**Лонгрид про типы раскладок**

Привет!

Вы добрались до конца первого спринта. Поздравляем!

Вёрстка — это очень большая область знаний. Всего в уроках не рассказать, но мы очень стараемся. Кроме знания конкретных тегов, свойств и значений, важно понимать основные мысли и концепции, заложенные в HTML и CSS. Поэтому мы решили делиться с вами не только практическим опытом, но и своими размышлениями о некоторых вещах. Проще всего это делать в виде лонгридов — длинных статей-рассуждений.

Материал, описанный в лонгридах, не является обязательным для дальнейшего обучения. Но он поможет вам стать круче гораздо быстрее. И, возможно, позволит избежать багов, связанных с непониманием поведения элементов на странице.

Приятного чтения!

В этом лонгриде перечислим основные способы расстановки элементов на странице. Многие из них устаревшие и в современной вёрстке не используются, но их хорошо бы знать на случай, если вы будете работать с легаси. Или вас спросят об этом на собеседовании. В конце будут современные способы раскладки и немного мыслей о будущем.

Прежде чем расставить все точки над i в теме лэйаутов, подробнее разберёмся с несколькими базовыми концепциями и понятиями. Это поможет нам говорить на одном языке. Начнём с погружения в поток.

**Поток документа**

Тему потока мы разбирали ещё в бесплатном курсе и потом закрепляли в нулевом спринте. Но для лучшего понимания разных типов раскладок можно пойти глубже в это важное понятие.

Поток документа — это то, *каким образом*, *в каком порядке* элементы выводятся на странице.

Поток документа — одна из базовых концепций вёрстки. Без хорошего понимания вы потонете в тёмных водах. Понимание потока даёт вам ускорение х10 к вёрстке. Представляя в голове поведение элементов в потоке, вы начнёте намного реже переключаться между редактором и браузером для проверки результата.

Если к HTML-разметке не применять стили, то элементы расположатся в **нормальном потоке**. Они встанут в том порядке, в котором идут в разметке, и разместятся в строку или в колонку в зависимости от дефолтных значений свойства display.

Мы условно делили элементы на блочные и строчные по тому принципу, как они выстраиваются на странице:

* Строчные встают в одну строку друг за другом, бок о бок.
* Блочные встают в колонку, друг на друга.

Элементы делятся по этому признаку не просто так. В потоке есть как раз эти два направления — строчное и блочное. По умолчанию в интерфейсах на языках, в которых пишут слева направо и сверху вниз, строчное направление идёт слева направо, а блочное — сверху вниз. Можно представить эти направления в виде осей *X* и *Y* с началом координат в левом верхнем углу.

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

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

<title>Потоки</title>

<link

rel="stylesheet"

href="https://code.s3.yandex.net/web-code/fonts/fonts.css"

/>

<link rel="stylesheet" href="./styles/style.css" />

</head>

<body>

<p>

Простое правило, которое часто нарушают: ставить рядом элементы с похожим

смыслом и окружать их пустым пространством. Пустота — мощное средство

воздействия.

</p>

<p>

Размеры элементов и расстояния между ними укладываются в гармоничную схему

— её называют сеткой. Если элемент выпадает, зритель может это заметить.

</p>

</body>

</html>

CSS:

body {

position: relative;

box-sizing: border-box;

min-width: 500px;

min-height: 100vh;

margin: 0;

padding: 55px 35px 45px 95px;

font-family: 'Open Sans', sans-serif;

font-size: 18px;

}

body::before {

content: 'Строчное направление \2192';

position: absolute;

top: 15px;

right: 0;

left: 0;

max-width: 100%;

border-bottom: 1px dashed #fd24af;

padding: 0 50px;

color: #fd24af;

font-size: 20px;

}

body::after {

content: 'Блочное направление \2192';

position: absolute;

top: 0;

bottom: 0;

left: 45px;

max-height: 100%;

border-left: 1px dashed #5f0dee;

padding: 65px 0;

color: #5f0dee;

font-size: 20px;

writing-mode: vertical-rl;

}

p {

border: 1px solid #000;

padding: 40px;

}

p + p {

margin-top: 30px;

}

Но раз есть нормальный поток, значит, есть и ненормальный?

