**Сетка на float**

**Пробуем управлять потоком**

О понятии [потока документа](https://htmlacademy.ru/courses/45/) рассказывается в части про позиционирование.

Поток — это порядок отображения элементов на странице. По умолчанию блочные элементы отображаются как прямоугольные области, идущие друг за другом сверху вниз, а строчные элементы располагаются сверху вниз и слева направо и при необходимости переносятся на новую строку.

Потоком можно управлять и изменять привычное поведение элементов в потоке. Например, можно заставить блочные элементы двигаться не сверху вниз, а выстраиваться в несколько колонок.

А научившись управлять потоком, вы сможете строить сетки.

В первой серии заданий мы продемонстрируем как один и тот же HTML-код можно выстроить в разные потоки. Например, вот так:



Clear:both

**Другой поток, финал**

Существует несколько способов управлять потоком и строить сетки:

1. флоаты;
2. инлайн-блоки;
3. табличная вёрстка;
4. флексбоксы.

Табличная вёрстка — самый простой для понимания способ построения сеток. Но он *считается* устаревшим и использовать его не рекомендуется. Сейчас уже можно использовать для такой вёрстки не таблицы, а элементы с display: table, display: table-row и display: table-cell. Подробно эта тема рассматривается в части [«Таблицы на CSS»](https://htmlacademy.ru/courses/86).

Флексбоксы — это новая и очень мощная технология для построения сеток. Подробно она разбирается в частях [Флексбокс. Знакомство](https://htmlacademy.ru/courses/96) и [Флексбокс. Погружение](https://htmlacademy.ru/courses/113).

В этой части мы подробно разберём флоаты и инлайн-блоки. Основной упор сделаем на флоаты, так как сейчас на многих сайтах сетки построены на них.

**Погружение во флоаты**

А теперь детально разберём свойство float, его значения, особенности поведения и способы применения.

Так как у этого свойства нет устоявшегося русского перевода или термина, то мы будем называть его «флоатом», а также использовать конструкции наподобие «зафлоатить».

Изначально float было предназначено для того, чтобы включать обтекание элементов текстом. Наподобие того, как в более старой версии HTML текст обтекал изображения c атрибутами align="left" или align="right".

То есть, можно сказать, что float — это свойство, включающее режим обтекания. Но, как часто бывает, судьба уготовила ему совсем другую роль.

Свойство float имеет следующие значения:

1. left — прижимает элемент к левому краю родителя, другие элементы обтекают его справа;
2. right — прижимает элемент к правому краю родителя, другие элементы обтекают его слева;
3. none — отключает режим обтекания и возвращает элементу нормальное поведение.

Обратите внимание, что зафлоатить элемент по центру нельзя.

Иллюстрация в задании взята из статьи [«Раскладка в CSS: float»](http://softwaremaniacs.org/blog/2005/12/01/css-layout-float/).

# float и ширина

Вы уже знаете, что по умолчанию блочные элементы растягиваются на всю доступную ширину родителя.

Если мы задаём элементу свойство float:left или float:right, то он прижимается к левому или правому краю, а также начинает ужиматься по ширине под своё содержимое. С той стороны, которая не прижата к краю родителя, появляется свободное место. Это место может быть занято другими элементами.

Зафлоаченному элементу можно явно задавать размеры и отступы.

Есть тонкость, связанная со строчными элементами. Если зафлоатить строчный элемент, то он начинает вести себя как блочный, а именно: воспринимать размеры и отступы.

**float и выпадание из потока**

Про выпадание элементов из потока при абсолютном и фиксированном позиционировании мы поговорим в части [«Позиционирование»](https://htmlacademy.ru/courses/45). А сейчас рассмотрим выпадание при флоатах.

Зафлоаченные элементы выпадают из потока, но лишь частично:

* *Блочные* элементы, которые идут в коде *после* зафлоаченного блока, перестают его замечать. Они подтягиваются вверх и занимают его место, как будто его и нет.
* *Строчные* же элементы, расположенные в коде *после* зафлоаченного блока, начинают обтекать его со свободной стороны.

Ещё раз: для блочных элементов флоатные не существуют, но текст внутри блоков флоатные обтекает.

