведения отлично прослеживаются на примерах:

➡ `white-space: pre-wrap:`

```
if (hours > 18) {  
    // Пойти поиграть в  
    теннис  
}
```

Отличный выбор для безопасной вставки кода посетителями.

➡ `white-space: pre-line:`

```
if (hours > 18) {  
    // Пойти поиграть в теннис  
}
```

Сохранены переводы строк, ничего не вылезает, но пробелы интерпретированы в режиме обычного HTML.

Никто не мешает использовать эти значения на сайте, а для старых IE игнорировать проблему (как-то все равно будет смотреться), или задавать контейнеру `overflow-x: hidden`.

Свойство "outline"

Свойство `outline` задаёт дополнительную рамку вокруг элемента, за пределами его CSS-блока. Поддерживается во всех браузерах, IE8+.

Например:

```
<div style="border:3px solid blue;outline: 3px solid red">
```

Элемент

```
</div>
```



- Рамку `outline` можно задать только со всех сторон: свойств `outline-top`, `outline-left` не существует.
- В отличие от `border`, рамка `outline` не участвует в блочной модели **CSS**. Поэтому его используют, когда хотят добавить рамку без изменения других CSS-параметров.

Так как `outline` находится за границами элемента — **outline-рамки соседей могут перекрывать друг друга**:

```
1 <div style="outline: 3px solid green">
```

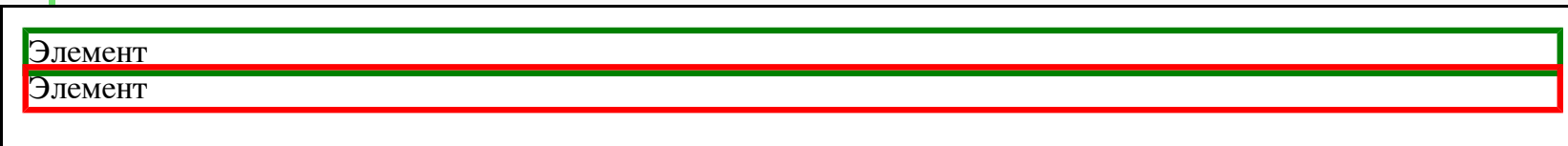
```
2   Элемент
```

```
3 </div>
```

```
4 <div style="outline: 3px solid red">
```

```
5   Элемент
```

```
6 </div>
```



В примере выше верхняя рамка нижнего элемента находится на территории верхнего и наоборот.

Все браузеры, кроме IE<10, также поддерживают свойство `outline-offset`, задающее отступ `outline` от внешней границы элемента:

```
<div style="border:3px solid blue; outline: 3px solid red; outline-offset:5px">
```

Везде, кроме IE<10, между рамками будет расстояние 5px

```
</div>
```



Ещё раз заметим, что основная особенность `outline` — в том, что при наличии `outline-offset` или без него — он не занимает места в блоке.

Свойство "box-sizing"

Свойство `box-sizing` может принимать одно из двух значений — `border-box` или `content-box`. В зависимости от выбранного значения браузер по-разному трактует значение свойств `width/height`.

Значения box-sizing

content-box

Это значение по умолчанию. В этом случае свойства width/height обозначают то, что находится *внутри padding*.

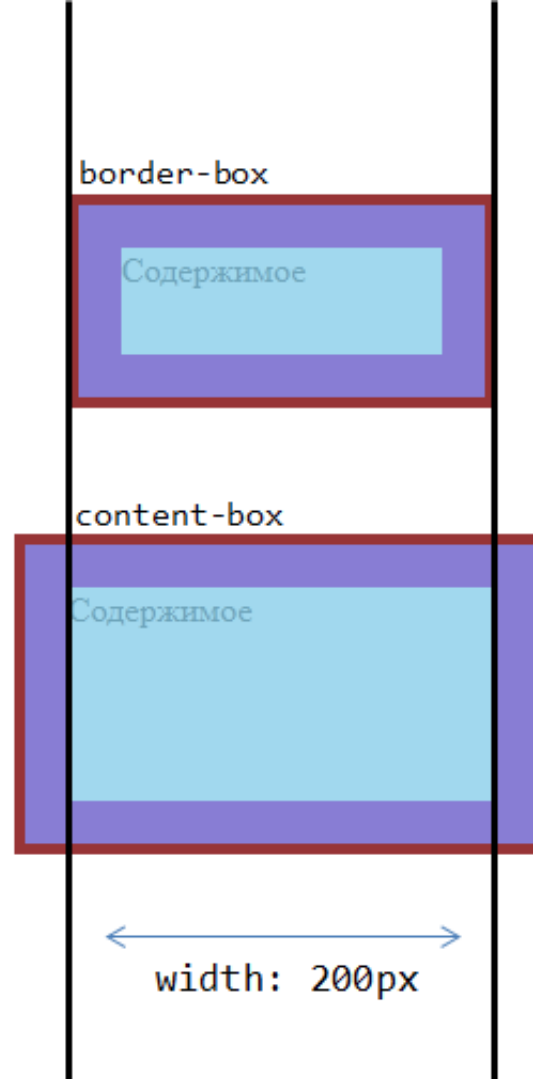
border-box

Значения width/height задают высоту/ширину *всего элемента*.

Для большей наглядности посмотрим на картинку этого div в зависимости от box-sizing:

```
1 div {  
2   width: 200px;  
3   height: 100px;  
4   box-sizing: border-box (вверху) | content-box (внизу);  
5  
6   padding: 20px;  
7   border: 5px solid brown;  
8 }
```

`border: 5px`
`padding: 20px`



В верхнем случае браузер нарисовал весь элемент размером в `width` x `height`, в нижнем — интерпретировал `width/height` как размеры внутренней области.

Исторически сложилось так, что по умолчанию принят `content-box`, а `border-box` некоторые браузеры используют если не указан `DOCTYPE`, в режиме совместимости.

Но есть как минимум один случай, когда явное указание `border-box` может быть полезно: растягивание элемента до ширины родителя.

Пример: подстроить ширину к родителю

Задача: подогнать элемент по ширине внешнего элемента, чтобы он заполнял всё его пространство. Без привязки к конкретному размеру элемента в пикселях.

Например, мы хотим, чтобы элементы формы ниже были одинакового размера:

```

01 <style>
02   form {
03     width: 200px;
04     border: 2px solid green;
05   }
06
07   form input, form select {
08     display: block;
09     padding-left: 5px; /* padding для красоты */
10   }
11 </style>
12
13 <form>
14   <input>
15   <input>
16   <select><option>опции</option></select>
17 </form>

```

Как сделать, чтобы элементы растянулись чётко по ширине FORM? Попробуйте добиться этого самостоятельно, перед тем как читать дальше.

Попытка `width:100%`

Первое, что приходит в голову — поставить всем INPUT 'ам ширину `width: 100%`.

Попробуем:

```

01 <style>
02   form {
03     width: 200px;
04     border: 2px solid green;
05   }
06
07   form input, form select {
08     display: block;
09     padding-left: 5px;
10     width: 100%;
11   }
12 </style>
13
14 <form>
15   <input>
16   <input>
17   <select><option>опции</option></select>
18 </form>

```

Как видно, не получается. **Элементы вылезают за пределы родителя.**

Причина — ширина элемента 100% по умолчанию относится к внутренней области, не включающей padding и border. То есть, внутренняя область растягивается до 100% родителя, и к ней снаружи прибавляются padding/border, которые и вылезают.

Есть два решения этой проблемы.

Решение: дополнительный элемент

Можно убрать padding/border у элементов INPUT/SELECT и завернуть каждый из них в дополнительный DIV, который будет обеспечивать дизайн:


```

01 <style>
02   form {
03     width: 200px;
04     border: 2px solid green;
05   }
06
07   /* убрать padding/border */
08   form input, form select {
09     padding: 0;
10     border: 0;
11     width: 100%;
12   }
13
14   /* внешний div даст дизайн */
15   form div {
16     padding-left: 5px;
17     border: 1px solid black;
18   }
19
20 </style>
21
22 <form>
23   <div><input></div>
24   <div><input></div>
25   <div><select><option>опции</option></select></div>
26 </form>

```

В принципе, это работает. Но нужны дополнительные элементы. А если мы делаем дерево или большую редактируемую таблицу, да и вообще — любой интерфейс, где элементов и так много, то лишние нам точно не нужны.

Кроме того, такое решение заставляет пожертвовать встроенным в браузер дизайном элементов INPUT/SELECT.

Решение: box-sizing

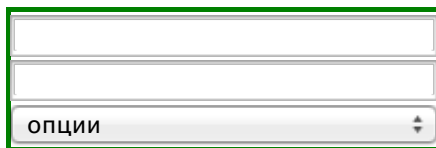
Существует другой способ, гораздо более естественный, чем предыдущий.

При помощи box-sizing: border-box мы можем сказать браузеру, что ширина, которую мы ставим, относится к элементу полностью, включая border и padding:

```

01 <style>
02   form {
03     width: 200px;
04     border: 2px solid green;
05   }
06
07   form input, form select {
08     display: block;
09     padding-left: 5px;
10     -moz-box-sizing: border-box; /* в Firefox нужен префикс */
11     box-sizing: border-box;
12     width: 100%;
13   }
14 </style>
15
16 <form>
17   <input>
18   <input>
19   <select><option>опции</option></select>
20 </form>

```



Мы сохранили «родную» рамку вокруг INPUT/SELECT и не добавили лишних элементов. Всё замечательно.

Свойство `box-sizing` поддерживается в IE начиная с версии 8.

Свойство "margin"

Свойство `margin` задаёт отступы вокруг элемента. У него есть несколько особенностей, которые мы здесь рассмотрим.

Объединение отступов

Вертикальные отступы поглощают друг друга, горизонтальные — нет.

Например, вот документ с вертикальными и горизонтальными отступами:

```

1 <body style="background: #aef">
2   <p style="margin:15px; background:white">
3
4     <span style="margin:15px; background:orange">Параграф 1</span>
5     <span style="margin:15px; background:orange">Текст</span>
6
7   </p>
8   <p style="margin:10px; background:white">Параграф 2</p>
9 </body>

```

Параграф 1 Текст

Параграф 2

Каждый параграф P задаёт отступ: верхний — 15px, нижний 10px. Но больший вертикальный отступ поглощает меньший, так что по вертикали расстояние между параграфами — 15px.

Горизонтальный отступ между элементами SPAN (Параграф 1 и Текст) шире, так как правый и левый просто складываются. Получается $15 + 15 = 30\text{px}$.

Отрицательные margin-top/left

Отрицательные значения margin-top/margin-left смещают элемент со своего обычного места.

При этом другие элементы занимают освободившееся пространство, в отличие от `position: relative`, когда место, где был элемент, остается как бы «занятым».

То есть, элемент продолжает полноценно участвовать в потоке.

Пример: вынос заголовка

Например, есть документ с информационными блоками:

```

01 <style>
02   div {
03     border: 1px solid blue;
04     margin: 2em;
05     font: .8em/1.25 Arial, sans-serif;
06   }
07   h2 {
08     background: #aef;
09     margin: 0 0 0.8em 0;
10   }
11 </style>
12
13 <div>
14   <h2>Общие положения</h2>
15
16   <p>Настоящие Правила дорожного движения устанавливают единый порядок дорожного движения на всей

```

```

17   территории Российской Федерации. Другие нормативные акты, касающиеся дорожного движения, должны
18   основываться на требованиях Правил и не противоречить им.</p>
19 </div>
20 <div>
21   <h2>Общие обязанности водителей</h2>
22   <p>Водитель механического транспортного средства обязан иметь при себе и по требованию сотрудников
23   милиции передавать им для проверки:</p>
24   <ul>
25     <li>водительское удостоверение на право управления транспортным средством соответствующей категории;</li>
26     <li>...и так далее...</li>
27   </ul>
28 </div>

```

Общие положения

Настоящие Правила дорожного движения устанавливают единый порядок дорожного движения на всей территории Российской Федерации. Другие нормативные акты, касающиеся дорожного движения, должны основываться на требованиях Правил и не противоречить им.

Общие обязанности водителей

Водитель механического транспортного средства обязан иметь при себе и по требованию сотрудников милиции передавать им для проверки:

- водительское удостоверение на право управления транспортным средством соответствующей категории;
- ...и так далее...

Использование отрицательного `margin-top` позволяет вынести заголовки над блоком.

```

1 h2 {
2   /* вверх на высоту строки (1.25em) и ещё немного */
3   margin-top: -1.3em;
4 }

```

Результат:

Общие положения

Настоящие Правила дорожного движения устанавливают единый порядок дорожного движения на всей территории Российской Федерации. Другие нормативные акты, касающиеся дорожного движения, должны основываться на требованиях Правил и не противоречить им.

Общие обязанности водителей

Водитель механического транспортного средства обязан иметь при себе и по требованию сотрудников милиции передавать им для проверки:

- водительское удостоверение на право управления транспортным средством соответствующей категории;
- ...и так далее...

А вот, что бы было при использовании position:

```
1 h2 {  
2   position: relative;  
3   top: -1.3em;  
4 }
```

Результат:

Общие положения

Настоящие Правила дорожного движения устанавливают единый порядок дорожного движения на всей территории Российской Федерации. Другие нормативные акты, касающиеся дорожного движения, должны основываться на требованиях Правил и не противоречить им.

Общие обязанности водителей

Водитель механического транспортного средства обязан иметь при себе и по требованию сотрудников милиции передавать им для проверки:

- водительское удостоверение на право управления транспортным средством соответствующей категории;
- ...и так далее...

При использовании `position`, в отличие от `margin`, на месте заголовков, внутри блоков, осталось пустое пространство.

Пример: вынос отчерка

Организуем информацию чуть по-другому. Пусть после каждого заголовка будет отчерк:

```
1 <div>
2   <h2>Заголовок</h2>
3   <hr>
4
5   <p>Текст Текст Текст.</p>
6 </div>
```

Пример документа с такими отчерками:

Общие положения

Настоящие Правила дорожного движения устанавливают единый порядок дорожного движения на всей территории Российской Федерации. Другие нормативные акты, касающиеся дорожного движения, должны основываться на требованиях Правил и не противоречить им.

Общие обязанности водителей

Водитель механического транспортного средства обязан иметь при себе и по требованию сотрудников милиции передавать им для проверки:

- водительское удостоверение на право управления транспортным средством соответствующей категории;
- ...и так далее...

Для красоты мы хотим, чтобы отчерк HR начинался левее, чем основной текст. Отрицательный `margin-left` нам поможет:

```
1 hr.margin { margin-left: -2em; }
2
3 /* для сравнения */
4 hr.position { position: relative; left: -2em; }
```

Результат:

Общие положения (hr.margin)

Настоящие Правила дорожного движения устанавливают единый порядок дорожного движения на всей территории Российской Федерации. Другие нормативные акты, касающиеся дорожного движения, должны основываться на требованиях Правил и не противоречить им.

Общие обязанности водителей (hr.position)

Водитель механического транспортного средства обязан иметь при себе и по требованию сотрудников милиции передавать им для проверки:

- водительское удостоверение на право управления транспортным средством соответствующей категории;
- ...и так далее...

Обратите внимание на разницу между методами сдвига!

➡ `hr.margin` сначала сдвинулся, а потом нарисовался до конца блока.

➡ `hr.position` сначала нарисовался, а потом сдвинулся — в результате справа осталось пустое пространство.

Уже отсюда видно, что отрицательные `margin` — исключительно полезное средство позиционирования!

Отрицательные `margin-right/bottom`

Отрицательные `margin-right/bottom` ведут себя по-другому, чем `margin-left/top`.

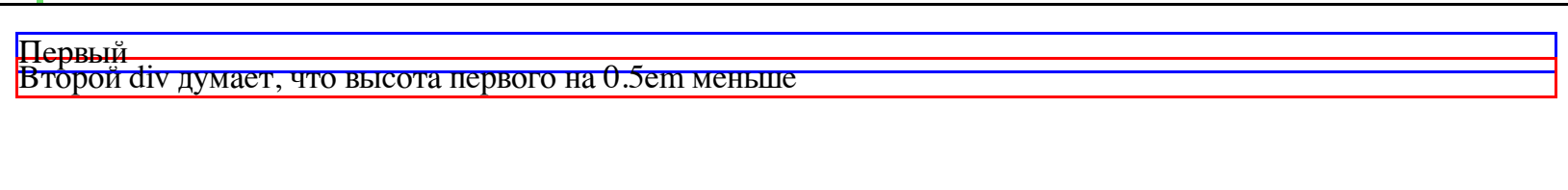
Они не сдвигают элемент, а «укорачивают» его. То есть, хотя сам размер блока не уменьшается, но следующий элемент будет думать, что он меньше на указанное в `margin-right/bottom` значение.

Например, в примере ниже вторая строка налезает на первую:


```

1 <div style="border: 1px solid blue; margin-bottom: -0.5em">
2   Первый
3 </div>
4
5 <div style="border: 1px solid red">
6   Второй div думает, что высота первого на 0.5em меньше
7 </div>

```



Это используют, в частности для красивых вносок, с приданием иллюзии глубины.

Например:

У DIV'a с холодильниками стоит `margin-bottom: -1em`

Наши холодильники - самые лучшие холодильники в мире! Наши холодильники - самые лучшие холодильники в мире! Наши холодильники - самые лучшие холодильники в мире! Наши холодильники - самые лучшие холодильники в мире!

Так считают: 5 человек

Оставьте свой отзыв!

Итого

- ➡ Отрицательные `margin-left/top` сдвигают элемент влево-вверх. Остальные элементы это учитывают, в отличие от сдвига через `position`.
- ➡ Отрицательные `margin-right/bottom` заставляют другие элементы думать, что блок меньше по размеру справа-внизу, чем он на самом деле.
- ➡ Отличная статья на тему отрицательных `margin`: [The Definitive Guide to Using Negative Margins \[8\]](#)

Свойство "display"

Свойство `display` имеет много разных значений. Обычно, используются только три из них: `none`, `inline` и `block`, потому что когда-то браузеры другие не поддерживали.

Но, после прихода IE7 и, особенно, IE8+, стало возможным использовать и другие значения тоже. Рассмотрим здесь весь список.

