**ПОНЯТИЕ И СПОСОБЫ ВЕРСТКИ**

**Верстка** – создание структуры гипертекстового документа на основе HTML-разметки. Проще говоря, перевод дизайна-макета в читаемый браузером HTML.

От верстки зависит:

* Скорость загрузки сайта
* Корректность отображения ее в браузере
* Адаптивность под разные устройства

**Типы верстки**

**Табличная –** устаревший вид верстки. Страницы конструируются из набора таблиц, внутри которых располагается контент.

**+** Простота по сравнению с блочной.

**+** Хорошая кроссбраузерность.

**-** Требуется больше HTML, сложно стилизовать через CSS, требуется много атрибутов.

**-** Медленная загрузка. Таблица не отображается, пока не будет полностью загружена.

**-** Плохая индексация поисковыми системами.

**Блочная верстка –** основным применяемым элементом является тег <div>. Каждый элемент страницы помещается в свой блок <div> и наполняется содержимым средствами HTML. Все стили вынесены из html в css файлы.

**+** Меньший объем кода.

**+** Удобство изменения элементов. В основном путем правки стилей (например, перенести блок с правой стороны на левую). С таблицами это сделать сложнее.

**+** Лучше индексируется поисковыми системами.

**+** Большая гибкость, по сравнению с таблицами. Гораздо проще реализовать нестандартные расположения и оформления элементов.

**+** Возможность реализовывать более адаптивные страницы.

**+** Загрузка блоков происходит быстрее, чем таблиц. Браузер может кэшировать css.

**-** Большая сложность в освоении.

**-** Возможны проблемы с кроссбраузерностью.

**Семантическая вёрстка** — подход к разметке, который опирается не на содержание сайта, а на смысловое предназначение каждого блока и логическую структуру документа. Название тега отображает его роль на странице.

Основные семантические теги:

* <header> - вводная часть смыслового раздела или всего сайта, обычно содержит подсказки и навигацию. Чаще всего повторяется на других страницах.
* <main> - основное, не повторяющееся на других страницах, содержание страницы.
* <footer> - заключительная часть смыслового раздела или всего сайта, обычно содержит информацию об авторах, список литературы, копирайт и т.д. Чаще всего повторяется на всех страницах сайта.
* <nav> - навигационный раздел со ссылками на другие страницы или другие части страниц.
* <section> - Объект, являющийся составной частью другого объекта (часть чего-то еще). Объекты section объединены общим смыслом.
* <article> - Независимая и самодостаточная часть страницы (что-то, что само по себе).
* <aside> - опциональный контент. Не критичная доп информация.

Section и article очень похожи. Чтобы выбрать можно использовать следующие правила:

1. Будет ли содержимое иметь осмысленное значение само по себе, например, при публикации в RSS-потоке? Если да, то выбираем **article**
2. Если части содержимого объединены общим значением, то выбираем **section**
3. Наконец, если нет никакого семантического значения, то выбираем **div**

* <details> Определяет доп информацию, которую пользователь может открывать и закрывать.
* <dialog> - определяет диалоговое окно.
* <figure> - используется для группировки различных самодостаточных элементов. Обычно <figure> это рисунок, иллюстрация, диаграмма, фрагмент кода, и т.д., на который ссылаются в основном потоке документа, но может быть перенесён в другую часть документа или в приложение не нарушив основной поток.
* <figcaption> - подпись к элементу <figure>
* <mark> - маркированный/выделенный текст, актуальный в другом контексте
* <progress> - индикатор прогресса выполнения задачи.
* <summary> - видимый заголовок для элемента details.

**Фиксированная верстка**

Размеры указываются точно в пикселях и являются статичными вне зависимости от размеров экрана. При изменении размеров экрана будет появляться горизонтальная полоса прокрутки.

Оправдана лишь в том случае, если точно известно, что продукт будет применяться только на экранах с определенным разрешением.

**Резиновая верстка**

**Резиновая верстка** – когда всему макету и отдельным его частям задается не фиксированная ширина, а эластичная – в процентах. За исключением минимальной и максимальной ширины.

Проценты считаются от ширины родительского блока. Если вы поместите такой блок внутрь body, то его ширина будет пропорциональная ширине экрана.

С вложенными блоками у нас есть два варианта - сделать их фиксированной ширины, а отступы между ними сделать резиновыми, либо сделать их пропорциональными ширине блока.

**Адаптивная верстка**

**Адаптивная верстка** основана на использовании медиа-запросов для адаптации контента под различные устройства и параметры экранов. Имеется несколько конкретных разрешений, для которых описана верстка.

Отображение на различных разрешениях может выполняться с перестройкой блоков, с изменением внешнего вида элементов страницы. (например при большом разрешении экрана панель навигации будет выглядеть строкой, а на телефоне кнопкой в углу.)

**Респонсивная (отзывчивая верстка)**

Постоянно подстраивается под размер экрана. Использует медиа-запросы в сочетании с резиновой версткой.