Такое поведение флоатов даёт интересные эффекты:

1. Эффект *прохождения сквозь блоки*. Проявляется, когда зафлоаченный элемент выше, чем несколько последующих за ним блоков.
2. Эффект *выпадания из родителя* или *схлопывания родителя*. Проявляется тогда, когда все дочерние блоки в родителе зафлоачены. В этом случае родитель схлопывается по высоте, как будто в нём нет содержимого, а блоки выпадают из него.

# Флоат рядом с флоатом

Флоатные элементы становятся невидимыми для блочных элементов и видимыми для текста. А как же они взаимодействуют друг с другом?

Если вкратце то: флоатные элементы видят друг друга.

Идущие друг за другом флоаты выстраиваются в ряд, пока им хватает свободного места. Если места не хватает, то они начинают переноситься на следующую строчку. Почти как текст.

Как раз эта особенность флоатов и позволила применять их для создания сеток. Ведь в начале двухтысячных в CSS не было никакого другого способа создавать колонки и задавать им размеры.

Блочные элементы всегда располагались в разных строках. А строчные элементы, хоть и могли располагаться на одной строке, совершенно не воспринимали размеры. Позиционирование тоже не подходило для сеток, так как элементы выпадали из потока.

В общем, флоаты оказались как нельзя кстати для тех, кто решил уходить с табличной вёрстки.

# Когда флоатов много, а места мало

Следует отметить, что поведение нескольких флоатов, когда им не хватает места в одной строке, является очень странным.

Когда не влезающий флоат переносится на новую строку, возможно несколько вариантов и не все из них логичны. Например, флоат может «зацепиться» за один из предшествующих флоатов и встать ниже не в самом начале строки, а за предшествующим.

В этом упражнении примеры специально подобраны так, чтобы вы увидели странности поведения флоатов при недостатке места.

Естественно, эти особенности повлияли на приёмы использования флоатов в создании раскладок (ещё одно название для сеток). Но об этом позже.

**Свойство clear**

Свойство clear запрещает обтекание элемента другими элементами. Вот его значения:

1. left — запрещено обтекание слева;
2. right — запрещено обтекание справа;
3. both — запрещено обтекание с обеих сторон;
4. none — обтекание разрешено.

Если после флоатного элемента расположен элемент с запрещённым обтеканием, то последний опускается под флоатный.

Свойство clear учит блочные элементы «видеть» зафлоаченные.

**Борьба с выпаданием флоатов: распорки**

Эффект выпадания флоатов из родителя был большой проблемой при построении сеток. Ведь сетки на флоатах обычно делаются вот так:

1. Создаём блок-контейнер для колонок.
2. В контейнер добавляем флоатные блоки-колонки.
3. Рассчитываем ширины колонок так, чтобы им хватало места в родителе.

Такой подход работает неплохо. Но проблемы начинаются тогда, когда у контейнера есть фон, отличный от фона страницы. Когда колонки выпадали, родитель схлопывался и фон пропадал.

Необходимо было найти способ борьбы с выпаданием. Тут и пригодилось свойство clear:both. Стали применять такую структуру:

<div class="container"> - блок-контейнер

<div class="column1">...</div> - колонка, флоат

<div class="column2">...</div> - колонка, флоат

<div class="clearfix"></div> - распорка с clear:both

</div>

То есть начали добавлять после зафлоаченных колонок пустой элемент-распорку со свойством clear:both. Этот элемент видел колонки, не давал им пройти через себя, а заодно и растягивал родительский блок по высоте.

Для таких распорок прижилось специальное название класса — clearfix.

# Борьба с выпаданием флоатов: псевдораспорки

Хорошие верстальщики не очень любят, когда в вёрстке появляются ненужные дополнительные элементы и стараются от них избавляться.

До поры до времени с блоками-распорками приходилось мириться, так как без них было никак не обойтись. Но время шло и в браузерах появилась поддержка так называемых псевдоэлементов.

Псевдоэлементы позволяют с помощью CSS вставить в структуру HTML-документа узлы, которых нет в HTML коде. То есть можно вставить в код элемент и не писать его в HTML. Благодаря псевдоэлементам появилось решение проблемы флоатов без использования дополнительного элемента.