Значение none

Самое простое значение. Элемент не показывается, вообще. Как будто его и нет.

```
1 <div style="border:1px solid black">
2   Невидимый div (
3     <div style="display: none">Я - невидим!</div>
4   ) Стоит внутри скобок
5 </div>
```

Невидимый div () Стоит внутри скобок

Значение block

- ➡ Блочные элементы располагаются один над другим, вертикально (если нет особых свойств позиционирования, например float).
- ➡ Блок стремится расшириться на всю доступную ширину. Можно указать ширину и высоту явно.

Это значение display многие элементы имеют по умолчанию: DIV, заголовок, параграф.

```
1 <div style="border:1px solid black">
2   <div style="border: 1px solid blue; width: 50%">Первый</div>
3   <div style="border: 1px solid red">Второй</div>
4 </div>
```

Первый
Второй

Блоки прилегают друг к другу вплотную, если у них нет margin.

Значение inline

- ➡ Элементы располагаются на той же строке, последовательно.
- ➡ Ширина и высота элемента определяются по содержимому. Поменять их нельзя.

Например, инлайновые элементы по умолчанию: SPAN, ссылка.

```
1 <span style="border:1px solid black">
2   <span style="border: 1px solid blue; width:50%">Ширина</span>
3   <a style="border: 1px solid red">Игнорируется</a>
4 </span>
```

ШиринаИгнорируется

Во всём остальном — это блок, то есть:

- ➡ Элемент всегда прямоугольный.
- ➡ Работают свойства width/height.

Это значение `display` используют, чтобы отобразить в одну строку блочные элементы, в том числе разных размеров.

Например:

```
01 <style>
02 li {
03   display: inline-block;
04   list-style: none;
05   border: 1px solid red;
06 }
07 </style>
08
09 <ul style="border:1px solid black; padding:0">
10   <li>Инлайн Блок<br>3 строки<br>высота/ширина явно не заданы</li>
11   <li style="width:100px;height:100px">Инлайн<br>Блок 100x100</li>
12   <li style="width:60px;height:60px">Инлайн<br>Блок 60x60</li>
13   <li style="width:100px;height:60px">Инлайн<br>Блок 100x60</li>
14   <li style="width:60px;height:100px">Инлайн<br>Блок 60x100</li>
15 </ul>
```



Свойство `vertical-align` позволяет выровнять такие элементы внутри внешнего блока:

```

01 <style>
02 li {
03     display: inline-block;
04     list-style: none;
05     border: 1px solid red;
06     vertical-align: middle;
07 }
08 </style>
09
10 <ul style="border:1px solid black; padding:0">
11     <li>Инлайн Блок<br>3 строки<br>высота/ширина явно не заданы</li>
12     <li style="width:100px;height:100px">Инлайн<br>Блок 100x100</li>
13     <li style="width:60px;height:60px">Инлайн<br>Блок 60x60</li>
14     <li style="width:100px;height:60px">Инлайн<br>Блок 100x60</li>
15     <li style="width:60px;height:100px">Инлайн<br>Блок 60x100</li>
16 </ul>

```



Как и в случае с инлайн-элементами, пробелы между блоками появляются из-за пробелов в HTML. Если элементы списка идут вплотную, например, генерируются в JavaScript — их не будет.

IE7 допускает это значение только для элементов, которые по умолчанию inline.

Значения table-*

Современные браузеры (не IE6,7) позволяют описывать таблицу любыми элементами, если поставить им соответствующие значения display.

Для таблицы целиком table, для строки — table-row, для ячейки — table-cell и т.д.

Пример использования:

```

01 <form style="display: table">
02   <div style="display: table-row">
03     <label style="display: table-cell">Имя:</label>
04     <input style="display: table-cell">
05   </div>
06   <div style="display: table-row">
07     <label style="display: table-cell">Фамилия:</label>
08     <input style="display: table-cell">
09   </div>
10 </form>

```

Имя:

Фамилия:

Важно то, что это действительно полноценная таблица. Используются табличные алгоритмы вычисления ширины и высоты элемента, описанные в стандарте [9] .

Это хорошо для семантической вёрстки и позволяет избавиться от лишних тегов.

С точки зрения современного CSS, обычные TABLE, TR, TD и т.д. — это просто элементы с предопределёнными значениями display:

```

1 table { display: table }
2 tr { display: table-row }
3 thead { display: table-header-group }
4 tbody { display: table-row-group }
5 tfoot { display: table-footer-group }
6 col { display: table-column }
7 colgroup { display: table-column-group }
8 td, th { display: table-cell }
9 caption { display: table-caption }

```

Очень подробно об алгоритмах вычисления размеров и отображении таблиц рассказывает стандарт [CSS 2.1 - Tables \[10\]](#) .

Вертикальное центрирование с table-cell

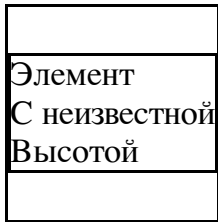
Внутри ячеек свойство [vertical-align \[11\]](#) выравнивает содержимое по вертикали.

Это можно использовать для центрирования:

```

1 <style>
2   div { border: 1px solid black }
3 </style>
4
5 <div style="height:100px; display: table-cell; vertical-align: middle">
6   <div>Элемент<br>С неизвестной<br>Высотой</div>
7 </div>

```



CSS не требует, чтобы вокруг `table-cell` была структура таблицы: `table-row` и т.п. Может быть просто такой одинокий DIV, это допустимо.

При этом он ведёт себя как ячейка TD, то есть подстраивается под размер содержимого и умеет вертикально центрировать его при помощи `vertical-align`.

Значение `display: table-cell` поддерживается во всех браузерах, кроме IE<8. В IE6,7 можно использовать для центрирования CSS-выражения или реальную таблицу.

Значения `list-item` и `run-in`

У свойства `display` есть и другие значения. Они используются реже, поэтому посмотрим на них кратко:

`list-item`

По умолчанию для списка. Элемент генерирует кроме блока с содержимым ещё и блок с номером(значком) списка:

```
<div style="display: list-item; list-style: inside square">Пункт 1</div>
```

■ Пункт 1

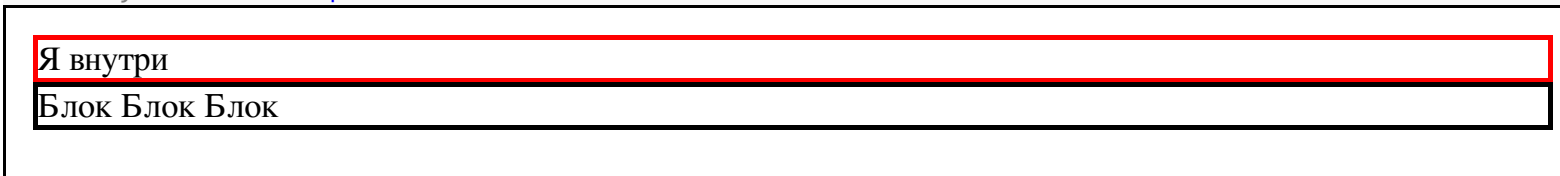
`run-in`

Если после `run-in` идёт block, то `run-in` становится его первым инлайн-элементом, то есть отображается в начале block.

В примере ниже первый DIV, благодаря `display: run-in`, окажется визуально внутри второго.

Это значение не поддерживается Firefox и IE<8, в этих браузерах пример отобразится как два обычных блока.

```
<div style="display: run-in; border: 2px solid red">Я внутри</div>  
<div style="border: 2px solid black">Блок Блок Блок</div>
```



Лишнее место под IMG

Иногда под IMG «вдруг» появляется лишнее место. Посмотрим на эти грабли внимательнее, почему такое бывает и как этого избежать.

Демонстрация проблемы

Например:

```
1 <style> * { margin: 0; padding: 0; } </style>  
2 <table>  
3   <tr>  
4     <td style="border:1px red solid">  
5         
6     </td>  
7   </tr>  
8 </table>
```



Посмотрите внимательно! Вы видите расстояние между рамками снизу? Это потому, что **браузер резервирует дополнительное место под инлайновым элементом, чтобы туда выносить «хвосты» букв.**

Вот так выглядит та же картинка с выступающим вниз символом рядом:



В примере картинка обёрнута в красный TD. Таблица подстраивается под размер содержимого, так что проблема явно видна. Но она касается не только таблицы. Аналогичная ситуация возникнет, если вокруг IMG будет другой элемент с явно указанным размером, «облегающий» картинку по высоте. Браузер постарается зарезервировать место под IMG, хотя оно там не нужно.

Решение: сделать элемент блочным

Самый лучший способ избежать этого — поставить `display: block` таким картинкам:


```

01 <style>
02   * { margin: 0; padding: 0; }
03   img { display: block }
04 </style>
05 <table>
06   <tr>
07     <td style="border:1px red solid">
08       
09     </td>
10   </tr>
11 </table>

```



Под блочным элементом ничего не резервируется. Проблема исчезла.

Решение: задать vertical-align

А что, если мы, по каким-то причинам, *не хотим* делать элемент блочным?

Существует ещё один способ избежать проблемы — использовать свойство [vertical-align \[12\]](#) .

Если установить vertical-align в top, то инлайн-элемент будет отпозиционирован по верхней границе текущей строки.

При этом браузер не будет оставлять место под изображением, так как запланирует продолжение строки сверху, а не снизу:

```

01 <style>
02   * { margin: 0; padding: 0; }
03   img { vertical-align: top }
04 </style>
05 <table>
06   <tr>
07     <td style="border:1px red solid">
08       
09     </td>
10   </tr>
11 </table>

```



А вот, как браузер отобразит соседние символы в этом случае: pp



С другой стороны, сама строка никуда не делась, изображение по-прежнему является её частью, а браузер планирует разместить другое текстовое содержимое рядом, хоть и сверху. Поэтому если изображение маленькое, то произойдёт дополнение пустым местом до высоты строки:

Например, для ``:



Таким образом, требуется уменьшить еще и `line-height` контейнера. Окончательное решение для маленьких изображений с `vertical-align`:

```
01 <style>
02   * { margin: 0; padding: 0; }
03   td { line-height: 1px }
04   img { vertical-align: top }
05 </style>
06 <table>
07   <tr>
08     <td style="border:1px red solid">
09       
10     </td>
11   </tr>
12 </table>
```

Результат:



Итого

- ➡ Пробелы под картинками появляются, чтобы оставить место под «хвосты» букв в строке. Строка «подразумевается», т.к. `display:inline`.
- ➡ Можно избежать пробела, если изменить `display`, например, на `block`.
- ➡ Альтернатива: `vertical-align:top` (или `bottom`), но для маленьких изображений может понадобиться уменьшить `line-height`, чтобы контейнер не оставлял место под строку.

Свойство "float"

Свойство `float` в CSS занимает особенное место. До его появления расположить два блока один слева от другого можно было лишь при помощи таблиц. Но в его работе есть ряд особенностей. Поэтому его иногда не любят, но при их понимании `float` станет вашим верным другом и помощником.

Далее мы рассмотрим, как работает `float`, разберём решения сопутствующих проблем, а также ряд полезных рецептов.

Как работает float

Синтаксис:

`float: left | right | none | inherit;`

При применении этого свойства происходит следующее:

1. Элемент позиционируется как обычно, а затем *вынимается из потока* и сдвигается влево (для `left`) или вправо (для `right`) до того как коснётся либо границы родителя, либо другого элемента с `float`.
2. Если пространства по горизонтали не хватает для того, чтобы вместить элемент, то он сдвигается вниз до тех пор, пока не начнёт помещаться.
3. Другие непозиционированные блочные элементы без `float` ведут себя так, как будто элемента с `float` нет, так как он убран из потока.
4. Строки (`inline`-элементы), напротив, «знают» о `float` и обтекают элемент по сторонам.

Ещё детали:

1. **Элемент при наличии `float` получает `display: block`.** То есть, указав элементу, у которого `display: inline` свойство `float: left/right`, мы автоматически сделаем его блочным. В частности, для него будут работать `width/height`.

Исключением являются некоторые редкие `display` наподобие `inline-table` и `run-in` (см. [Relationships between 'display', 'position', and 'float' \[13\]](#)).
2. **Ширина `float`-блока определяется по содержимому.** ([CSS 2.1, 10.3.5 \[14\]](#)).
3. **Вертикальные отступы `margin` элементов с `float` не сливаются с отступами соседей**, в отличие от обычных блочных элементов.

Это пока только теория. Далее мы рассмотрим происходящее на примере.

Текст с картинками

Одно из первых применений `float`, для которого это свойство когда-то было придумано — это вёрстка текста с картинками, отжатыми влево или вправо.

Например, вот текст о Винни-Пухе с картинками, которым поставлено свойство `float` ([ссылка на HTML \[15\]](#)):

Винни-Пух

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не только о Винни-Пухе, но и обычно тоже называется «Винни-Пух»), один из самых известных героев детской литературы. В 1960-е—1970-е годы, благодаря пересказу Заходера «Винни-Пух и все-все-все», а затем и фильмам «Союзмультфильм», где мишку озвучивал Евгений Леонидов, Винни-Пух стал очень популярен и в Советском Союзе.

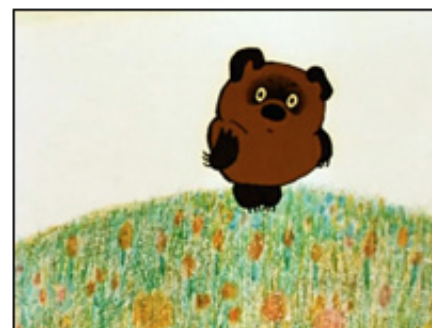
float: left



Первый перевод «Винни-Пуха» в СССР вышел в 1958 году в Литве (лит. Mikė Pūkuotukas), его выполнил 20-летний литовский писатель Виргилюс Чепайтис, пользовавшийся польским переводом Ирены Тувим. Впоследствии Чепайтис, познакомившись с английским оригиналом, с переработал свой перевод, переиздававшийся неоднократно.

История Винни-Пуха в России начинается с того же года, когда с книгой познакомился Борис Владимирович Заходер. В студии «Союзмультфильм» под руководством Фёдора Хитрука было создано три мультфильма. Сценарий написал Хитрук в соавторстве с Заходером; работа соавторов не всегда шла гладко, что стало в конечном счёте причиной прекращения выпуска мультфильмов (первоначально планировалось выпустить сериал по всей книге). Текст и картинки взяты с Wikipedia.

float: right



float: right



Его HTML-код ([открыть \[16\]](#)) выглядит примерно так:

```
1 
2 <p>Текст...</p>
3 <p>Текст...</p>
4
5 
6 <p>Текст...</p>
7
8 
9 <p>Текст...</p>
```

Каждая картинка, у которой есть float, обрабатывается в точности [по алгоритму \[17\]](#), указанному выше.

Посмотрим, например, как выглядело бы начало текста без float:



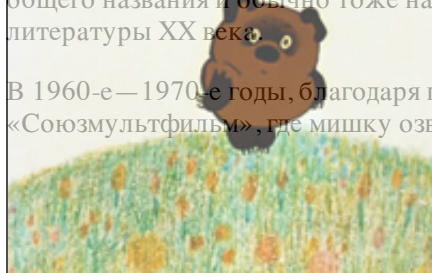
Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.

В 1960-е—1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.

1. Элемент IMG вынимается из документа потока. Иначе говоря, последующие блоки начинают вести себя так, как будто его нет, и заполняют освободившееся место (изображение для наглядности полупрозрачно):

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.

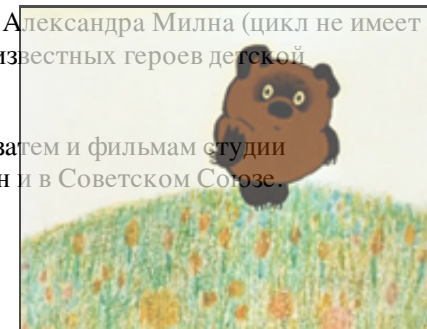
В 1960-е—1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.



2. Элемент IMG сдвигается максимально вправо(при float:right):

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.

В 1960-е — 1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.



3. Строки, в отличие от блочных элементов, «чувствуют» float и уступают ему место, обтекая картинку слева:

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.

В 1960-е — 1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.



При float:left — всё то же самое, только IMG смещается влево (или не смещается, если он и так у левого края), а строки — обтекают справа

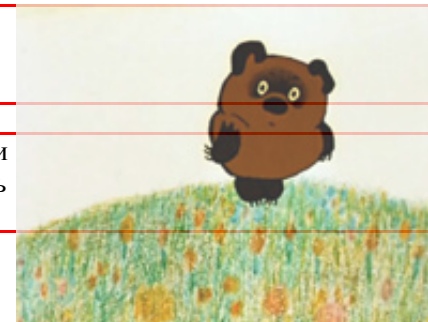
Строки и инлайн-элементы смещаются, чтобы уступить место float. Обычные блоки — ведут себя так, как будто элемента нет.

Чтобы это увидеть, добавим параграфам фон и рамку, а также сделаем изображение немного прозрачным:

Винни-Пух

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.

В 1960-е — 1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.



Как видно из рисунка, параграфы проходят «за» float. При этом строки в них о float 'ах знают и обтекают их, поэтому соответствующая часть параграфа пуста.

Блок с float

Свойство float можно поставить любому элементу, не обязательно картинке. При этом элемент станет блочным.

Посмотрим, как это работает, на конкретной задаче — сделать рамку с названием вокруг картинки с Винни.

HTML будет такой:

```
1 <h2>Винни-Пух</h2>
2
3 <div class="left-picture">
4   
5   <div>Кадр из советского мультфильма</div>
6 </div>
7
8 <p>Текст...</p>
```

..То есть, div.left-picture включает в себя картинку и заголовок к ней. Добавим стиль с float:

```
1 .left-picture {
2   float: left;
3
4   /* рамочка и отступ для красоты (не обязательно) */
5   margin: 0 10px 5px 0;
6   text-align: center;
7   border: 1px solid black;
8 }
```

Результат:

Винни-Пух



Кадр из советского мультфильма

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге).

Один из самых известных героев детской литературы XX века.