Всё так. «Ненормальный» поток формируется при помощи CSS. Стили — очень мощный и гибкий инструмент, как вы уже поняли. Можно взять одну и ту же разметку и сделать с десяток совершенно разных сайтов только при помощи CSS.

Все типы построения лейаутов — это и есть способы влияния на поведение потока. В этой статье будет обобщение всех этих инструментов, но сначала посмотрим на концепцию, которая находится в смежной системе координат с мыслями о потоке — понятие контекста.

**Контексты**

В потоке существует идея контекста. Контекст создаёт зоны особого поведения элементов, отличного от нормального потока. Эта идея появилась не сразу. Сперва она возникла в позиционировании, потом засияла во флексах и гридах.

Позиционирование, флексы и гриды создают внутри себя контекст поведения дочерних элементов. Таким образом, мы управляем через родителя детьми. Это крайне удобный формат, чтобы мыслить большими блоками и поведением внутри них, а не отдельными сущностями.

Кроме контекста поведения, создаётся собственный контекст наложения элементов друг на друга. По умолчанию элементы накладываются друг на друга по порядку — кто в разметке ниже, тот выше в *стопке* элементов, ближе к глазам пользователя. Если верхний элемент будет больше нижних, то может закрывать их собой.

Внутри элементов с собственным контекстом вы можете управлять этой самой стопкой элементов совершенно отдельно от всех остальных элементов на странице, меняя z-index. Часто это бывает удобным для переиспользования элементов на разных страницах сайта или даже разных сайтах.

**Виды раскладок**

На заре своего появления интернет был очень простым. Сайты состояли из текста и картинок. Постепенно, по мере распространения, задачи верстальщиков усложнялись. Чтобы справляться с ними, пришлось адаптировать технологии, придумывать подходящие инструменты. Перечислим способы создания раскладок по мере эволюции этой идеи. Да, сначала приходилось страдать, но теперь всё не так плохо!

**Таблицы**

Этот способ расположения элементов на странице был очень популярным лет двадцать назад. По факту тогда он был единственным, который позволял сделать шапку, две колонки контента и подвал. Или что-то в целом сложнее, чем кусок текста с заголовками и картинками. Сотни тысяч сайтов были свёрстаны на таблицах. Кроме колонок, таблицы давали возможность выравнивать текст в каждой из своих ячеек по горизонтали и по вертикали — неслыханные чудеса для того времени!

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

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

<title>Таблица</title>

<link

rel="stylesheet"

href="https://code.s3.yandex.net/web-code/fonts/fonts.css"

/>

<link rel="stylesheet" href="./styles/style.css" />

</head>

<body>

<table border="1" width="100%" align="center" cellpadding="20">

<tr>

<th colspan="2">

<h2>Привет из прошлого!</h2>

</th>

</tr>

<tr>

<td valign="top" width="20%">

<center>

<a href="#0">Домой</a><br />

<a href="#0">О нас</a><br />

<a href="#0">Контакты</a>

</center>

</td>

<td valign="center">

<p>Так выглядели сайты раньше</p>

<p>

Как-то даже непривычно. А где баннеры, уведомления о куках, просьбы

подписаться на рассылку и прочие окошки, выпрыгивающие на тебя из-за

угла?