**КРОССБРАУЗЕРНОСТЬ**

Кроссбраузерность — это одинаковое отображение и работа сайта в различных браузерах.

2 файла помогают обеспечивать кроссбраузерность

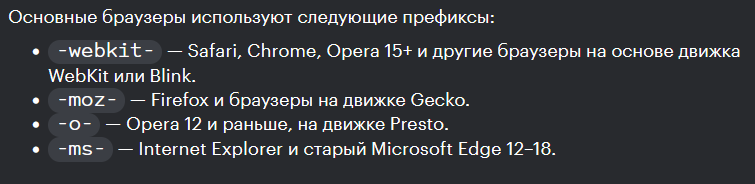
<link rel="stylesheet" type="text/css" href="css/normalize.css" /> - css-файл, который обеспечивает для HTML-элементов лучшую кроссбраузерность в стилях по умолчанию. Это современная альтернатива традиционному reset.css. Normalize.css сохраняет многие полезные стили браузера.

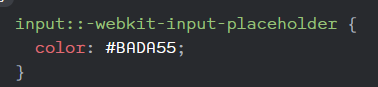
sanitize.css - это библиотека CSS, которая обеспечивает последовательную, кроссбраузерную стандартную стилизацию элементов HTML наряду с полезными значениями по умолчанию.

Она разработана вместе с normalize.css

<link rel="stylesheet" type="text/css" href="css/reset.css" /> -отменяет стили браузера

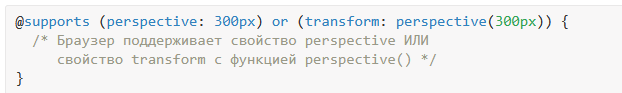
**Вендорные префексы –** приставки перед свойствами, значениями, селекторами, функциями или другими сущностями css, позволяют задавать различные стили для разных браузеров. Свойство помеченное определенным префиксом будет работать только в определенном браузере.





Чтобы не писать префиксы руками, используют специальные инструменты. **Автопрефексер** – генерирует вендорные префексы.

Директива **@supports()** позволяет проверить, поддерживает ли браузер ту или иную возможность. Допустимо создавать комплексные условия с помощью логических операторов not, and, or.



Если свойство не поддерживается, то в CSS есть естественный механизм фолбэков, благодаря каскадности. Если свойство не поддерживается, будут применены стили, объявленные выше.

**Условные комментарии** – код, позволяющий задать правила отображения только для IE. Прочие браузеры принимают эту запись за обычный комментарий.



Могут использоваться следующие операторы

|  |  |  |
| --- | --- | --- |
| **Оператор** | **Как использовать** | **Пояснения** |
| IE | [if IE] | Сейчас поддерживается только такая запись, которая расшифровывается как Internet Explorer |
| value | [if IE 7] | Отвечает за указания версии программы, считывается, если совпадает с существующими версиями |
| ! | [if !IE] | Передает значение логического отрицания. Это значит, что будет установлено противоположное значение аргумента |
| lt | [if lt IE 5.5] | less-than меньше чем. Отрабатывает, когда первый аргумент меньше второго |
| lte | [if lte IE 6] | less-than or equal меньше или равно. Реагирует на ту ситуацию, когда второе значение больше первого или равно ему |
| gt | [if gt IE 5] | greater-than больше чем. Отрабатывает, когда первый аргумент больше второго |
| gte | [if gte IE 7] | greater-than or equal больше или равно. Реагирует на ту ситуацию, когда второй аргумент меньше или равен первому |
| ( ) | [if !(IE 7)] | Объединяет определенные части кода в одно целое |
| & | [if (gt IE 5)&(lt IE 7)] | Означает логическую команду И. Приводит код в действие, когда оба условия истинны |
| | | [if (IE 6)|(IE 7)] | Означает логическую команду ИЛИ. Приводит код в действие, когда одно из указанных условий истинно |
| true | [if true] | Только правдивые значения |
| false | [if false] | Всегда противоречащие значения |

**Хаки –** это специальные css-конструкции, которые одни браузеры понимают, а другие игнорируют. Если есть возможность избежать использования хаков, то лучше их не использовать.

**VIEWPORT –** видимая пользователю область веб-страницы. Та часть, которую он может увидеть не прибегая к прокрутке. Размеры этой области определяются размером экрана.

Viewport мобильных устройств сильно отличается от компьютерных мониторов. Но даже при одинаковых размерах экрана плотность пикселей может быть разной.

Метатег viewport разработан чтобы указывать браузерам в каком масштабе необходимо отображать пользователю видимую область страницы.

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

Width=device-width отвечает чтобы ширина видимой области равнялась ширине устройства (величина независящая от разрешения экрана). Используется чтобы адаптивный дизайн сайта отображался на всех устройствах одинаково независимо от плотности пикселей.

Initial-scale устанавливает первоначальный масштаб веб-страницы (1=100%);

Нормальное масштабирование при просмотре сайта.