Заметим, что блок `div.left-picture` «обернул» картинку и текст под ней, а не растянулся на всю ширину. Это следствие того, что ширина блока с `float` определяется по содержимому.

Очистка под `float`

Разберём ещё одну особенность использования свойства `float`.

Для этого выведем персонажей из мультфильма «Винни-Пух». Цель:

Винни-Пух



Кадр из советского мультфильма

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге).

Один из самых известных героев детской литературы XX века.

Сова



Кадр из советского мультфильма

Персонаж мультфильма про Винни-Пуха. Очень умная.

Говорит редко, но чрезвычайно мудро.

Реализуем её, шаг за шагом.

Шаг 1. Добавляем информацию

Попробуем просто добавить Сову после Винни-Пуха:

```

1 <h2>Винни-Пух</h2>
2 <div class="left">Картинка</div>
3 <p>..Текст о Винни..</p>
4
5 <h2>Сова</h2>
6 <div class="left">Картинка</div>
7 <p>..Текст о Сове..</p>

```

Результат [такого кода \[18\]](#) будет странным, но предсказуемым:

Винни-Пух



Кадр из советского мультфильма

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге).

Один из самых известных героев детской литературы XX века.

Сова



Кадр из советского мультфильма

Персонаж мультфильма про Винни-Пуха. Очень умная.

Говорит редко, но чрезвычайно мудро.

Произошло следующее:

- ➔ **Заголовок <h2>Сова</h2> не заметил float** (он же блочный элемент) и расположился сразу после предыдущего параграфа <p>..Текст о Винни..</p>.
- ➔ После него идёт float-элемент — картинка «Сова». Он был сдвинут влево. Согласно [алгоритму \[19\]](#), он движется до левой границы или до касания с другим float-элементом, что и произошло (картинка «Винни-Пух»).

→ Так как у совы `float: left`, то **последующий текст обтекает её справа**.

Шаг 2. Свойство `clear`

Мы, конечно же, хотели бы расположить заголовок «Сова» и остальную информацию ниже Винни-Пуха.

Для решения возникшей проблемы придумано свойство `clear`.

Синтаксис:

```
clear: left | right | both;
```

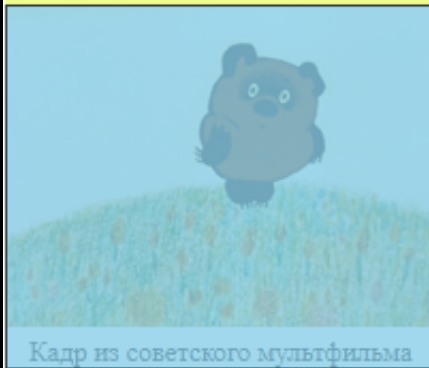
Применение этого свойства сдвигает элемент вниз до тех пор, пока не закончатся `float`'ы слева/справа/с обеих сторон.

Применим его к заголовку H2:

```
h2 {  
  clear: left;  
}
```

Результат [получившегося кода \[20\]](#) будет ближе к цели, но всё ещё не идеален:

Винни-Пух



Кадр из советского мультфильма

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге).

Один из самых известных героев детской литературы XX века.

отступ
маловат

Сова



Кадр из советского мультфильма

Персонаж мультфильма про Винни-Пуха. Очень умная.

Говорит редко, но чрезвычайно мудро.

Элементы теперь в нужном порядке. Но куда пропал отступ `margin-top` у заголовка «Сова»?

Теперь заголовок «Сова» прилегает снизу почти вплотную к картинке, с учётом её `margin-bottom`, но без своего большого отступа `margin-top`.

Таково поведение свойства `clear`. Оно сдвинуло элемент `h2` вниз ровно настолько, чтобы элементов `float` не было *сбоку от его верхней границы*.

Если посмотреть на элемент заголовка внимательно в инструментах разработчика, то можно заметить отступ `margin-top` у заголовка по-прежнему есть, но он располагается «за» элементом `float` и не учитывается при работе в `clear`.

Чтобы исправить ситуацию, можно добавить перед заголовком пустой промежуточный элемент без отступов, с единственным свойством `clear: both`. Тогда уже под ним отступ заголовка будет работать нормально:

```
1 <h2>Винни-Пух</h2>
2 <div class="left">Картинка</div>
3 <p>Текст</p>
4
5 <div style="clear:both"></div>
6
7 <h2>Сова</h2>
8 <div class="left">Картинка</div>
9 <p>Текст</p>
```

Результат [получившегося кода \[21\]](#) :

Винни-Пух



Кадр из советского мультфильма

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге).

Один из самых известных героев детской литературы XX века.

Сова



Кадр из советского мультфильма

Персонаж мультфильма про Винни-Пуха. Очень умная.

Говорит редко, но чрезвычайно мудро.

- ➡ Свойство `clear` гарантировало, что `<div style="clear:both">` будет под картинкой с `float`.
- ➡ Заголовок `<h2>Сова</h2>` идёт после этого `<div>`. Так что его отступ учитывается.

Заполнение блока-родителя

Итак, мы научились располагать другие элементы *под* `float`. Теперь рассмотрим следующую особенность.

Из-за того, что блок с `float` удалён из потока, родитель не выделяет под него места.

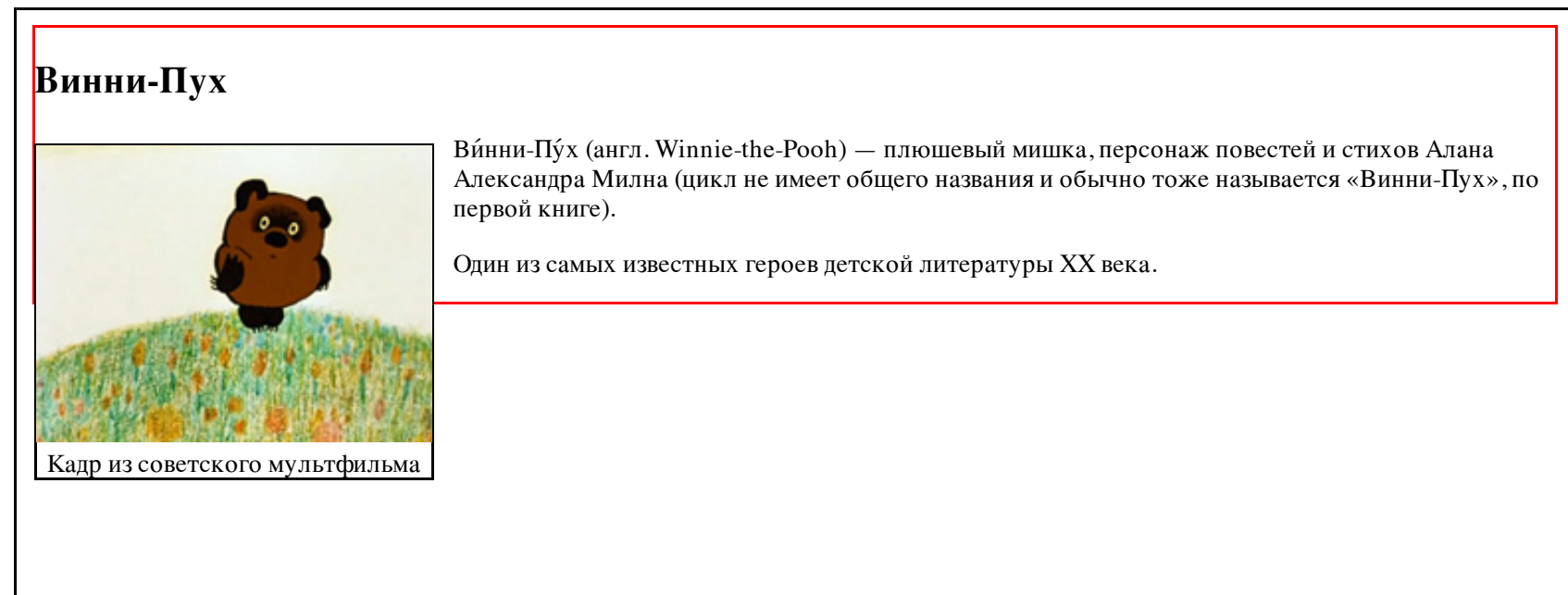
Например, выделим для информации о Винни-Пухе красивый элемент-контейнер `<div class="hero">`:

```
1 <div class="hero">
2
3   <h2>Винни-Пух</h2>
4
5   <div class="left">Картинка</div>
6
7   <p>Текст.</p>
8 </div>
```

Стиль контейнера:

```
1 .hero {
2   background: #D2B48C;
3   border: 1px solid red;
4 }
```

Результат [получившегося кода \[22\]](#) :



Элемент с `float` оказался выпавшим за границу родителя `.hero`.

Чтобы этого не происходило, используют одну из следующих техник.

Поставить родителю `float`

Элемент с `float` обязан расширяться, чтобы вместить вложенные `float`.

Поэтому, если это допустимо, то установка `float` контейнеру всё исправит:

```
1 .hero {  
2   background: #D2B48C;  
3   border: 1px solid red;  
4   float: left;  
5 }
```

Винни-Пух



Кадр из советского мультфильма

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге).

Один из самых известных героев детской литературы XX века.

Разумеется, не всегда можно поставить родителю float, так что смотрим дальше.

Добавить в родителя элемент с clear

Добавим элемент `div style="clear:both"` в самый конец контейнера `.hero`.

Он с одной стороны будет «нормальным» элементом, в потоке, и контейнер будет обязан выделить под него пространство, с другой — он знает о float и сместится вниз.

Соответственно, и контейнер вырастет в размере:

```
01 <div class="hero">  
02  
03   <h2>Винни-Пух</h2>  
04  
05   <div class="left">Картинка</div>  
06  
07   <p>Текст.</p>  
08  
09   <div style="clear:both"></div>  
10 </div>
```

Результат — правильное отображение, как и в примере выше. [Открыть код \[23\]](#)

Единственный недостаток этого метода — лишний HTML-элемент в разметке.

Универсальный класс clearfix

Чтобы не добавлять в HTML-код лишний элемент, можно задать его через `:after`.

```
1 .clearfix:after {  
2   content: "."; /* добавить содержимое: "." */  
3   display: block; /* сделать блоком, т.к. inline не может иметь clear */  
4   clear: both; /* с обеих сторон clear */  
5   visibility: hidden; /* сделать невидимым, зачем нам точка внизу? */  
6   height: 0; /* сделать высоту 0, чтобы не занимал место */  
7 }
```

Добавив этот класс к родителю, получим тот же результат, что и выше. [Открыть код \[24\]](#)

Псевдоселектор `:after` не поддерживается в IE<8, но для старых IE сработает другое свойство:

```
.clearfix {  
  zoom: 1; /* спец-свойство IE */  
}
```

overflow:auto/hidden

Если добавить родителю `overflow: hidden` или `overflow: auto`, то всё станет хорошо.

```
1 .hero {  
2   overflow: auto;  
3 }
```

Этот метод работает во всех браузерах. [Открыть код \[25\]](#) .

Несмотря на внешнюю странность, этот способ не является «хаком». Такое поведение прописано в спецификации CSS.

Однако, установка `overflow` может привести к появлению полосы прокрутки, способ с дополнительным элементом (или `.clearfix:after`, если без IE<8) более безопасен.

Еще применения float

Одно применение — для верстки текста, мы уже видели. Рассмотрим ещё несколько.

Галерея

При помощи `float` можно сделать галерею изображений:

```
01 <ul class="gallery">
02
03   <li>
04     
05     <div>Картинка 1</div>
06   </li>
07
08   <li>
09     
10     <div>Картинка 2</div>
11   </li>
12   ...
13 </ul>
```

Стиль:

```
1 .gallery li {
2   float: left;
3   width: 130px;
4   list-style: none;
5 }
```

Элементы `float: left` двигаются влево, а если это невозможно, то вниз, автоматически адаптируясь под ширину контейнера.

Результат:



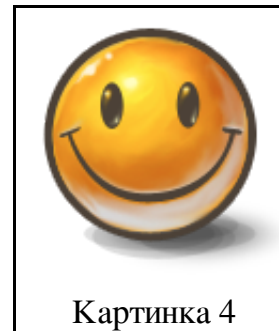
Картинка 1



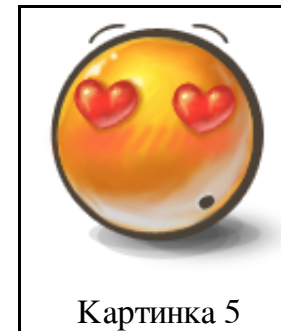
Картинка 2



Картинка 3



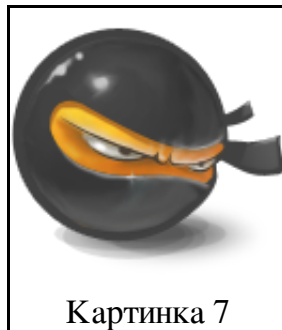
Картинка 4



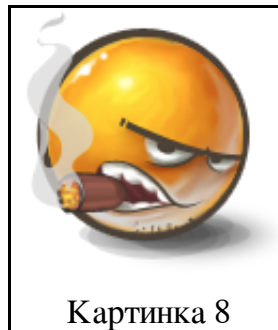
Картинка 5



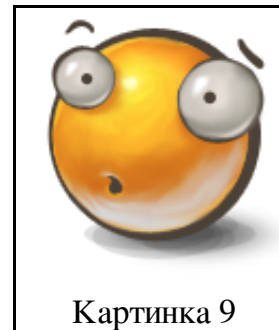
Картинка 6



Картинка 7



Картинка 8



Картинка 9

Вёрстка в несколько колонок

Свойство `float` позволяет делать несколько вертикальных колонок.

`float:left + float:right`

Например, для вёрстки в две колонки можно сделать два `<div>`. Первому указать `float:left` (левая колонка), а второму — `float:right` (правая колонка).

Чтобы они не ссорились, каждой колонке нужно дополнительно указать ширину:

```
1 <div>Шапка</div>
2 <div class="column-left">Левая колонка</div>
3 <div class="column-right">Правая колонка</div>
4 <div class="footer">Низ</div>
```

Стили:

```
01 .column-left {
02   float: left;
03   width: 30%;
04 }
05
06 .column-right {
07   float: left;
08   width: 70%;
09 }
10
11 .footer {
12   clear: both;
13 }
```

Результат (добавлены краски):

Шапка

Персонажи:

- Винни-Пух
- Ослик Иа
- Сова
- Кролик

Винни-Пух



Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.

В 1960-е—1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.

Низ


В эту структуру легко добавить больше колонок с разной шириной. Правой колонке можно было бы указать и `float: right`.

float + margin

Ещё вариант — сделать `float` для левой колонки, а правую оставить в потоке, но с отбивкой через `margin`:

```
01 .column-left {  
02   float: left;  
03   width: 30%;  
04 }  
05  
06 .column-right {  
07   margin-left: 30%;  
08 }  
09  
10 .footer {  
11   clear: both;  
12 }
```

Результат (добавлены краски):

Шапка	
Персонажи: <ul style="list-style-type: none">Винни-ПухОслик ИаСоваКролик	Винни-Пух  <p>Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.</p> <p>В 1960-е—1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.</p>
Низ	

В примере выше — показана небольшая проблема. Колонки не растягиваются до одинаковой высоты. Конечно, это не имеет значения,

если фон одинаковый, но что, если он разный?

Технически, нет кросс-браузерного CSS-способа сделать, чтобы они растягивались. Есть различные обходы и трюки, которые позволяют обойти проблему в ряде ситуаций, но они выходят за рамки нашего обсуждения. Если интересно — посмотрите, например, [Faux Columns \[26\]](#).

Техник построения структуры страницы и колонок («CSS layout») очень много. Более подробно вы можете с ними познакомиться в книгах и продвинутых статьях, либо просто придумать самому на основе того, как работает `float`.

Свойство "overflow"

Свойство `overflow` управляет тем, как ведёт себя содержимое блочного элемента, если его размер превышает допустимую длину/ширину.

Обычно блок увеличивается в размерах при добавлении в него элементов, заключая в себе всех потомков.

Но что, если высота/ширина указаны явно? Тогда блок не может увеличиться, и содержимое «переполняет» блок. Его отображение в этом случае задаётся свойством `overflow`.

Возможные значения

- ➡ `visible` (по умолчанию)
- ➡ `hidden`
- ➡ `scroll`
- ➡ `auto`

`visible`

Если не ставить `overflow` явно или поставить `visible`, то содержимое отображается за границами блока.

Например:

```

01 <style>
02 .overflow {
03   /* overflow не задан */
04   width: 200px;
05   height: 80px;
06   border: 1px solid black;
07 }
08 </style>
09
10 <div class="overflow">
11   visible ЭтотТекстВылезаетСправаЭтотТекстВылезаетСправа
12   Этот текст вылезает снизу Этот текст вылезает снизу
13   Этот текст вылезает снизу Этот текст вылезает снизу
14 </div>

```

visible
ЭтотТекстВылезаетСправаЭтотТекстВылезаетСправа
Этот текст вылезает снизу
Этот текст вылезает снизу
Этот текст вылезает снизу
Этот текст вылезает снизу

Как правило, такое переполнение указывает на ошибку в вёрстке. Если содержимое может быть больше контейнера — используют другие значения.

hidden

Переполняющее содержимое не отображается.

```

01 <style>
02 .overflow {
03     overflow: hidden;
04     width: 200px;
05     height: 80px;
06     border: 1px solid black;
07 }
08 </style>
09
10 <div class="overflow">
11     hidden ЭтотТекстОбрезанСправаЭтотТекстОбрезанСправа
12     Этот текст будет обрезан снизу Этот текст будет обрезан снизу
13     Этот текст будет обрезан снизу Этот текст будет обрезан снизу
14 </div>

```

hidden
ЭтотТекстОбрезанСправаЭт
Этот текст будет обрезан
снизу Этот текст будет
обрезан снизу Этот текст

Вылезавшее за границу содержимое становится недоступно.