</p>

</td>

</tr>

<tr>

<td colspan="2">&copy; Хоть значок копирайта никуда не делся!</td>

</tr>

</table>

</body>

</html>

CSS:

body {

min-width: 500px;

margin: 0;

padding: 45px;

font-family: 'Open Sans', sans-serif;

}

table {

border-spacing: 10px;

}

p:first-child {

margin-top: 0;

}

p:last-child {

margin-bottom: 0;

}

К сожалению, проблем с таблицами тоже было немало:

* Они плохо подстраиваются под разные экраны.
* Нужно всё время следить за большим количеством тегов, ошибки приводят к поломке таблицы.
* Сложно вкладывать таблицы друг в друга из-за большого количества кода.
* У них отдельные наборы CSS-свойств и HTML-атрибутов, которые приходится запоминать.

В современном вебе вёрстку страниц с помощью таблиц вспоминают как интересный исторический факт. Таблицы сейчас используют только для табличных данных или для решения специфичных задач типа вёрстки почтовых рассылок.

**Float**

В своё время поставить блоки рядом друг с другом можно было только с использованием свойства [float](https://doka.guide/css/float/). Это было очень больно, требовало особой сноровки, но позволяло получить желаемый результат. Больно это было потому, что родительский элемент переставал *чувствовать* плавающего потомка.

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="utf-8" />

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

<title>Выпадение плавающего элемента</title>

<link

rel="stylesheet"

href="https://code.s3.yandex.net/web-code/fonts/fonts.css"

/>

<link rel="stylesheet" href="./styles/style.css" />

</head>

<body>

<div class="block">

<img

class="block\_\_img"

src="./images/img.jpg"

alt="Жёлтый персонаж в недоумении."

/>

<div class="block\_\_colored">Ты</div>

<p class="block\_\_text">

Когда нужно сделать важное, но у тебя полчаса или час времени между

созвонами

</p>

</div>

</body>

</html>

CSS:

body {

min-height: 100vh;

margin: 0;

padding: 45px;

font-family: 'Open Sans', sans-serif;

background-image: url(https://code.s3.yandex.net/web-code/images/dotted-l.svg);

}

.block {

width: 500px;

margin: 0 auto;

border: 1px solid black;

padding: 10px;

background-color: white;

}

.block\_\_img {

float: right;

width: 250px;

height: 200px;

margin: 10px;

object-fit: cover;

}

.block\_\_colored {

padding: 15px;

color: white;

background-color: #fd24af;

}

.block\_\_text {

margin-bottom: 5px;

}

Чтобы вернуть чувствительность предку, приходилось использовать clearfix — пустой блок в конце родителя со свойством clear: both. Попробуйте создать такой блок и вернуться в поток.

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="utf-8" />

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

<title>Выпадение плавающего элемента</title>

<link

rel="stylesheet"

href="https://code.s3.yandex.net/web-code/fonts/fonts.css"

/>

<link rel="stylesheet" href="./styles/style.css" />

</head>

<body>

<div class="block">

<img

class="block\_\_img"

src="./images/img.jpg"

alt="Голубой персонаж с влюблёнными глазами."

/>

<div class="block\_\_colored">Ты</div>

<p class="block\_\_text">Когда у тебя нет созвонов с утра</p>

</div>

<div class="clearfix">

</div> </body>

</html>

CSS:

body {

min-height: 100vh;

margin: 0;

padding: 45px;

font-family: 'Open Sans', sans-serif;

background-image: url(https://code.s3.yandex.net/web-code/images/dotted-l.svg);

}

.block {

width: 500px;

margin: 0 auto;

border: 2px solid black;

padding: 10px;

background-color: white;

}

.block\_\_img {

float: right;

width: 250px;

height: 200px;

margin: 10px;

object-fit: cover;

}

.block\_\_colored {

padding: 15px;

color: white;

background-color: #5f0dee;

}

.block\_\_text {

margin-bottom: 5px;

}

.clearfix {

clear: both;

}

Огромным облегчением стало появление псевдоэлементов, которыми стало можно заменить пустой блок в разметке.

**Управление свойством display**

Самый простой и быстрый способ изменить стандартное поведение элементов — изменить дефолтное значение свойства display на другое.