Назовём это решение псевдораспорками. Есть несколько его вариаций, но вот одна из распространённых. В CSS добавляется следующее правило:

.clearfix::after {

content: "";

display: table;

clear: both;

}

А затем класс clearfix добавляется к контейнеру, внутри которого лежат флоатные колонки. После этого в контейнер не нужно добавлять дополнительный элемент-распорку, так как распорка создаётся с помощью псевдоэлемента.

**Погружение в inline-block**

C блочно-строчными элементами вы [уже знакомились](https://htmlacademy.ru/courses/44/run/18) в части про блочную модель документа. Напомним, что это элементы, у которых свойство display имеет значение inline-block.

Блочно-строчные элементы ведут себя двояко. Снаружи они выглядят как обычные строчные, но внутри они ведут себя как блочные.

От строчных им достались следующие черты:

* по ширине они ужимаются под своё содержимое;
* могут располагаться в одну строку;
* реагируют на вертикальное выравнивание, vertical-align;
* реагируют на горизонтальное выравнивание, text-align, заданное у родителя.

От блочных:

* им можно задавать размеры с помощью width и height;
* а также внешние и внутренние отступы и рамки, которые работают во всех направлениях и увеличивают размер элемента.

# float vs inline-block

Блочно-строчные элементы могут выстраиваться друг за другом, а если не хватает места, то они переносятся на следующую строку.

Поведение похоже на флоатные элементы, однако, есть существенное отличие.

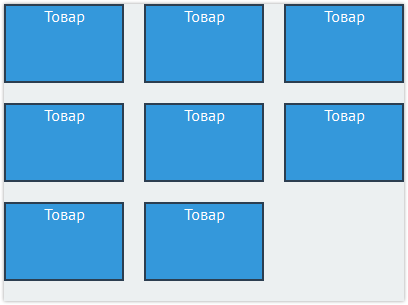
При переносе на следующую строку блочно-строчные ведут себя намного логичней. Высота строк у них рассчитывается по максимальному элементу с учётом отступов.

# Простая сетка на inline-block

Блочно-строчные элементы очень часто используют для создания декоративных элементов: кнопок, плашек, блочков. Также благодаря их умному поведению с их помощью создают различные списки товаров в каталогах.

Для создания сеток страниц эти элементы используются реже. Первая причина заключается в том, что их не поддерживают старые браузеры, например, IE7 и младше. А сетка страницы — слишком критичная вещь, чтобы позволять ей ломаться даже в старых браузерах.

Есть и ещё одна проблема, с которой мы столкнёмся в этом задании. Давайте попробуем построить сетку, как на картинке, когда у нас края товаров вплотную прилегают к краям контейнера:



Ширина контейнера 400px, ширина товара с рамками 120px, отступ между товарами 20px. Чтобы три товара влезали в строку «пиксель-в-пиксель», нужно у каждого третьего товара обнулить отступ справа. Используем для этого селектор .item:nth-child(3n).

**inline-block и пробелы в коде**

Мы рассчитали всё правильно, однако по три товара в строку не помещается.

Причина заключается в пробелах после тегов в HTML-коде. Блочно-строчные ведут себя как текст, поэтому если в коде есть пробел между элементами, то он отображается и на странице. Этот пробел увеличивает отступы между товарами, не давая им поместиться в одну строку.

Бороться с пробелом после блочно-строчных можно несколькими способами:

* удалять пробелы в коде;
* обнулять размер шрифта;
* играться с маргинами после блочно-строчного.

У каждого из способов есть свои недостатки, а подробнее эти и другие способы разбираются в [нашем переводе хорошей статьи](https://htmlacademy.ru/blog/21) (есть ещё более обширная [статья](http://css-live.ru/articles/zagadochnye-otstupy-mezhdu-inlajn-blokami.html) про блочно-строчные).

Мы попробуем последние два способа.

Способ со шрифтом заключается в том, что мы задаём нулевой размер шрифта у контейнера инлайн-блоков, а самим инлайн-блокам задаём исходный размер шрифта. Способ не работает, если вы используете относительные размеры шрифта.

Способ с маргинами заключается в том, что мы уменьшаем отступ после инлайн-блока на ширину пробела, около 4-5px. А если нам нужно, чтобы элементы стояли вплотную друг к другу, то задаём отрицательный отступ. Проблема с этим способом заключается в том, что размер пробела может быть разным в разных шрифтах и может изменяться при изменении размера шрифта.