Это свойство иногда используют от лени, когда какой-то элемент дизайна немного вылезает за границу, и вместо исправления вёрстки его просто скрывают. Как правило, долго оно не живёт, вёрстку всё равно приходится исправлять.

auto

При переполнении отображается полоса прокрутки.


```

01 <style>
02 .overflow {
03     overflow: auto;
04     width: 200px;
05     height: 100px;
06     border: 1px solid black;
07 }
08 </style>
09
10 <div class="overflow">
11     auto ЭтотТекстДастПрокруткуСправаЭтотТекстДастПрокруткуСправа
12     Этот текст даст прокрутку снизу Этот текст даст прокрутку снизу
13     Этот текст даст прокрутку снизу
14 </div>

```

auto
ЭтотТекстДастПрокруткуСп
Этот текст даст прокрутку
снизу Этот текст даст
прокрутку снизу Этот текст
даст прокрутку снизу

scroll

Полоса прокрутки отображается всегда.

```

01 <style>
02 .overflow {
03     overflow: scroll;
04     width: 200px;
05     height: 80px;
06     border: 1px solid black;
07 }
08 </style>
09
10 <div class="overflow">
11     scroll
12     Переполнения нет.
13 </div>

```

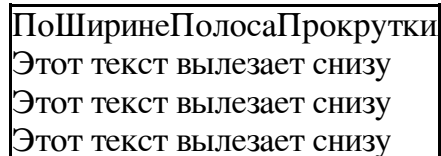
scroll Переполнения нет.

То же самое, что auto, но полоса прокрутки видна всегда, даже если переполнения нет.

overflow-x, overflow-y

Можно указать поведение блока при переполнении по ширине в `overflow-x` и высоте — в `overflow-y`:

```
01 <style>
02   .overflow {
03     overflow-x: auto;
04     overflow-y: hidden;
05     width: 200px;
06     height: 80px;
07     border: 1px solid black;
08   }
09 </style>
10
11 <div class="overflow">
12   ПоШиринеПолосаПрокруткиAutoПоШиринеПолосаПрокруткиAuto
13   Этот текст вылезает снизу Этот текст вылезает снизу
14   Этот текст вылезает снизу Этот текст вылезает снизу
15 </div>
```



Итого

Свойства `overflow-x/overflow-y` (или оба одновременно: `overflow`) задают поведение контейнера при переполнении:

visible

По умолчанию, содержимое вылезает за границы блока.

hidden

Переполняющее содержимое невидимо.

auto

Полоса прокрутки при переполнении.

scroll

Полоса прокрутки всегда.

Кроме того, значение `overflow: auto | hidden` изменяет поведение контейнера, содержащего `float`'ы. Так как элемент с `float` находится вне потока, то обычно контейнер не выделяет под него место. Но если стоит такой `overflow`, то место выделяется, т.е. контейнер растягивается. Более подробно этот вопрос рассмотрен в статье [Свойство «float» \[27\]](#).

Свойство "position"

Свойство `position` позволяет сдвигать элемент со своего обычного места. Цель этой главы — не только напомнить, как оно работает, но и разобрать ряд частых заблуждений и граблей.

position: static

Статическое позиционирование производится по умолчанию, в том случае, если свойство position не указано.

Его можно также явно указать через CSS-свойство:

```
position: static;
```

Такая запись встречается редко и используется для переопределения других значений position.

Здесь и далее, для примеров мы будем использовать следующий документ:

```
1 <div style="background: #aef; width: 500px">
2   Без позиционирования ("position: static").
3
4   <h2 style="background: #fee; margin: 0">Заголовок</h2>
5
6   <div>А тут - всякий разный текст... <br/>
7     ... В две строки!</div>
8 </div>
```

Без позиционирования ("position: static").

Заголовок

А тут - всякий разный текст...

... В две строки!

В этом документе сейчас все элементы отпозиционированы статически, то есть никак.

Элемент с position: static еще называют *не позиционированным*.

position: relative

Относительное позиционирование сдвигает элемент относительно его обычного положения.

Для того, чтобы применить относительное позиционирование, необходимо указать элементу CSS-свойство position: relative и координаты left/right/top/bottom.

Этот стиль сдвинет элемент на 10 пикселей относительно обычной позиции по вертикали:

```
position: relative;  
top: 10px;
```

```
01 <style>  
02   h2 {  
03     position: relative;  
04     top: 10px;  
05   }  
06 </style>  
07  
08 <div style="background: #aef; width: 500px">  
09   Заголовок сдвинут на 10px вниз.  
10  
11   <h2 style="background: #fee; margin: 0;">Заголовок</h2>  
12  
13   <div>А тут - всякий разный текст... <br/>  
14     ... В две строки!</div>  
15 </div>
```

Заголовок сдвинут на 10px вниз.

Заголовок

А тут - всякий разный текст...

... В две строки!

Координаты

Для сдвига можно использовать координаты:

- ➡ top - сдвиг от «обычной» верхней границы
- ➡ bottom - сдвиг от нижней границы
- ➡ left - сдвиг слева
- ➡ right - сдвиг справа

Не будут работать одновременно указанные top и bottom, left и right. Нужно использовать только одну границу из каждой пары.

Возможны отрицательные координаты и координаты, использующие другие единицы измерения. Например, left: 10% сдвинет элемент на 10% его ширины вправо, а left: -10% — влево. При этом часть элемента может оказаться за границей окна:

```

01 <style>
02   h2 {
03     position: relative;
04     left: -10%;
05   }
06 </style>
07
08 <div style="background: #aef; width: 500px">
09   Заголовок сдвинут на 10% влево.
10
11   <h2 style="background: #fee; margin: 0;">Заголовок</h2>
12
13   <div>А тут - всякий разный текст... <br/>
14     ... В две строки!</div>
15 </div>

```

Заголовок сдвинут на 10% влево.

Заголовок

А тут - всякий разный текст...

... В две строки!

Свойства `left/top` не будут работать для `position: static`. Если их все же поставить, браузер их проигнорирует. Эти свойства предназначены для работы только с позиционированными элементами.

position: absolute

Синтаксис:

```
position: absolute;
```

Абсолютное позиционирование делает две вещи:

1. **Элемент исчезает с того места, где он должен быть и позиционируется заново.** Остальные элементы, располагаются так, как будто этого элемента никогда не было.
2. **Координаты `top/bottom/left/right` для нового местоположения отсчитываются от ближайшего позиционированного родителя**, т.е. родителя с позиционированием, отличным от `static`. Если такого родителя нет — то относительно документа.

Кроме того:

- ➡ **Ширина элемента с `position: absolute` устанавливается по содержимому.** Детали алгоритма вычисления ширины описаны в стандарте [28].
- ➡ **Элемент получает `display: block`**, который перекрывает почти все возможные `display` (см. [Relationships between 'display'](#),

'position', and 'float' [29]).

Например, отпозиционируем заголовок в правом-верхнем углу документа:

```
01 <style>
02   h2 {
03     position: absolute;
04     right: 0;
05     top: 0;
06   }
07 </style>
08
09 <div style="background: #aef; width: 500px">
10   Заголовок в правом-верхнем углу документа.
11
12   <h2 style="background: #fee; margin: 0;">Заголовок</h2>
13
14   <div>А тут - всякий разный текст... <br/>
15     ... В две строки!</div>
16 </div>
```

Заголовок в правом-верхнем углу документа.
А тут - всякий разный текст...
... В две строки!

Заголовок

Важное отличие от `relative`: **так как элемент удаляется со своего обычного места, то элементы под ним сдвигаются, занимая освободившееся пространство**. Это видно в примере выше: строки идут одна за другой.

Так как при `position: absolute` размер блока устанавливается по содержимому, то широкий Заголовок «съёжился» до прямоугольника в углу.

Иногда бывает нужно поменять элементу `position` на `absolute`, но так, чтобы элементы вокруг не сдвигались. Как правило это делают, меняя соседей — добавляют `margin/padding` или вставляют в документ пустой элемент с такими же размерами.



Одновременное указание left/right, top/bottom

В абсолютно позиционированном элементе можно одновременно задавать противоположные границы.

Браузер растянёт такой элемент до границ. Работает везде, кроме IE6:

```
1 <style>
2 div {
3   position: absolute;
4   left: 10px; right: 10px; top: 10px; bottom: 10px;
5 }
6 </style>
7 <div style="background:#aef;text-align:center">10px от границ</div>
```

10px от границ



Внешним блоком является окно

Как растянуть абсолютно позиционированный блок на всю ширину документа?

Первое, что может прийти в голову:

```
1 | div {  
2 |   position: absolute;  
3 |   left: 0; top: 0; /* в левый-верхний угол */  
4 |   width: 100%; height: 100%; /* .. и растянуть */  
5 | }
```

Но это будет работать лишь до тех пор, пока у страницы не появится скроллинг!

Прокрутите вниз ифрейм:

Прокрутите меня...

Вы увидите, что голубой фон оканчивается задолго до конца документа.

Дело в том, что в CSS 100% относится к ширине внешнего блока («containing block»). А какой внешний блок имеется в виду здесь, ведь элемент изъят со своего обычного места?

В данном случае им является так называемый (*"initial containing block" [30]*), которым является окно, а не документ.

То есть, координаты и ширины вычисляются относительно окна, а не документа.

Вопрос: а получится ли так?

```
1 | div {  
2 |   position: absolute;  
3 |   left: 0; top: 0; /* в левый-верхний угол, и растянуть.. */  
4 |   right: 0; bottom: 0; /* ..указанием противоположных границ */  
5 | }
```

Нет, не получится. Координаты top/right/left/bottom вычисляются относительно окна.

Значение bottom: 0 — нижняя граница окна, а не документа, блок растянется до неё. То есть, будет то же самое, что и в предыдущем примере.

position: absolute в позиционированном родителе

Если у элемента есть позиционированный предок, то `position: absolute` работает относительно него, а не относительно документа.

То есть, достаточно поставить родительскому `div` позицию `relative`, даже без координат — и заголовок будет в его правом-верхнем углу, вот так:

```
01 <style>
02   h2 {
03     position: absolute;
04     right: 0;
05     top: 0;
06   }
07 </style>
08
09 <div style="background: #aef; width: 500px; position: relative">
10   Заголовок в правом-верхнем углу DIV'a.
11
12   <h2 style="background: #fee; margin: 0;">Заголовок</h2>
13
14   <div>А тут - всякий разный текст... <br/>
15     ... В две строки!</div>
16 </div>
```

Заголовок в правом-верхнем углу DIV'a.
А тут - всякий разный текст...
... В две строки!

Заголовок

Нужно пользоваться таким позиционированием с осторожностью, т.к оно может перекрыть текст. Этим оно отличается от `float`.

Сравните:

➡ Используем `position` для размещения элемента управления:

```
1 <button style="position: absolute; right: 10px; opacity: 0.8">
2   Кнопка
3 </button>
4 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
5 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
```

1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 Кнопка
6 7 8 9 1 2 3 4 5 6 7 8 9

Часть текста перекрывается. Кнопка более не участвует в потоке.

➡ Используем float для размещения элемента управления:

```
1 <button style="float: right; margin-right: 10px; opacity: 0.8;">
2   Кнопка
3 </button>
4 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
5 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
```

1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 Кнопка
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9

Браузер освобождает место справа, текст перенесён. Кнопка продолжает находиться в потоке, просто сдвинута.

position: fixed

Это подвид абсолютного позиционирования.

Синтаксис:

```
position: fixed;
```

Позиционирует объект точно так же, как absolute, но относительно window.

Разница в нескольких словах:

Когда страницу прокручивают, фиксированный элемент остается на своем месте и не прокручивается вместе со страницей.

В следующем примере, при прокрутке документа, ссылка #top всегда остается на своем месте.

```

01 <style>
02 #top {
03     position: fixed;
04     right: 10px;
05     top: 10px;
06     background: #fee;
07 }
08 </style>
09
10 <a href="#" id="top">Наверх (остается при прокрутке)</a>
11
12 Фиксированное позиционирование.
13
14 <p>Текст страницы.. Прокрути меня...</p>
15 <p>Много строк..</p><p>Много строк..</p>
16 <p>Много строк..</p><p>Много строк..</p>
17 <p>Много строк..</p><p>Много строк..</p>
18 <p>Много строк..</p><p>Много строк..</p>

```

Фиксированное позиционирование.

[Наверх \(остается при прокрутке\)](#)

Текст страницы.. Прокрути меня...

Много строк..

Много строк..

Много строк..

Много строк..

Поддерживается во всех современных браузерах, в IE начиная с версии 7. Для реализации аналогичного функционала в IE6 используют CSS-выражения.

Итого

Виды позиционирования и их особенности.

static

Иначе называется «без позиционирования». В явном виде задаётся только если надо переопределить другое правило CSS.

relative

Сдвигает элемент относительно текущего места.

- ➡ Противоположные границы left/right (top/bottom) одновременно указать нельзя.
- ➡ Окружающие элементы ведут себя так, как будто элемент не сдвигался.

absolute

Визуально переносит элемент на новое место.

Новое место вычисляется по координатам left/top/right/bottom относительно ближайшего позиционированного родителя.

Если такого родителя нет, то им считается окно.

- ➡ Ширина элемента по умолчанию устанавливается по содержимому.
- ➡ Можно указать противоположные границы `left/right (top/bottom)`. Элемент растянется. Возможность не поддерживается в IE<8.
- ➡ Окружающие элементы заполняют освободившееся место.

fixed

Подвид абсолютного позиционирования, при котором элемент привязывается к координатам окна, а не документа.

При прокрутке он остаётся на том же месте.

Почитать

CSS-позиционирование по-настоящему глубоко в спецификации [Visual Formatting Model, 9.3 и ниже \[31\]](#).

Еще есть хорошее руководство [CSS Positioning in 10 steps \[32\]](#), которое охватывает основные типы позиционирования.

Особенности свойства "height" в %

Обычно свойство `height`, указанное в процентах, означает высоту относительно внешнего блока.

Но это работает не всегда.

Точнее говоря, для этого нужно чтобы:

- ➡ высота внешнего блока указана явно.
- ➡ или элемент с `height` абсолютно позиционирован.

Для произвольного блочного элемента height в процентах работать не будет!

Чтобы лучше понимать ситуацию, рассмотрим пример из практики.

Пример

Наша цель — получить вёрстку такого вида:

	<p>Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.</p> <p>В 1960-е—1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.</p>
--	---

При этом блок с левой стрелкой должен быть отдельным элементом внутри контейнера. Это удобно для интеграции с JavaScript, чтобы отлавливать на нём клики мыши.

То есть, HTML-код требуется такой:

```

1 <div class="container">
2   <div class="toggler">
3     <!-- стрелка влево при помощи CSS, ширина фиксирована -->
4   </div>
5   <div class="content">
6     ...Текст...
7   </div>
8 </div>

```

Как это реализовать? Подумайте перед тем, как читать дальше...

Придумали?.. Если да — продолжаем.

Есть разные варианты, но, возможно, вы решили сдвинуть `.toggler` влево, при помощи `float:left` (тем более что он фиксированной ширины), а блок `.content` — отжать вправо, используя `margin`. Классическая двухколоночная вёрстка.

Затем элементу `.toggler` нужно назначить `height: 100%`, чтобы он занял всё пространство по вертикали.

Попытка `height:100% + float:left`

CSS:

```

01 .container {
02   border: 1px solid black;
03 }
04
05 .content {
06   /* Содержимое отодвинем вправо при помощи margin-left */
07   margin-left: 35px;
08 }
09
10 .toggler {
11   /* Зададим размеры блока со стрелкой */
12   height: 100%;
13   width: 30px;
14   float: left;
15
16   background: #EEE url("arrow_left.png") center center no-repeat;
17   border-right: #AAA 1px solid;
18   cursor: pointer;
19 }

```

А теперь — посмотрим этот вариант в действии:

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.

В 1960-е—1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.

Как видно, блок со стрелкой вообще исчез! Куда же он подевался?

Ответ нам даст спецификация CSS 2.1 [пункт 10.5 \[33\]](#) .

«Если высота внешнего блока вычисляется по содержимому, то высота в % не работает, и заменяется на `height:auto`. Кроме случая, когда у элемента стоит `position:absolute`.»

В нашем случае высота `.container` как раз определяется по содержимому, поэтому для `.toggler` проценты не действуют, а размер вычисляется как при `height:auto`.

Какая же она — эта автоматическая высота? Вспоминаем, что обычно размеры `float` определяются по содержимому ([10.3.5 \[34\]](#)). А содержимого-то в `.toggler` нет, так что высота нулевая. Поэтому этот блок и не виден.

Если бы мы точно знали высоту внешнего блока и добавили её в CSS — это решило бы проблему.

Например:

```
.container {  
  height: 200px; /* теперь height в % внутри будет работать */  
}
```

Результат:

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.

В 1960-е—1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.

...Но приемлем ли этот путь в данном случае? Нет, ведь мы не знаем, сколько будет текста и какой будет высота.

Поэтому решим задачу по-другому.

Решение: `height:100%` + `position:absolute`

Проценты будут работать, если поставить `.toggler` свойство `position: absolute` и спозиционировать его в левом-верхнем углу `.container` (у которого сделать `position:relative`):

```

01 .container {
02   position: relative;
03   border: 1px solid black;
04 }
05
06 .content {
07   margin-left: 35px;
08 }
09
10 .toggler {
11   position: absolute;
12   left: 0;
13   top: 0;
14
15   height: 100%;
16   width: 30px;
17   cursor: pointer;
18
19   border-right: #AAA 1px solid;
20   background: #EEE url("arrow_left.png") center center no-repeat;
21 }

```

Результат:

Винни-Пух (англ. Winnie-the-Pooh) — плюшевый мишка, персонаж повестей и стихов Алана Александра Милна (цикл не имеет общего названия и обычно тоже называется «Винни-Пух», по первой книге). Один из самых известных героев детской литературы XX века.

В 1960-е—1970-е годы, благодаря пересказу Бориса Заходера «Винни-Пух и все-все-все», а затем и фильмам студии «Союзмультфильм», где мишку озвучивал Евгений Леонов, Винни-Пух стал очень популярен и в Советском Союзе.

Итого

- ➡ Свойство `height`, указанное в %, работает только если для внешнего блока указана высота.
- ➡ Стандарт CSS 2.1 предоставляет обход этой проблемы, отдельно указывая, что проценты работают при `position: absolute`. На практике это часто выручает.