Например, ссылку <a> можно превратить из строчного элемента в блочный, задав ей display: block. Тогда она начнёт себя вести как настоящий блок: займёт всю строку, на которой стоит, а стоящие после неё элементы *упадут* на следующую строку.

Можно поступить аналогично с блочным по умолчанию заголовком <p> и сделать его строчным, задав свойство display: inline. Тогда два абзаца будут выглядеть как сплошной текст.

Ещё есть гибридное значение inline-block, которое снаружи делает элемент строчным, а внутри — блочным. Элемент с display: inline-block встаёт в строку с соседними строчными или строчно-блочными элементами, но при этом реагирует на свойства ширины и высоту.

Долгое время использование inline-block было популярным способом ставить блоки в ряд. Но это компромиссное решение хоть и было долгожданным, но имело свои неприятные особенности. Например, приходилось *сбрасывать* неконтролируемый отступ, который образуется между строчно-блочными элементами, при помощи [хака с font-size: 0](https://praktikum.yandex.ru/trainer/frontend-developer/lesson/ee3785bd-ae11-485a-a0ae-1c0dbc13b614/) у родительского элемента.

В целом изменение отображения хорошо работает до сих пор. Главный минус в том, что его можно применить только к конкретному элементу. Оно не наследуется и не позволяет удобно управлять вложенными элементами.

Но знать про этот способ обязательно нужно. В некоторых ситуациях он отлично работает.

Например, поставим рядом со ссылкой иконку прямо внутри предложения. Используем <span> с фоном и зададим ему размеры. Поскольку <span> изначально строчный элемент, то он не будет реагировать на свойства размеров. Чтобы решить эту проблему, достаточно сделать его строчно-блочным.

**Позиционирование**

Ещё один способ изменить положение элемента — использовать свойство position.

Мы довольно подробно разбирали все виды позиционирования [в одном из блоков прошедшего спринта](https://praktikum.yandex.ru/trainer/frontend-developer/lesson/edb4a856-37de-4643-97f7-6b7c097a88ea/), поэтому не будем останавливаться на них сейчас. Важно понимать, что использование позиционирования обоснованно в очень редких случаях.

Позиционирование часто ломает нормальный поток документа. Кроме того, вёрстку, построенную на позиционировании, сложно адаптировать под разные экраны. А это очень важно в современном мире миллиона разных устройств. Например, в примере выше меню наедет на контент на маленьких экранах.

**Мультиколонки**

Пожалуй, единственный тип раскладки, о котором мы не говорили в первом спринте, — мультиколонки. Это имитация газетных полос в вебе. Для создания мультниколонок используется свойство [columns](https://doka.guide/css/columns/).

В некоторых ситуациях это ровно то, что нужно по дизайну. Но стоит поэкспериментировать с этим типом раскладки и понять, что у него тоже есть свои недостатки.

С текстом мультиколонки работают очень хорошо, а вот блоки с их помощью размещать крайне неудобно. Например, так выстраиваются пять карточек, родителю которых задано правило для трёх колонок.

**Флексбоксы**

Веб изменился, когда появились флексбоксы. Наступила новая эпоха — эпоха свободных, интересных, необычных раскладок без страданий и костылей. Как следует из части названия технологии — флекс — этот тип раскладки задумывался как очень гибкий, решающий множество задач и лечащий часть болячек старого веба.

Флексы принесли революционные идеи:

* Теперь управление элементами происходило не через прописывание свойств каждому из них, а через родителя — флекс-контейнер.

Когда мы пишем display: flex и правила выравнивания, направления элементу, то его дочерние элементы начинают подчиняться этим правилам. Удобно! Можно мыслить целыми секциями, а не отдельными элементами. Плюс это даёт отказоустойчивость в случае увеличения или уменьшения количества контента или элементов, а также удобно при адаптации сайта под разные разрешения.

* Впервые в вебе направления перестали мыслиться как верх, низ, лево и право.