Знаете ли вы селекторы?

CSS3-селекторы — фундаментально полезная вещь.

Даже если вы почему-то (старый IE?) не пользуетесь ими в CSS, есть много фреймворков для их кросс-браузерного использования CSS3 из JavaScript.

Поэтому эти селекторы необходимо знать.

Основные виды селекторов

Основных видов селекторов всего несколько:

- `*` — любые элементы.
- `div` — элементы с таким тегом.
- `#id` — элемент с данным `id`.
- `.class` — элементы с таким классом.
- `[name="value"]` — селекторы на атрибут (см. далее).
- `:visited` — псевдофильтры, остальные разные условия на элемент (см. далее).

Селекторы можно комбинировать, записывая последовательно, без пробела:

- `.c1.c2` — элементы одновременно с двумя классами `c1` и `c2`
- `a#id.c1.c2:visited` — элемент `a` с данным `id`, классами `c1` и `c2`, и псевдофильтром `visited`

Отношения

В CSS3 предусмотрено четыре вида отношений между элементами.

Самые известные вы наверняка знаете:

- `div p` — элементы `p`, являющиеся потомками `div`.
- `div > p` — только непосредственные потомки

Есть и два более редких:

- `div ~ p` — правые соседи: все `p` на том же уровне вложенности, которые идут после `div`.
- `div + p` — первый правый сосед: `p` на том же уровне вложенности, который идёт сразу после `div` (если есть).

Посмотрим их на примере HTML:

```
1 <h3>Балтославянские языки</h3>
2
3 <ol id="languages">
4   ...Вложенный OL/LI список языков...
5 </ol>
```



```

01 #languages li {
02     color: brown; /* потомки #languages, подходящие под селектор LI */
03 }
04
05 #languages > li {
06     color: black; /* первый уровень детей #languages подходящих под LI */
07 }
08
09 #e-slavic { font-style: italic; }
10
11 #e-slavic ~ li { /* правые соседи #e-slavic с селектором LI */
12     color: red;
13 }
14
15 #latvian { font-style: italic; }
16
17 #latvian * { /* потомки #latvian, подходяще под * (т.е. любые) */
18     font-style: normal;
19 }
20
21 #latvian + li { /* первый правый сосед #latvian с селектором LI */
22     color: green;
23 }

```

Пример:

Балтославянские языки

1. Славянские языки

1. Славянские микроязыки
2. Праславянский язык
3. Восточнославянские языки `#e-slavic`
4. Западнославянские языки `#e-slavic ~ li`
5. Южнославянские языки `#e-slavic ~ li`
6. ... `#e-slavic ~ li`

2. Балтийские языки

1. Литовский язык
2. Латвийский язык `#latvian`
 1. Латгальский язык `#latvian *`
3. Прусский язык `#latvian + li`
4. ... (следующий элемент уже не `#latvian + li`)

Фильтр по месту среди соседей

При выборе элемента можно указать его место среди соседей.

Список селекторов для этого:

- `:first-child` — первый потомок своего родителя.
- `:last-child` — последний потомок своего родителя.
- `:only-child` — единственный потомок своего родителя, соседних элементов нет.
- `:nth-child(a)` — потомок номер a своего родителя, например `:nth-child(2)` — второй потомок. Нумерация начинается с 1.
- `:nth-child(an+b)` — расширение предыдущего селектора через указание номера потомка формулой, где a, b — константы, а под n подразумевается любое целое число.

Этот селектор будет фильтровать все элементы, которые попадают под формулу при каком-либо n . Например:

- `:nth-child(2n)` даст элементы номер 2, 4, 6..., то есть чётные.
- `:nth-child(2n+1)` даст элементы номер 1, 3..., то есть нечётные.
- `:nth-child(3n+2)` даст элементы номер 2, 5, 8 и так далее.

Пример использования для выделения в списке:

- Древнерусский язык
- Древненовгородский диалект `li:nth-child(2n)`
- **Западнорусский письменный язык** `li:nth-child(3)`
- Украинский язык `li:nth-child(2n)`
- Белорусский язык
- Другие языки `li:nth-child(2n)`

```
1 li:nth-child(2n) { /* чётные */
2   background: #eee;
3 }
4
5 li:nth-child(3) { /* 3-ий потомок */
6   color: red;
7 }
```

- `:nth-last-child(a)`, `:nth-last-child(an+b)` — то же самое, но отсчёт начинается с конца, например `:nth-last-child(2)` — второй элемент с конца.

Фильтр по месту среди соседей с тем же тегом

Есть аналогичные селекторы, которые учитывают не всех соседей, а только с тем же тегом:

- `:first-of-type`
- `:last-of-type`
- `:only-of-type`
- `:nth-of-type`
- `:nth-last-of-type`

Они имеют в точности тот же смысл, что и обычные `:first-child`, `:last-child` и так далее, но во время подсчёта игнорируют элементы с другими тегами, чем тот, к которому применяется фильтр.

Пример использования для раскраски списка DT «через один» и предпоследнего DD:

Первый dt

Описание dd

Второй dt `dt:nth-of-type(2n)`

Описание dd

Третий dt

Описание dd `dd:nth-last-of-type(2)`

Четвёртый dt `dt:nth-of-type(2n)`

Описание dd

```
1 dt:nth-of-type(2n) {
2   /* чётные dt (соседи с другими тегами игнорируются) */
3   background: #eee;
4 }
5
6 dd:nth-last-of-type(2) {
7   /* второй dd снизу */
8   color: red;
9 }
```

Как видим, селектор `dt:nth-of-type(2n)` выбрал каждый второй элемент dt, причём другие элементы (dd) в подсчётах не участвовали.

Селекторы атрибутов

На атрибут целиком

- ➡ `[attr]` — атрибут установлен,
- ➡ `[attr="val"]` — атрибут равен val.

На начало атрибута

- ➡ `[attr^="val"]` — атрибут начинается с val, например "value".
- ➡ `[attr|="val"]` — атрибут равен val или начинается с val-, например равен "val-1".

На содержание

- ➡ `[attr*="val"]` — атрибут содержит подстроку val, например равен "myvalue".
- ➡ `[attr~="val"]` — атрибут содержит val как одно из значений через пробел.
Например: `[attr~="delete"]` верно для "edit delete" и неверно для "undelete" или "no-delete".

На конец атрибута

- ➡ `[attr$="val"]` — атрибут заканчивается на val, например равен "myval".

Другие псевдофильтры

- ➡ `:not(селектор)` — все, кроме подходящих под селектор.
- ➡ `:focus` — в фокусе.

- `:hover` — под мышью.
- `:empty` — без детей (даже без текстовых).
- `:checked`, `:disabled`, `:enabled` — состояния INPUT.
- `:target` — этот фильтр сработает для элемента, ID которого совпадает с анкором `#...` текущего URL.

Например, если на странице есть элемент с `id="intro"`, то правило `:target { color: red }` подсветит его в том случае, если текущий URL имеет вид `http://...#intro`.

Псевдоэлементы `::before`, `::after`

«Псевдоэлементы» — различные вспомогательные элементы, которые браузер записывает или может записать в документ.

При помощи *псевдоэлементов* `::before` и `::after` можно добавлять содержимое в начало и конец элемента:

```
01 <style>
02 li::before {
03   content: " [[ ";
04 }
05
06 li::after {
07   content: " ]] ";
08 }
09 </style>
10
11 Обратите внимание: содержимое добавляется <b>внутри</b> LI.
12
13 <ul>
14   <li>Первый элемент</li>
15   <li>Второй элемент</li>
16 </ul>
```

Обратите внимание: содержимое добавляется **внутри** LI.

- [[Первый элемент]]
- [[Второй элемент]]

Версии IE<8 не понимают этих селекторов.

Когда-то все браузеры реализовали эти псевдоэлементы с одним двоеточием: `:after`/`:before`. Стандарт с тех пор изменился и сейчас все, кроме IE8, понимают также современную запись с двумя двоеточиями. А для IE8 нужно по-прежнему одно. Поэтому обычно её и используют.

Псевдоэлементы не являются селекторами, запись с одним двоеточием существует по историческим причинам.

Практика

Вы можете использовать информацию выше как справочную для решения задач ниже, которые уже реально покажут, владеете вы CSS-селекторами или нет.

CSS без IE6(7)

CSS-возможности, которыми мы можем пользоваться, если НЕ поддерживаем IE6.

Селекторы атрибутов:

- ➡ `[attr]` — атрибут установлен,
- ➡ `[attr="val"]` — атрибут равен `val`,
- ➡ `[attr^="val"]` — атрибут начинается с `val`, например `"value"`.
- ➡ `[attr*="val"]` — атрибут содержит `val`, например равен `"myvalue"`.
- ➡ `[attr$="val"]` — атрибут заканчивается на `val`, например равен `"myval"`.
- ➡ `[attr~="val"]` — атрибут содержит `val` как одно из значений через пробел, например: `[data-actions~="edit"]` верно для значения `data-actions="edit delete"`.
- ➡ `[attr|=“val”]` — атрибут равен `val` или начинается с `val-`, например равен `"val-1"`.

Селекторы элементов:

- ➡ `ul > li` — непосредственный потомок,
- ➡ `.prev + .me` — выбирает `.me`, которые стоят сразу после `.prev`, т.е. «правый брат».
- ➡ `.prev ~ .me` — выбирает `.me`, которые стоят после `.prev`, но не обязательно сразу после, между ними могут быть другие элементы,
- ➡ `.a.b` — несколько классов одновременно,
- ➡ `:hover` — курсор над элементом (в IE6 работает только с A),
- ➡ `:first-child` — первый потомок в своём родителе.

Внимание, IE7 не пересчитывает стили при изменении окружающих элементов для селекторов `.prev + .me`, `.prev` и `:first-child`. Иными словами, не обновляет стиль при добавлении/удалении соседей через JavaScript.

Свойства:

- ➡ `min-width/min-height` — минимальная ширина/высота
- ➡ `max-width/max-height` — максимальная ширина/высота
- ➡ `position: fixed`

Здесь перечислены в основном возможности. Разумеется, была поправлена и масса багов.

При отказе от поддержки IE7, и, тем более, IE8, список ещё шире и включает в себя почти весь CSS 2.1.

CSS-спрайты

CSS-спрайт — способ объединить много изображений в одно, чтобы:

1. Сократить количество обращений к серверу.
2. Загрузить несколько изображений сразу, включая те, которые понадобятся в будущем.
3. Если у изображений сходная палитра, то объединённое изображение будет меньше по размеру, чем совокупность исходных картинок.

Рассмотрим, как это работает, на примере дерева:

```
01 <ul>
02 <li class="open">
03   <div class="icon"></div>
04   <div class="text">Раздел 1<br>В две строки</div>
05   <ul>
06     <li class="closed">
07       <div class="icon"></div>
08       <div class="text">Раздел 1.1 в одну строку</div>
09     </li>
10     <li class="leaf">
11       <div class="icon"></div>
12       <div class="text">Страница 1.2<br> в две строки</div>
13     </li>
14   </ul>
15 </li>
16 <li class="closed">
17   <div class="icon"></div>
18   <div class="text">Раздел 2<br>В две строки</div>
19 </li>
20 </ul>
```

Раздел 1
В две строки
 Раздел 1.1 в одну строку
 Страница 1.2
 в две строки
Раздел 2
В две строки

Сейчас «плюс», «минус» и «статья» — три отдельных изображения. Объединим их в спрайт.

Шаг 1. Использовать background

Первый шаг к объединению изображений в «спрайт» — показывать их через background, а не через тег IMG.

В данном случае он уже сделан. Стил для дерева:

```

01 .icon {
02     width: 16px;
03     height: 16px;
04     float: left;
05 }
06
07 .open .icon {
08     cursor: pointer;
09     background: url(minus.gif);
10 }
11
12 .closed .icon {
13     cursor: pointer;
14     background: url(plus.gif);
15 }
16
17 .leaf .icon {
18     cursor: text;
19     background: url(article.gif);
20 }

```

Шаг 2. Объединить изображения

Составим из нескольких изображений одно `icons.gif`, расположив их, например, по вертикали.



Из ,  и  получится одна картинка: 

Шаг 3. Показать часть спрайта в «окошке»

А теперь самое забавное. Размер `DIV`'а для иконки — жёстко фиксирован:

```

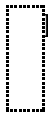
1 .icon {
2     width: 16px;
3     height: 16px;
4     float: left;
5 }

```

Это значит, что если поставить `background` 'ом объединённую картинку, то вся она не поместится, будет видна только верхняя часть:

 Пример раздела

Если бы высота иконки была больше, например, `16x48`, как в примере ниже, то было бы видно и остальное:



Пример раздела

..Но так как там всего 16px, то помещается только одно изображение.

Шаг 4. Сдвинуть спрайт

Сдвиг фона `background-position` позволяет выбирать, какую именно часть спрайта видно.

В спрайте `icons.gif` изображения объединены так, что сдвиг на 16px покажет следующую иконку:

```
01 .icon {  
02   width: 16px;  
03   height: 16px;  
04   float: left;  
05   background: url(icons.gif) no-repeat;  
06 }  
07  
08 .open .icon {  
09   background-position: 0 -16px; /* вверх на 16px */  
10   cursor: pointer;  
11 }  
12  
13 .closed .icon {  
14   background-position: 0 0px; /* по умолчанию */  
15   cursor: pointer;  
16 }  
17  
18 .leaf .icon {  
19   background-position: 0 -32px; /* вверх на 32px */  
20   cursor: text;  
21 }
```

Результат:

Раздел 1

В две строки

Раздел 1.1 в одну строку

Страница 1.2

в две строки

Раздел 2

В две строки

➡ В спрайт могут объединяться изображения разных размеров, т.е. сдвиг может быть любым.

→ Сдвигать можно и по горизонтали и по вертикали.

Отступы

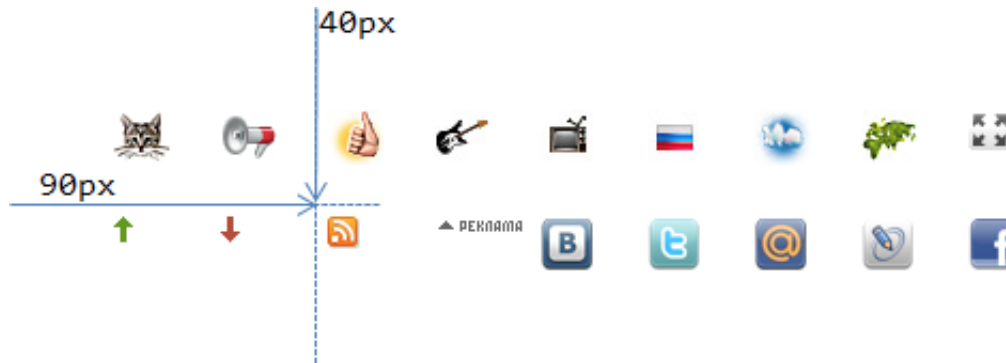
Обычно отступы делаются `margin/padding`, но иногда их бывает удобно предусмотреть в спрайте.

Тогда если элемент немного больше, чем размер изображения, то в «окошке» не появится лишнего.

Пример спрайта с отступами:



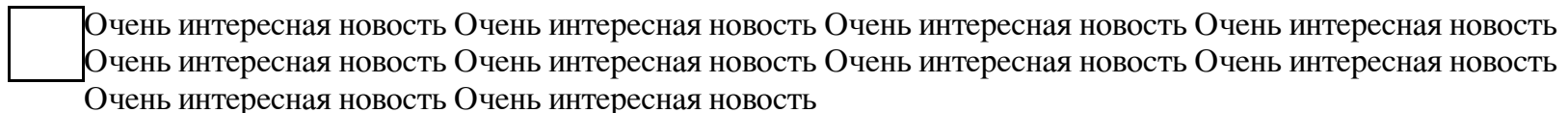
Иконка RSS находится в нём на координатах (90px, 40px):



Это значит, что чтобы показать эту иконку, нужно сместить фон:

`background-position: -90px -40px;`

При этом в левом-верхнем углу фона как раз и будет эта иконка:



Элемент, в котором находится иконка (в рамке), больше по размеру, чем картинка.

Его стиль:

```
1 .rss {  
2   width: 35px; /* ширина/высота больше чем размер иконки */  
3   height: 35px;  
4   border: 1px solid black;  
5   float: left;  
6   background-image: url(sprite.png);  
7   background-position: -90px -40px;  
8 }
```

Если бы в спрайте не было отступов, то в такое большое «окошко» наверняка влезли бы другие иконки.

Итого



Когда использовать для изображений `IMG`, а когда — `CSS background`?

Решение лучше всего принимать, исходя из принципов семантической вёрстки.

Задайте вопрос — что здесь делает изображение? Является ли оно самостоятельным элементом страницы (фотография, аватар посетителя), или же оформляет что-либо (иконка узла дерева)?

Элемент `IMG` следует использовать в первом случае, а для оформления у нас есть `CSS`.

Спрайты позволяют:

1. Сократить количество обращений к серверу.
2. Загрузить несколько изображений сразу, включая те, которые понадобятся в будущем.
3. Если у изображений сходная палитра, то объединённое изображение будет меньше по размеру, чем совокупность исходных картинок.

Если фоновое изображение нужно повторять по горизонтали или вертикали, то спрайты тоже подойдут — изображения в них нужно располагать в этом случае так, чтобы при повторении не были видны соседи, т.е., соответственно, вертикально или горизонтально, но не «решёткой».

Далее мы встретимся со спрайтами при создании интерфейсов, чтобы кнопка при наведении меняла своё изображение. Один спрайт будет содержать все состояния кнопки, а переключение внешнего вида — осуществляться при помощи сдвига `background-position`.

Для автоматизированной сборки спрайтов используются специальные инструменты, например [SmartSprites \[35\]](#).

Центрирование горизонтальное и вертикальное

В `CSS` есть всего несколько техник центрирования элементов. Если их знать, то большинство задач решаются просто.

Горизонтальное

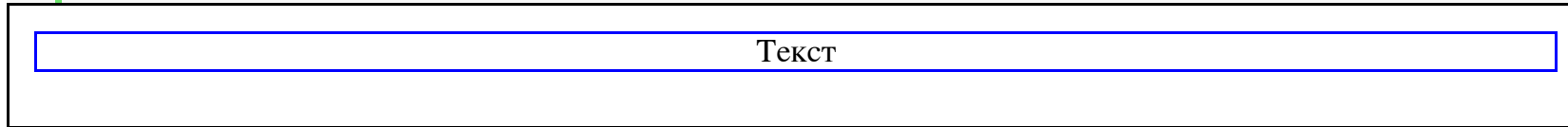
`text-align`

Для центрирования инлайновых элементов — достаточно поставить родителю `text-align: center`:

```

1 <style>
2   .outer {
3     text-align: center;
4     border: 1px solid blue;
5   }
6 </style>
7
8 <div class="outer">Текст</div>

```

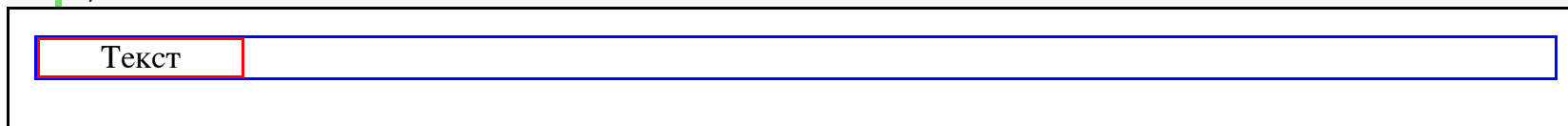


Для центрирования блока это уже не подойдёт, свойство просто не подействует. Например:

```

01 <style>
02   .outer {
03     text-align: center;
04     border: 1px solid blue;
05   }
06   .inner {
07     width: 100px;
08     border: 1px solid red;
09   }
10 </style>
11
12 <div class="outer">
13   <div class="inner">Текст</div>
14 </div>

```



Впрочем, в примере выше блок будет центрирован в IE6,7 (это отклонение от стандарта).

margin: auto

Для всех браузеров, кроме IE6,7, блок по горизонтали центрируется `margin: auto`:

```

01 <style>
02   .outer {
03     text-align: center; /* Для IE<8 */
04     border: 1px solid blue;
05   }
06   .inner {
07     width: 100px;
08     border: 1px solid red;
09     margin: auto; /* Для IE8+ и других браузеров */
10   }
11 </style>
12
13 <div class="outer">
14   <div class="inner">Текст</div>
15 </div>

```



В отличие от `width/height`, значение `auto` для `margin` само не появляется. Обычно `margin` равно конкретной величине для элемента, например `0` для `DIV`. Нужно поставить его явно.

Значение `margin-left:auto/margin-right:auto` заставляет браузер выделять под `margin` всё доступное сбоку пространство. А если и то и другое `auto`, то слева и справа будет одинаковый отступ, таким образом элемент окажется в середине. Детали вычислений описаны в разделе спецификации [Calculating widths and margins \[36\]](#).

Вертикальное

Для горизонтального центрирования всё просто. Вертикальное же изначально не было предусмотрено в спецификации CSS и по сей день вызывает ряд проблем.

Есть три основных решения.

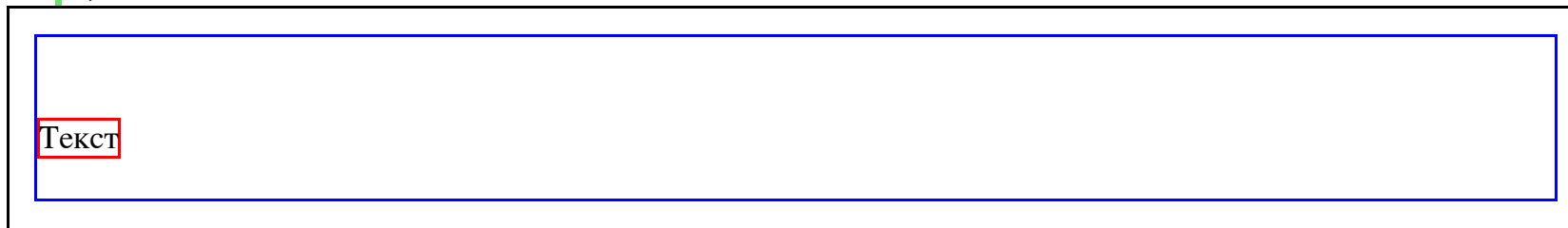
`position:absolute + margin`

Центрируемый элемент позиционируем абсолютно и опускаем до середины по вертикали при помощи `top: 50%`:

```

01 <style>
02   .outer {
03     position: relative;
04     height: 5em;
05     border: 1px solid blue;
06   }
07
08   .inner {
09     position: absolute;
10     top: 50%;
11     border: 1px solid red;
12   }
13 </style>
14
15 <div class="outer">
16   <div class="inner">Текст</div>
17 </div>

```



Это, конечно, не совсем центр. По центру находится верхняя граница. Нужно ещё приподнять элемент на половину своей высоты.

Высота центрируемого элемента должна быть известна. Родитель может иметь любую высоту.

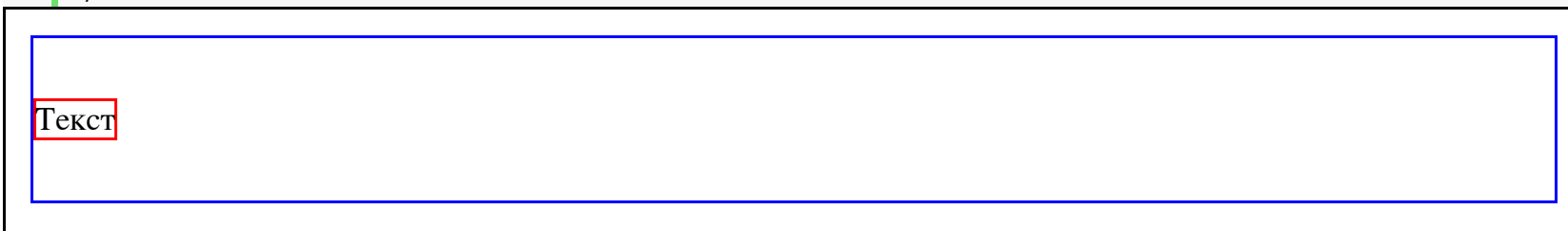
Если мы знаем, что это ровно одна строка, то её высота равна `line-height`.

Приподнимем элемент на пол-высоты при помощи `margin-top`:

```

01 <style>
02   .outer {
03     position: relative;
04     height: 5em;
05     border: 1px solid blue;
06   }
07
08   .inner {
09     position: absolute;
10     top: 50%;
11     margin-top: -0.625em;
12     border: 1px solid red;
13   }
14 </style>
15
16 <div class="outer">
17   <div class="inner">Текст</div>
18 </div>

```



При стандартных настройках браузера высота строки `line-height: 1.25`, если поделить на два $1.25\text{em} / 2 = 0.625\text{em}$.

Конечно, высота может быть и другой, главное чтобы мы её знали заранее.

Можно аналогично центрировать и по горизонтали, если известен горизонтальный размер, при помощи `left: 50%` и отрицательного `margin-left`.

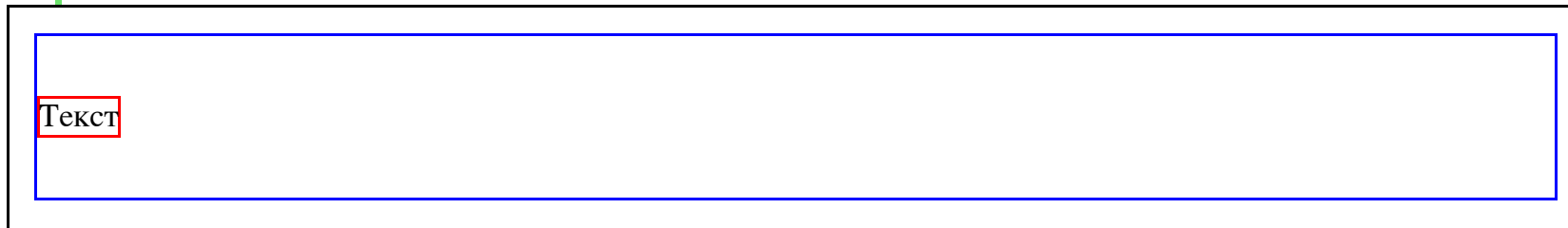
Одна строка: `line-height`

Вертикально отцентрировать одну строку в элементе с известной высотой `height` можно, указав эту высоту в свойстве `line-height`:

```

01 <style>
02   .outer {
03     height: 5em;
04     line-height: 5em;
05     border: 1px solid blue;
06   }
07 </style>
08
09 <div class="outer">
10   <span style="border:1px solid red">Текст</span>
11 </div>

```



Это работает, но лишь до тех пор, пока строка одна, а если содержимое вдруг переносится на другую строку, то начинает выглядеть довольно уродливо.

Свойство `vertical-align`

У свойства `vertical-align` [37], которое управляет вертикальным расположением элемента, есть два режима работы.

Таблица с `vertical-align`

В таблицах свойство `vertical-align` указывает расположение содержимого ячейки.

Его возможные значения:

baseline

Значение по умолчанию.

middle, top, bottom

Располагать содержимое посередине, вверху, внизу ячейки.

Например, ниже есть таблица со всеми 3-мя значениями:

```

01 <style>
02   table { border-collapse: collapse; }
03   td {
04     border: 1px solid blue;
05     height: 100px;
06   }
07 </style>
08
09 <table>
10 <tr>
11   <td style="vertical-align: top">top</td>
12   <td style="vertical-align: middle">middle</td>
13   <td style="vertical-align: bottom">bottom</td>
14 </tr>
15 </table>

```

top		
	middle	
		bottom

Обратим внимание, что в ячейке с `vertical-align: middle` содержимое находится по центру. Таким образом, можно обернуть нужный элемент в таблицу размера `width:100%;height:100%` с одной ячейкой, у которой указать `vertical-align:middle`, и он будет отцентрирован.

Но мы рассмотрим более красивый способ, который поддерживается во всех современных браузерах, и в IE8+. В них не обязательно делать таблицу, так как доступно значение `display:table-cell`. Для элемента с таким `display` используются те же алгоритмы вычисления ширины и центрирования, что и в TD. И, в том числе, работает `vertical-align`:

Пример центрирования:

```

<div style="display: table-cell; vertical-align: middle; height: 100px; border: 1px solid red">
  <button>Кнопка<br>с любой высотой<br>и шириной</button>
</div>

```

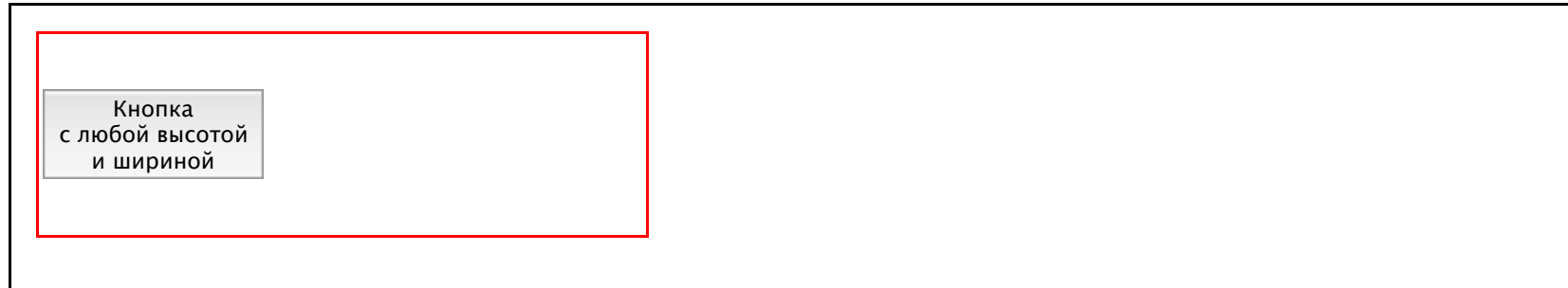


Этот способ замечателен тем, что он не требует знания высоты элементов.

Однако у него есть особенность. Вместе с `vertical-align` родительский блок получает табличный алгоритм вычисления ширины и начинает подстраиваться под содержимое. Это не всегда желательно.

Чтобы его растянуть, нужно указать width явно, например: 300px:

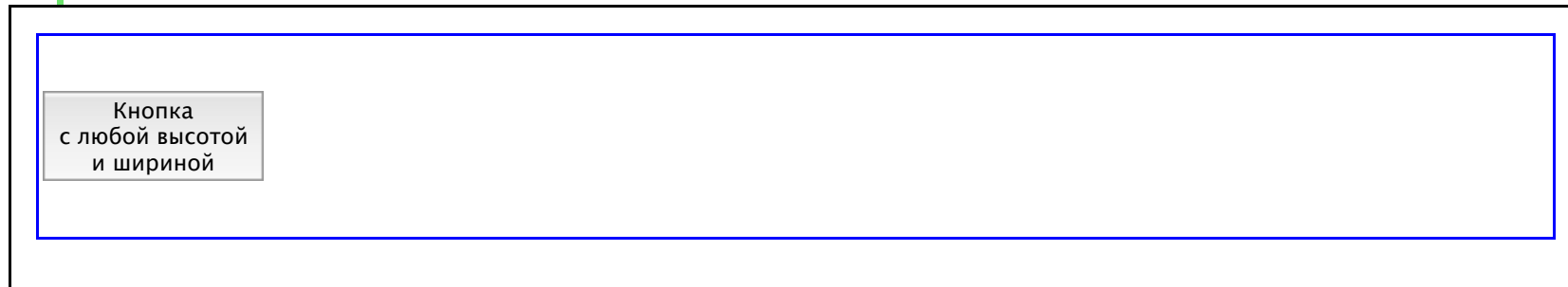
```
<div style="display: table-cell; vertical-align: middle; height: 100px; width: 300px; border: 1px solid red">  
  <button>Кнопка<br>с любой высотой<br>и шириной</button>  
</div>
```



Можно и в процентах, но в примере выше они не сработают, потому что структура таблицы «сломана» — ячейка есть, а собственно таблицы-то нет.

Это можно починить, завернув «псевдоячейку» в элемент с `display: table`, которому и поставим ширину:

```
1 <div style="display: table; width: 100%">  
2   <div style="display: table-cell; vertical-align: middle; height: 100px; border: 1px solid blue">  
3     <button>Кнопка<br>с любой высотой<br>и шириной</button>  
4   </div>  
5 </div>
```



Если дополнительно нужно горизонтальное центрирование — оно обеспечивается другими средствами, например `margin: 0 auto` для блочных элементов или `text-align: center` на родителе — для других.

Центрирование в строке с `vertical-align`

Для инлайновых элементов (`display: inline/inline-block`), включая картинки, свойство `vertical-align` центрирует *сам инлайн-элемент в окружающем его тексте*.

В этом случае набор значений несколько другой:

Картинка размером в 30px, значения vertical-align:

baseline(по умолчанию) 30
middle(по середине) 30
subвровень с <sub> 30
superвровень с <sup> 30
text-top(верхняя граница вровень с текстом) 30
text-bottom(нижняя граница вровень с текстом) 30

Это можно использовать и для центрирования, если высота родителя известна, а центрируемого элемента — нет.

Допустим, высота внешнего элемента 120px. Укажем её в свойстве line-height:

```
01 <style>
02   .outer {
03     line-height: 120px;
04   }
05   .inner {
06     display: inline-block; /* центрировать..*/
07     vertical-align: middle; /* ..по вертикали */
08     line-height: 1.25; /* переопределить высоту строки на обычную */
09     border: 1px solid red;
10   }
11 </style>
12 <div class="outer" style="height: 120px;border: 1px solid blue">
13   <span class="inner">Центрирован<br>вертикально</span>
14 </div>
```

Центрирован
вертикально

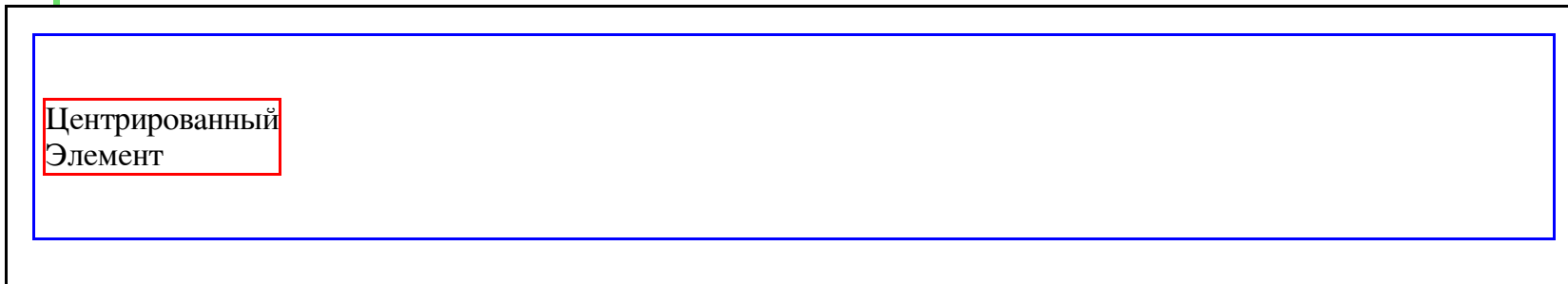
Свойство `line-height` наследуется, поэтому надо знать «правильную» высоту строки и переопределять её для `inner`.

Центрирование с `vertical-align` без таблиц

Если центрирование должно работать для любой высоты родителя и центрируемого элемента, то обычно используют таблицы(или `display:table-cell`) с `vertical-align`.