Внутри флексов есть основная и дополнительная оси. Они могут идти в любых направлениях. Выравнивание реализуется универсально. Неважно, как выстроились элементы.

Это невероятно удобно в разрезе мультиязычных сайтов, которые могут быть написаны справа налево или даже сверху вниз. Весь сайт начинает отображаться справа налево или сверху вниз всего за счёт одного-двух свойств. Выравнивание задаётся ключевыми словами flex-start (начало) и flex-end (конец) и не имеет значения, где эти начало и конец находятся в данный момент.

Кроме того, флексы дали инструмент для простого вертикального выравнивания. Поверьте, это было настоящей проблемой и занятным вопросом на собеседовании.

Но всё же флексы не были решением всех проблем вёрстки, хотя и стали прорывом.

**Гриды**

Гриды задумывались как панацея. При их проектировании разработчики постарались учесть все накопленные ошибки, баги и выводы и *вылечить* их одной новой технологией.

Из-за большого количества требований гриды превратились в космолёт. Внутри них очень много всего: новая единица измерения, новые функции, новые принципы расположения элементов. Но в реальной жизни используются далеко не все возможности. Важно просто знать, на что способен инструмент, чтобы вспомнить об этом в нужный момент.

Основные инновации гридов:

* Мышление не только рядами или колонками по отдельности, но и рядами и колонками сразу. Пересечение линий даёт ячейки; можно расставить элементы страницы по этим ячейкам, как нам угодно. Похоже на таблицы на максималках! Можно даже накладывать элементы друг на друга. Получается полноценная замена позиционированию.
* Многие разработанные в рамках гридов новинки получили распространение во всём CSS. Новые свойства gap, функции minmax(), ключевые слова fit-content и fill-content, о которых мы поговорим позже на курсе. Мир стилей обогатился за счёт развития одного из способов раскладки.
* Как и во флексах, привязка к направлениям упразднена.

Итак, мы перечислили все виды существующих раскладок, устаревшие и современные. В коде современных сайтов вы, скорее всего, встретите флексбоксы и гриды — они гораздо удобнее всех прочих способов. Остальные способы раскладки нужны для очень специфичных случаев. Или вы можете встретить их в легаси.

**Что дальше?**

Общая проблема всех раскладок — как только появляется новый способ расположения элементов, повышаются требования к сложности сайтов. В результате технологии всё время догоняют тренды дизайна. Это замкнутый круг постоянного развития и прогресса.

Работа над новыми спецификациями тоже не останавливается. У разработчиков стандартов и браузеров есть несколько наработок с новыми видами раскладок. Например, есть мысль позволить создавать раскладки при помощи функции рандомизация — случайного выбора порядка элементов. Но появляется вопрос к доступности сайтов с подобными раскладками.

Ещё есть мысль высчитывать раскладку на основе количества дочерних элементов. Например, если колонок две, то одна будет больше другой, а если колонок три, то все они будут равными по ширине.

Ещё внесено предложение и ведётся разработка механизма работы ключевого слова subgrid в качестве значения для свойств [grid-template-columns](https://doka.guide/css/grid-template-columns/" \t "_blank) и [grid-template-rows](https://doka.guide/css/grid-template-rows/" \t "_blank). Что это и зачем? Если указать такое значение, то вложенные в грид-элемент потомки начнут наследовать значение gap, а также начнут подчиняться свойствам выравнивания основного грид-контейнера. Круто, правда? Управлять уже не одним, а двумя-тремя уровнями элементов при помощи одного набора свойств!

**Заключение**

Цель этого длинного рассказа — расширить ваш кругозор, а также заронить зерно мышления о странице не как об отдельных блоках, а как о целой системе раскладки, о паттерне со своими внутренними правилами. Умея видеть закономерности на этапе проектирования вёрстки, изучения макета, вы сможете сразу представлять, какими средствами его реализовать.