Можно сделать аналогичную вещь без таблиц, при помощи `inline-block`, `vertical-align` и вспомогательного элемента:

```
01 <style>
02 .before {
03   display: inline-block;
04   height: 100%;
05   vertical-align: middle;
06 }
07
08 .inner {
09   display: inline-block;
10   vertical-align: middle;
11 }
12 </style>
13
14 <div class="outer" style="height:100px;border:1px solid blue">
15   <span class="before"></span>
16   <span class="inner" style="border:1px solid red">
17     Центрированный<br>Элемент
18   </span>
19 </div>
```



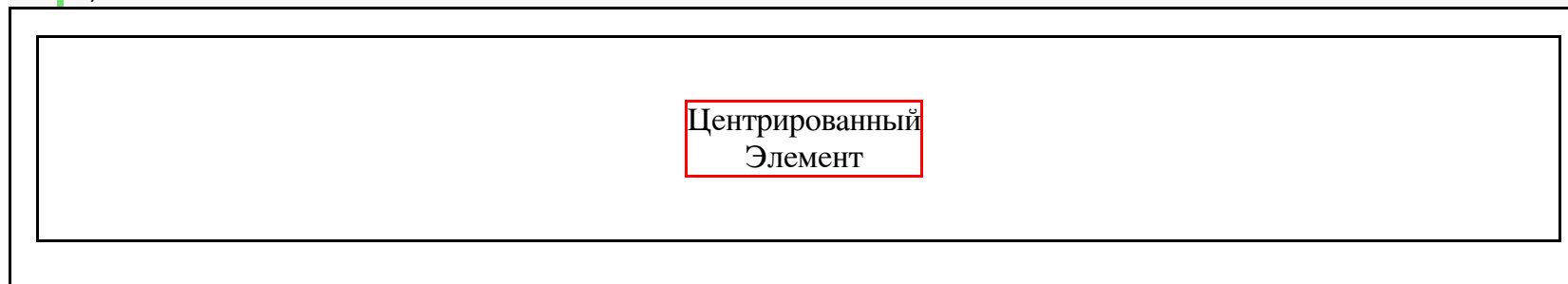
- ➡ Перед центрируемым элементом помещается вспомогательный инлайн-блок `before`, занимающий всю возможную высоту.
- ➡ Центрируемый блок выровнен по его середине.

Для всех современных браузеров и IE8 можно добавить вспомогательный элемент через `:before`:

```

01 <style>
02 .outer:before {
03   content: '';
04   display: inline-block;
05   height: 100%;
06   vertical-align: middle;
07 }
08
09 .inner {
10   display: inline-block;
11   vertical-align: middle;
12 }
13
14 /* добавим горизонтальное центрирование */
15 .outer {
16   text-align: center;
17 }
18 </style>
19
20 <div class="outer" style="height:100px; width: 100%; border:1px solid black">
21   <span class="inner" style="border:1px solid red">
22     Центрированный<br>Элемент
23   </span>
24 </div>

```



В пример выше добавлено также горизонтальное центрирование `text-align: center`. Но вы можете видеть, что на самом деле внутренний элемент не центрирован горизонтально, он немного сдвинут вправо.

Это происходит потому, что центрируется *весь текст*, а перед `inner` находится пробел, который занимает место.

Варианта два:

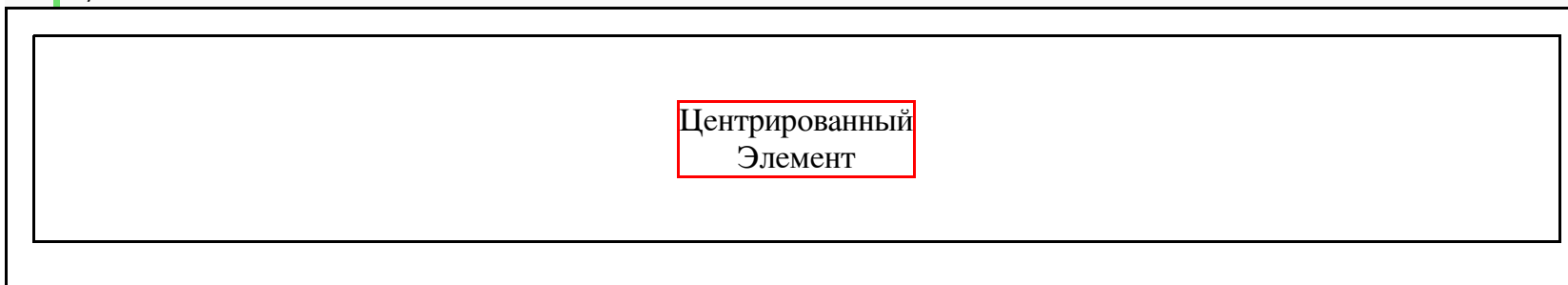
1. Убрать лишний пробел между `div` и началом `inner`, будет `<div class="outer">....`
2. Оставить пробел, но сделать отрицательный `margin-left` у `inner`, равный размеру пробела, чтобы `inner` сместился левее.

Второе решение:

```

01 <style>
02 .outer:before {
03   content: '';
04   display: inline-block;
05   height: 100%;
06   vertical-align: middle;
07 }
08
09 .inner {
10   display: inline-block;
11   vertical-align: middle;
12   margin-left: -0.35em;
13 }
14
15 .outer {
16   text-align: center;
17 }
18 </style>
19
20 <div class="outer" style="height:100px; width: 100%; border:1px solid black">
21   <span class="inner" style="border:1px solid red">
22     Центрированный<br>Элемент
23   </span>
24 </div>

```



Итого

Обобщим решения, которые обсуждались в этой статье.

Для горизонтального центрирования:

- ➡ `text-align: center` — центрирует инлайн-элементы в блоке. В IE<8 центрирует всё, но это нестандартное поведение.
- ➡ `margin: 0 auto` — центрирует блок внутри родителя. У блока должна быть указана ширина.

Для вертикального центрирования одного блока внутри другого:

Если размер центрируемого элемента известен, а родителя - нет

Родителю `position: relative`, потомку `position: absolute; top: 50% и margin-top: -<половина-высоты-потомка>`. Аналогично можно отцентрировать и по горизонтали.

Если нужно отцентрировать одну строку в блоке, высота которого известна

Поставить блоку `line-height: <высота>`. Нужны конкретные единицы высоты (px, em...). Значение `line-height: 100%` не будет работать, т.к. проценты берутся не от высоты блока, а от текущей `line-height`.

Высота родителя известна, а центрируемого элемента - нет.

Поставить `line-height` родителю во всю его высоту, а потомку поставить `display: inline-block`.

Высота обоих элементов неизвестна.

Два варианта:

1. Сделать элемент-родитель ячейкой таблицы при помощи `display:table-cell(IE8)` или реальной таблицы, и поставить ему `vertical-align:middle`. Отлично работает, но мы имеем дело с таблицей вместо обычного блока.
2. Решение с вспомогательным элементом `outer:before` и инлайн-блоками. Вполне универсально и не создаёт таблицу.

Правила форматирования CSS

Для того, чтобы CSS легко читался, полезно соблюдать пять правил форматирования.

Каждое свойство — на отдельной строке

Так — неверно:

```
#snapshot-box h2 { padding: 0 0 6px 0; font-weight: bold; position: absolute; left: 0; top: 0; }
```

Так — правильно:

```
1 #snapshot-box h2 {  
2   position: absolute;  
3   left: 0;  
4   top: 0;  
5   padding: 0 0 6px 0;  
6   font-weight: bold;  
7 }
```

Цель — лучшая читаемость, проще найти и поправить свойство.

Каждый селектор — на отдельной строке

Неправильно:

```
1 #snapshot-box h2, #profile-box h2, #order-box h2 {  
2   padding: 0 0 6px 0;  
3   font-weight: bold;  
4 }
```

Правильно:

```
1 #snapshot-box h2,  
2 #profile-box h2,  
3 #order-box h2 {  
4   padding: 0 0 6px 0;  
5   font-weight: bold;  
6 }
```

Цель — лучшая читаемость, проще найти селектор.

Свойства, сильнее влияющие на документ, идут первыми

Рекомендуется располагать свойства в следующем порядке:

1. Сначала положение элемента относительно других:
position, left/right/top/bottom, float, clear, z-index.
2. Затем размеры и отступы:
width, height, margin, padding...
3. Рамка border, она частично относится к размерам.
4. Общее оформление содержимого:
list-style-type, overflow...
5. Цветовое и стилевое оформление:
background, color, font...

Общая логика сортировки: «от общего — к локальному и менее важному».

При таком порядке свойства, определяющие структуру документа, будут видны наиболее отчётливо, в начале, а самые незначительно влияющие (например цвет) — в конце.

Например:

```
01 #snapshot-box h2 {  
02   position: absolute; /* позиционирование */  
03   left: 0;  
04   top: 0;  
05  
06   padding: 0 0 6px 0; /* размеры и отступы */  
07  
08   color: red;          /* стилевое оформление */  
09   font-weight: bold;  
10 }
```

- **Свойство без префикса пишется последним.**

Например:

```
-webkit-box-shadow: 0 0 100px 20px #000;  
box-shadow: 0 0 100px 20px #000;
```

Это нужно, чтобы стандартная (окончательная) реализация всегда была важнее, чем временные браузерные.

Организация CSS-файлов проекта

Стили можно разделить на две основные группы:

1. **Блоки-компоненты имеют свой CSS.** Например, CSS для диалогового окна, CSS для профиля посетителя, CSS для меню.

Такой CSS идёт «в комплекте» с модулем, его удобно выделять в отдельные файлы и подключать через @import.

Конечно, при разработке будет много CSS-файлов, но при выкладке проекта система сборки и сжатия CSS заменит директивы @import на их содержимое, объединяя все CSS в один файл.

2. **Страничный и интегрирующий CSS.**

Этот CSS описывает общий вид страницы, расположение компонент и их дополнительную стилизацию, зависящую от места на странице и т.п.

```

1 .tab .profile { /* профиль внутри вкладки */
2   float: left;
3   width: 300px;
4   height: 200px;
5 }

```

Важные страничные блоки можно выделять особыми комментариями:

```

01 /** =====
02  * Профиль посетителя
03  * =====
04  */
05
06 .profile {
07   border: 1px solid gray;
08 }
09
10 .profile h2 {
11   color: blue;
12   font-size: 1.8em;
13 }

```

CSS-препроцессоры, такие как [SASS \[38\]](#), [LESS \[39\]](#), [Stylus \[40\]](#), [Prefix-free \[41\]](#) делает написание CSS сильно удобнее..

Выберите один из них и используйте. Единственно, они добавляют дополнительную предобработку CSS, которую нужно учесть, и желательно, на сервере.

Решения задач



Решение задачи: Масштабируемый компонент

Нужно использовать либо % либо em.

Код работающей страницы:


```



01 <!DOCTYPE HTML>
02 <html>
03 <head>
04   <meta charset="utf-8">
05 </head>
06 <body style="font-size:100%">
07
08 <script>
09   function scale(multi) {
10     var s = document.body.style;
11     s.fontSize = parseInt(s.fontSize)*multi + '%';
12   }
13 </script>
14 <button onclick="scale(1.25)">Увеличить &uarr;</button>
15 <button onclick="scale(0.8)">Уменьшить &darr;</button>
16 <hr>
17
18 <h2>Заголовок</h2>
19
20 <div style="font-size:125%">Информация большим шрифтом</div>
21 <div style="font-size:70%">Информация мелким шрифтом</div>
22
23 <p>Все элементы должны менять размер шрифта при изменении font-size у BODY.</p>
24
25 </body>
26 </html>

```

По умолчанию, BODY в браузере имеет font-size:16px. Здесь он поставлен в 100% для удобства работы функции scale.

Внутренние элементы имеют размеры шрифта в em/%, поэтому автоматически масштабируются.

А что будет, если оставить где-то размер px? Попробуйте увеличить:

 Смотреть  [48]

```

01 <body style="font-size:100%">
02
03 <script>
04   function scale(multi) {
05     var s = document.body.style;
06     s.fontSize = parseInt(s.fontSize)*multi + '%';
07   }
08 </script>
09 <button onclick="scale(1.25)">Увеличить &uarr;</button>
10 <button onclick="scale(0.8)">Уменьшить &darr;</button>
11 <hr>
12
13 <h2>Заголовок</h2>
14
15 <div style="font-size:20px">Информация большим шрифтом</div>
16 <div style="font-size:12px">Информация мелким шрифтом</div>
17
18 <p>Все элементы должны менять размер шрифта при изменении font-size у BODY.</p>
19
20 </body>

```

Элементы, размер шрифта которых задан в `px`, останутся прежними!

В этой задаче речь идёт о странице, но всё то же самое верно и для графического компонента. Элемент и его содержимое удобно масштабировать, если размеры заданы в относительных единицах.



Решение задачи: Отступ между элементами, размер одна строка

Выбор элементов

Для выбора элементов, начиная с первого, можно использовать селектор `nth-child` [49].

Его вид: `li:nth-child(n+2)`, т.к. `n` идёт от нуля, соответственно первым будет второй элемент (`n=0`), что нам и нужно.

Решение

Отступ, размером в одну строку, при `line-height: 1.5` — это `1.5em`.

Правило:

```
li:nth-child(n+2) {  
  margin-top: 1.5em;  
}
```

Полный код в песочнице: tutorial/browser/css/between-margin.html [50] .

Ещё решение

Ещё один вариант селектора: `li + li`

```
li + li {  
  margin-top: 1.5em;  
}
```





Решение задачи: Нерабочие margin?

Ошибка заключается в том, что margin при задании в процентах высчитывается *относительно ширины*. Так написано в стандарте [51] .

При этом не важно, какой отступ: левый, правый, верхний или нижний. Все они в процентах отсчитываются от ширины. Поэтому получается, что margin-top: 40px (=10% * 400px) и height: 160px (=80% * 200) полностью покрывают всю высоту.

Ситуацию можно исправить, например, явным заданием margin-top/margin-bottom в пикселях. Или меньше процентов:

 Смотреть  [55]

```
01 <style>
02   .block {
03     width: 400px;
04     height: 200px;
05
06     border: 1px solid #CCC;
07     background: #eee;
08   }
09
10   .spacer {
11     width: 50px;
12     height: 80%;
13
14     margin-top: 5%;
15     margin-bottom: 5%;
16
17     margin-left: auto;
18     margin-right: auto;
19
20     border: 1px solid black;
21     background: #FFF;
22   }
23 </style>
24
25
26 <div class="block">
27   <div class="spacer"></div>
28 </div>
```



Решение задачи: Расположить текст внутри INPUT



Подсказка

Надвиньте элемент с текстом на INPUT при помощи отрицательного margin.

Решение

Надвинем текст на INPUT при помощи отрицательного `margin-top`. Поднять следует на одну строку, т.е. на `1.25em`, можно для красоты чуть больше — `1.3em`:

Также нам понадобится обнулить «родной» margin у INPUT, чтобы не сбивал вычисления.

 Смотреть  [59]

```
01 <style>
02   input {
03     margin: 0;
04     width: 12em;
05   }
06
07   #placeholder {
08     color: red;
09     margin: -1.3em 0 0 0.2em;
10   }
11 </style>
12
13 <input type="password" id="input">
14 <div id="placeholder">Скажи пароль, друг</div>
```

Полный код решения: [tutorial/browser/css/input-password-margin.html](https://tutorialbrowsercss.com/input-password-margin.html) [60]



Решение задачи: Разница inline-block и float

Наводящие вопросы

1. Что произойдёт, если контейнеру UL поставить рамку `border` — в первом и во втором случае?
2. Что будет, если элементы LI различаются по размеру? Будут ли они корректно перенесены на новую строку в обоих случаях?
3. Как будут вести себя блоки, находящиеся под галереей?

Ответы

Разница колоссальная.

В первую очередь она в том, что `inline-block` продолжают участвовать в потоке, а `float` — нет.

Ответы на вопросы по примеру:

Что будет, если контейнеру UL поставить рамку `border`?

Контейнер не выделяет пространство под `float`. А больше там ничего нет. В результате он просто сожмётся в одну линию сверху.

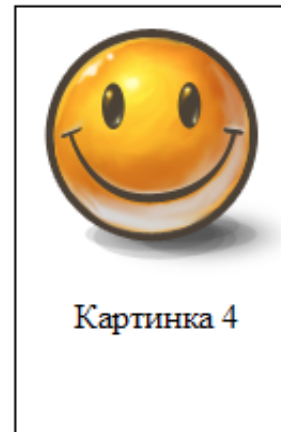
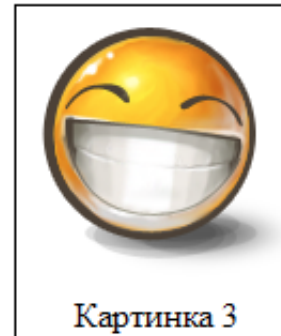
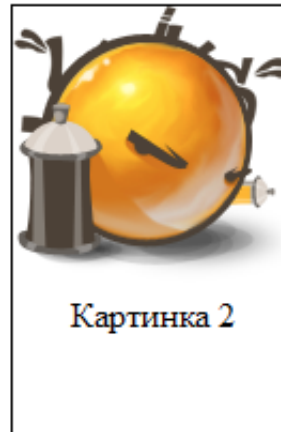
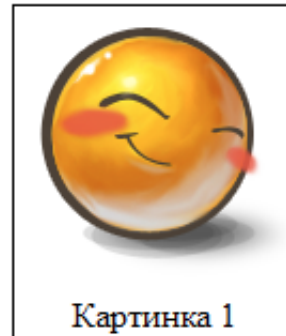
Попробуйте сами, добавьте рамку в [песочнице \[61\]](#) .

А в случае с `inline-block` всё будет хорошо, т.к. элементы остаются в потоке.

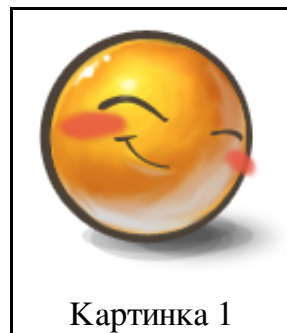
Что будет, если элементы LI различаются по размеру? Будут ли они корректно перенесены на новую строку в обоих случаях?

При `float:left` элементы двигаются направо до тех пор, пока не наткнутся на границу внешнего блока (с учётом `padding`) или на другой `float`-элемент.

Может получиться вот так (посмотрите, изменяя размер галереи [в отдельном окне \[62\]](#)):



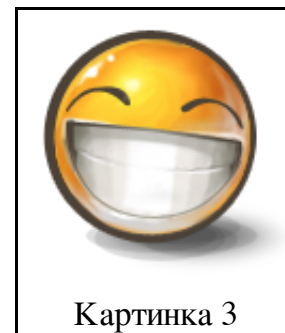
При использовании `inline-block` таких странностей не будет, блоки перенесутся корректно на новую строку. И, кроме того, можно выровнять элементы по высоте при помощи `li { vertical-align:middle }`:



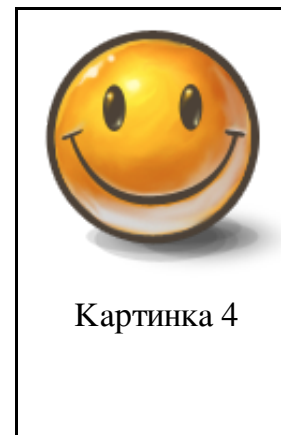
Картинка 1



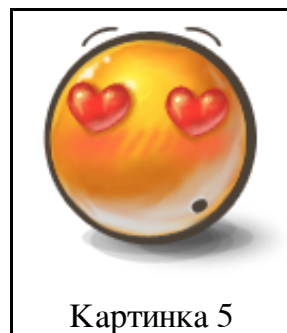
Картинка 2



Картинка 3



Картинка 4



Картинка 5



Картинка 6

[Открыть в новом окне \[63\]](#) [Открыть в песочнице \[64\]](#)

Как будут вести себя блоки, находящиеся под галереей?

В случае с `float` нужно добавить дополнительную очистку с `clear`, чтобы поведение было идентично обычному блоку.

Иначе блоки, находящиеся под галереей, вполне могут «заехать» по вертикали на территорию галереи.



Решение задачи: Дерево с многострочными узлами

Для решения можно применить принцип двухколоночной верстки `float + margin`. Иконка будет левой колонкой, а содержимое — правой: tutorial/browser/css/tree/index.html [65]



Решение задачи: Постраничная навигация (CSS)

HTML-структура:

```
1 <div class="nav">
2   
3   
4   <ul class="pages"> <li>...</li> </ul>
5 </div>
```

Стили:

```
01 .nav {
02   height: 40px;
03   width: 80%;
04   margin: auto;
05 }
06
07 .nav .left {
08   float: left;
09   cursor: pointer;
10 }
11
12 .nav .right {
13   float: right;
14   cursor: pointer;
15 }
16
17 .nav .pages {
18   list-style: none;
19   text-align: center;
20   margin: 0;
21   padding: 0;
22 }
23
24 .nav .pages li {
25   display: inline;
26   margin: 0 3px;
27   line-height: 40px;
28   cursor: pointer;
29 }
```

Основные моменты:

- Сначала идёт левая кнопка, затем правая, а лишь затем — текст.
Почему так, а не лево - центр - право?

Дело в том, что `float` смещает элемент вправо относительно обычного места. А какое обычное место будет у правого `IMG` без `float`?

Оно будет под списком, так как список — блочный элемент, а `IMG` — инлайн-элемент. При добавлении `float: right` элемент `IMG` сдвинется вправо, оставшись под списком.

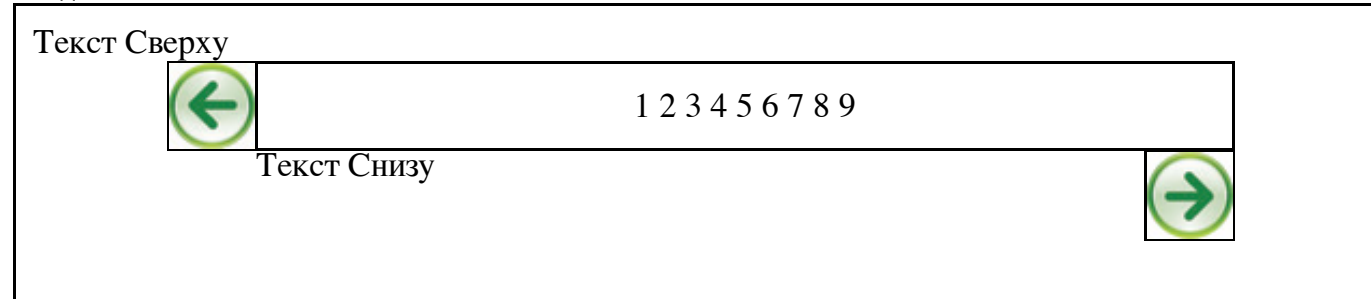
Код в порядке лево-центр-право (неправильный):

```

1 <div...>
2   
3   <ul class="pages"> (li) 1 2 3 4 5 6 7 8 9</ul>
4   
5 </div>

```

Его демо:



Правильный порядок: лево-право-центр, тогда float останется на верхней строке.

Код, который даёт правильное отображение:

```

1 <div ...>
2   
3   
4   <ul class="pages"> .. список .. </ul>
5 </div>

```

Также можно расположить стрелки при помощи `position: absolute`. Тогда, чтобы текст при уменьшении размеров окна не налез на стрелки — нужно добавить в контейнер левый и правый `padding`:

Выглядеть будет примерно так:

```

1 <div style="position:relative; padding: 0 40px;">
2   
3   <ul> (li) 1 2 3 4 5 6 7 8 9 </ul>
4   
5 </div>

```

- **Центрирование одной строки по вертикали осуществляется указанием `line-height`, равной высоте.**

Это красиво лишь для одной строки: если окно становится слишком узким, и строка вдруг разбивается на две — получается некрасиво, хотя и читаемо.

Если хочется сделать красивее для двух строк, то можно использовать другой способ центрирования.

Полное решение: tutorial/browser/css/nav-div/index.html [81] .





Решение задачи: Добавить рамку, сохранив ширину

Подсказка

Используйте свойство `box-sizing`.

Решение

Да, можно — указываем `box-sizing: border-box` и добавляем свойства:

 Смотреть  [85]

```
01 <style>
02   .left {
03     float:left;
04     width:30%;
05     background: #aef;
06   }
07
08   .right {
09     float:right;
10     width:70%;
11
12     box-sizing: border-box;
13     -moz-box-sizing: border-box;
14
15     border-left: 2px solid green;
16     padding-left: 10px;
17
18     background: tan;
19   }
20 </style>
21
22 <div class="left">
23   Левая<br>Колонка
24 </div>
25 <div class="right">
26   Правая<br>Колонка<br>...
27 </div>
```



Решение задачи: Модальное окно

Если использовать `position: absolute`, то DIV не растянется на всю высоту документа, т.к. координаты вычисляются *относительно окна*.

Можно, конечно, узнать эту высоту при помощи JavaScript, но CSS даёт более удобный способ. Будем использовать `position: fixed`:

Стиль:

```
1 #box {  
2   position: fixed;  
3   left: 0;  
4   top: 0;  
5   width: 100%;  
6   height: 100%;  
7   z-index: 999;  
8 }
```

Свойство `z-index` должно превосходить все другие элементы управления, чтобы они перекрывались.

Полный текст решения: <tutorial/browser/css/modal-div.html> [89] .



Решение задачи: Выберите элементы селектором

Решение: селекторы внизу файла <tutorial/browser/css/selectors/search.html> [90]



Решение задачи: Отступ между парами, размером со строку

Селектор

Для отступа между парами, то есть перед каждым нечётным элементом, можно использовать селектор [nth-child \[91\]](#).

Селектор будет `li:nth-child(odd)`, к нему нужно ещё добавить отсечение первого элемента:
`li:nth-child(odd):not(:first-child)`.

Можно поступить и по-другому: `li:nth-child(2n+3)` выберет все элементы для $n=0, 1, 2, \dots$, то есть 3й, 5й и далее, те же, что и предыдущий селектор. Немного менее очевидно, зато короче.

Правило

Отступ, размером в одну строку, при `line-height: 1.5` — это `1.5em`.

Поставим отступ перед каждым *нечётным* элементом, кроме первого:

```
li:nth-child(odd):not(:first-child) {  
  margin-top: 1.5em;  
}
```

Полный код в песочнице: [tutorial/browser/css/pair-margin.html \[92\]](https://tutorial/browser/css/pair-margin.html) .



Решение задачи: Поместите мяч в центр поля (CSS)

Сместим мяч в центр при помощи `left/top=50%`, а затем приподыдем его указанием `margin`:

```
1 #ball {  
2   position: absolute;  
3   left: 50%;  
4   top: 50%;  
5   margin-left: -20px;  
6   margin-top: -20px;  
7 }
```

Полный код решения: [tutorial/browser/dom/ball-css/index.html \[96\]](https://tutorial/browser/dom/ball-css/index.html)



Решение задачи: Форма + модальное окно

Структура решения

Шаги решения:

1. Для того, чтобы элементы окна не работали, их нужно перекрыть DIV 'ом с большим z-index.
2. Внутри него будет лежать «экран» с полупрозрачностью. Чтобы он растягивался, можно дать ему `position: absolute` и указать все координаты `top/left/right/bottom`. Это работает в IE8+.
3. Форму можно отцентрировать при помощи `margin` или `display: table-cell + vertical-align` на внешнем DIV.

Решение

<tutorial/browser/css/modal-login/index.html> [97] .



Решение задачи: vertical-align + table-cell + position = ?



Подсказка

Центрирование не работает из-за `position: absolute`.

Решение

Центрирование не работает потому, что `position: absolute` автоматически меняет элементу `display` на `block`.

В однострочном случае можно сделать центрирование при помощи `line-height`:

 Смотреть  [101]

```
01 <style>
02   .arrow {
03     position: absolute;
04     height: 60px;
05     border: 1px solid black;
06     font-size: 28px;
07
08     line-height: 60px;
09   }
10 </style>
11
12 <div class="arrow"><</div>
```

Если же центрировать нужно несколько строк или блок, то есть и другие [техники центрирования](#) [102], которые сработают без `display: table-cell`.

1. Reset.css <http://meyerweb.com/eric/tools/css/reset/>
2. CSS. Каскадные таблицы стилей. Подробное руководство. Эрик Мейер. <http://www.ozon.ru/context/detail/id/3881079/?partner=iliakan>
3. Большая книга CSS, Дэвид Макфарланд <http://www.ozon.ru/context/detail/id/5647176/?partner=iliakan>
4. CSS ручной работы, Дэн Седерхольм <http://www.ozon.ru/context/detail/id/5510936/?partner=iliakan>
5. Пиксель <http://ru.wikipedia.org/wiki/Пиксель>
6. Спецификации <http://www.w3.org/TR/CSS2/syndata.html#length-units>
7. CSS Values and Units 3 <http://dev.w3.org/csswg/css3-values/>
8. The Definitive Guide to Using Negative Margins <http://coding.smashingmagazine.com/2009/07/27/the-definitive-guide-to-using-negative-margins/>
9. Описанные в стандарте <http://www.w3.org/TR/CSS2/tables.html#width-layout>
10. CSS 2.1 - Tables <http://www.w3.org/TR/CSS2/tables.html>
11. Vertical-align <http://www.w3.org/TR/CSS2/visudet.html#propdef-vertical-align>
12. Vertical-align <http://www.w3.org/TR/CSS2/visudet.html#propdef-vertical-align>
13. Relationships between 'display', 'position', and 'float' <http://www.w3.org/TR/CSS2/visuren.html#dis-pos-flo>
14. CSS 2.1, 10.3.5 <http://www.w3.org/TR/CSS2/visudet.html#float-width>
15. Ссылка на HTML <http://learn.javascript.ru/files/tutorial/browser/css/winnie/index.html>
16. Открыть <http://learn.javascript.ru/play/tutorial/browser/css/winnie/index.html>
17. По алгоритму <http://learn.javascript.ru/float#float-algorithm>
18. Такого кода <http://learn.javascript.ru/play/tutorial/browser/css/winnie-clear-1/index.html>
19. Алгоритму <http://learn.javascript.ru/float#float-algorithm>
20. Получившегося кода <http://learn.javascript.ru/play/tutorial/browser/css/winnie-clear-2/index.html>
21. Получившегося кода <http://learn.javascript.ru/play/tutorial/browser/css/winnie-clear-3/index.html>
22. Получившегося кода <http://learn.javascript.ru/play/tutorial/browser/css/winnie-clear-4/index.html>
23. Открыть код <http://learn.javascript.ru/play/tutorial/browser/css/winnie-clearfill-div/index.html>
24. Открыть код <http://learn.javascript.ru/play/tutorial/browser/css/winnie-clearfill-clearfix/index.html>
25. Открыть код <http://learn.javascript.ru/play/tutorial/browser/css/winnie-clearfill-overflow/index.html>
26. Faux Columns <http://goodline.spb.ru/III-05-002.html>
27. Свойство «float» <http://learn.javascript.ru/float>
28. Описаны в стандарте <http://www.w3.org/TR/CSS2/visudet.html#abs-non-replaced-width>
29. Relationships between 'display', 'position', and 'float' <http://www.w3.org/TR/CSS2/visuren.html#dis-pos-flo>
30. "initial containing block" <http://www.w3.org/TR/CSS21/visudet.html#containing-block-details>
31. Visual Formatting Model, 9.3 и ниже <http://www.w3.org/TR/CSS2/visuren.html#positioning-scheme>
32. CSS Positioning in 10 steps <http://www.barelyfitz.com/screencast/html-training/css/positioning/>
33. Пункт 10.5 <http://www.w3.org/TR/CSS2/visudet.html#propdef-height>
34. 10.3.5 <http://www.w3.org/TR/CSS2/visudet.html#float-width>
35. SmartSprites <http://csssprites.org/>
36. Calculating widths and margins http://www.w3.org/TR/CSS21/visudet.html#Computing_widths_and_margins
37. Vertical-align <http://www.w3.org/TR/CSS2/visudet.html#propdef-vertical-align>
38. SASS <http://sass-lang.com/>
39. LESS <http://lesscss.org/>
40. Stylus <http://learnboost.github.com/stylus/>
41. Prefix-free <http://leaverou.github.com/prefixfree/>
42. Показать чистый исходник в новом окне [#viewSource](#)
43. Скрыть/показать номера строк [#noList](#)
44. Печать кода с сохранением подсветки [#printSource](#)
45. Показать чистый исходник в новом окне [#viewSource](#)
46. Скрыть/показать номера строк [#noList](#)
47. Печать кода с сохранением подсветки [#printSource](#)
- 48.
49. Nth-child <http://css-tricks.ru/Articles/Details/HowNthChildWorks>
50. Tutorial/browser/css/between-margin.html <http://learn.javascript.ru/play/tutorial/browser/css/between-margin.html>
51. В стандарте <http://www.w3.org/TR/CSS2/box.html#margin-properties>
52. Показать чистый исходник в новом окне [#viewSource](#)

53. Скрыть/показать номера строк [#noList](#)
54. Печать кода с сохранением подсветки [#printSource](#)
- 55.
56. Показать чистый исходник в новом окне [#viewSource](#)
57. Скрыть/показать номера строк [#noList](#)
58. Печать кода с сохранением подсветки [#printSource](#)
- 59.
60. Tutorial/browser/css/input-password-margin.html <http://learn.javascript.ru/play/tutorial/browser/css/input-password-margin.html>
61. Песочнице <http://learn.javascript.ru/play/tutorial/browser/css/gallery-float/index.html>
62. В отдельном окне <http://ru.lookatcode.com/files/tutorial/browser/css/gallery-float-diffsize/index.html>
63. Открыть в новом окне <http://learn.javascript.ru/files/tutorial/browser/css/gallery-inline-block/index.html>
64. Открыть в песочнице <http://learn.javascript.ru/play/tutorial/browser/css/gallery-inline-block/index.html>
65. Tutorial/browser/css/tree/index.html <http://learn.javascript.ru/play/tutorial/browser/css/tree/index.html>
66. Показать чистый исходник в новом окне [#viewSource](#)
67. Скрыть/показать номера строк [#noList](#)
68. Печать кода с сохранением подсветки [#printSource](#)
69. Показать чистый исходник в новом окне [#viewSource](#)
70. Скрыть/показать номера строк [#noList](#)
71. Печать кода с сохранением подсветки [#printSource](#)
72. Показать чистый исходник в новом окне [#viewSource](#)
73. Скрыть/показать номера строк [#noList](#)
74. Печать кода с сохранением подсветки [#printSource](#)
75. Показать чистый исходник в новом окне [#viewSource](#)
76. Скрыть/показать номера строк [#noList](#)
77. Печать кода с сохранением подсветки [#printSource](#)
78. Показать чистый исходник в новом окне [#viewSource](#)
79. Скрыть/показать номера строк [#noList](#)
80. Печать кода с сохранением подсветки [#printSource](#)
81. Tutorial/browser/css/nav-div/index.html <http://learn.javascript.ru/play/tutorial/browser/css/nav-div/index.html>
82. Показать чистый исходник в новом окне [#viewSource](#)
83. Скрыть/показать номера строк [#noList](#)
84. Печать кода с сохранением подсветки [#printSource](#)
- 85.
86. Показать чистый исходник в новом окне [#viewSource](#)
87. Скрыть/показать номера строк [#noList](#)
88. Печать кода с сохранением подсветки [#printSource](#)
89. Tutorial/browser/css/modal-div.html <http://learn.javascript.ru/play/tutorial/browser/css/modal-div.html>
90. Tutorial/browser/css/selectors/search.html <http://learn.javascript.ru/play/tutorial/browser/css/selectors/search.html>
91. Nth-child <http://css-tricks.ru/Articles/Details/HowNthChildWorks>
92. Tutorial/browser/css/pair-margin.html <http://learn.javascript.ru/play/tutorial/browser/css/pair-margin.html>
93. Показать чистый исходник в новом окне [#viewSource](#)
94. Скрыть/показать номера строк [#noList](#)
95. Печать кода с сохранением подсветки [#printSource](#)
96. Tutorial/browser/dom/ball-css/index.html <http://learn.javascript.ru/play/tutorial/browser/dom/ball-css/index.html>
97. Tutorial/browser/css/modal-login/index.html <http://learn.javascript.ru/play/tutorial/browser/css/modal-login/index.html>
98. Показать чистый исходник в новом окне [#viewSource](#)
99. Скрыть/показать номера строк [#noList](#)
100. Печать кода с сохранением подсветки [#printSource](#)
- 101.
102. Техники центрирования <http://learn.javascript.ru/css-center>