**Graceful degradation (**изящная деградация**) –** подход к разработке сайтов, при котором некоторые свойства или элементы дизайна заменяются на аналогичные в старых версиях браузеров, которые не поддерживают современные возможности. Также это может быть возможность работы приложения при отключенном JavaScript, либо корректное ото

Дизайн изначально предназначен для просмотра на современных браузерах. В старых версиях браузеров главное сохранить функционал приложения, несущественные и неподдерживаемые браузером особенности можно урезать.

Graceful degradation может выражаться в возможности работы с отключенным JavaScript, в акуратном отображении сайта в браузере без поддержки спецификации CSS3, в адекватном отображении сайта с отключенными картинками. Все эти отказы не должны влиять на работу веб-приложения. Однако, если все работает, то пользователю гораздо удобнее пользоваться сайтом.

Противоположностью является принцип **progressive enchancement**. В этом случае мы создаем рабочую версию для старых браузеров, а затем можем добавлять некоторые особенности для более новых. Здесь можно выделить 4 этапа:

* Создание страниц на чистом html
* Добавление правил css
* Применение css3
* Создание скриптов js

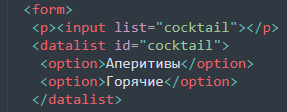
Проверить кроссбраузерность сайта можно с помощью специальных инструментов (Browsershots, Browser Sandbox, IE Testers)

**ТИПЫ INPUT**

Тег <input> создает поле для ввода данных. Используется внутри тега <form>.

Атрибуты:

* **accept –** фильтр на типы загружаемых файлов, устанавливается для поля загрузки файлов.
* **accesskey** – устанавливает сочетание клавиш для быстрого доступа.
* **alt** - устанавливает альтернативный текст для поля с изображением (пока браузер не загрузит изображение).
* **autocomplete –** включить или выключить автозаполнение (по умолчанию on).
* **autofocus –** устанавливает фокус в поле формы (курсор сразу в поле).
* **border –** толщина рамки вокруг изображения.
* **checked –** предварительно активированный переключатель или флажок.
* **form –** связывает поле с формой по её идентификатору. Используется, когда поле располагается за пределами тега <form>.
* **formaction –** определяет адрес обработчика формы (для определенного инпута).
* **list –** указывает на список вариантов ввода, созданный с помощью тега <datalist>. Изначально этот список скрыт и становится доступным при получении полем фокуса.



* **max –** устанавливает верхнее значение для ввода числа или даты в поле формы.
* **min –** нижнее значение для ввода числа или даты.
* **maxlength –** максимальное число символов, которое может быть введено пользователем в текстовом поле.
* **multiple –** позволяет указать одновременно несколько файлов в поле для загрузки файлов, а также несколько email через запятую.
* **name –** определяет имя элемента формы. Используется для получения данных в обработчике формы, или для доступа к введенным данным через скрипты.
* **pattern –** регулярное выражение, которому должен соответствовать ввод.
* **placeholder –** подсказывающий текст внутри поля формы, исчезающий при получении фокуса (либо при начале ввода).
* **disabled –** блокирует доступ и изменение поля формы. Отображается серым. Состояние поля можно менять с помощью скриптов. Значение не передается на сервер при отправке формы.
* **readonly –** введенное значение не может изменяться пользователем. Применимо не ко всем видам полей.
* **required –** делает поле обязательным для заполнения.
* **size –** ширина текстового поля, определяется числом символов моноширинного шрифта (по умолчанию 20).
* **src –** адрес на изображение, которое отображается в поле type=”image”.
* **step –** шаг приращения для числовых полей (по умолчанию 1).
* **tabindex –** определяет порядок перехода между элементами при нажатии на клавишу tab.
* **type –** сообщает браузеру к какому типу относится элемент формы.
* **value –** определяет значение элемента формы.

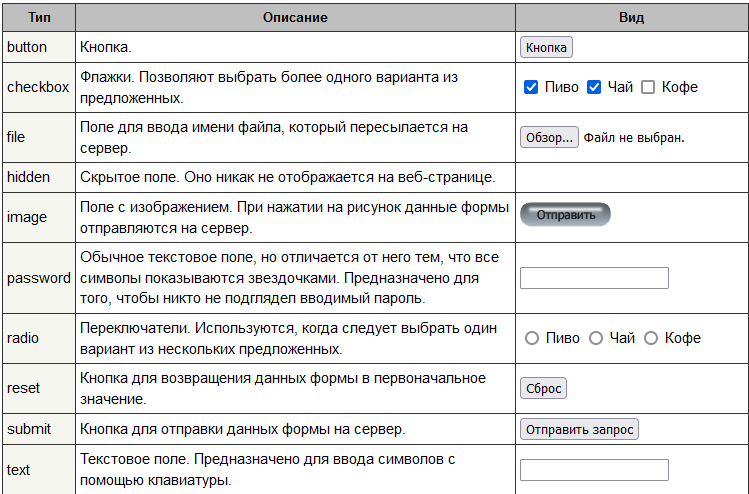
Для кнопок (input type="button | reset | submit") устанавливает текстовую надпись на них;

Для текстовых полей устанавливает предварительно введенную строку.

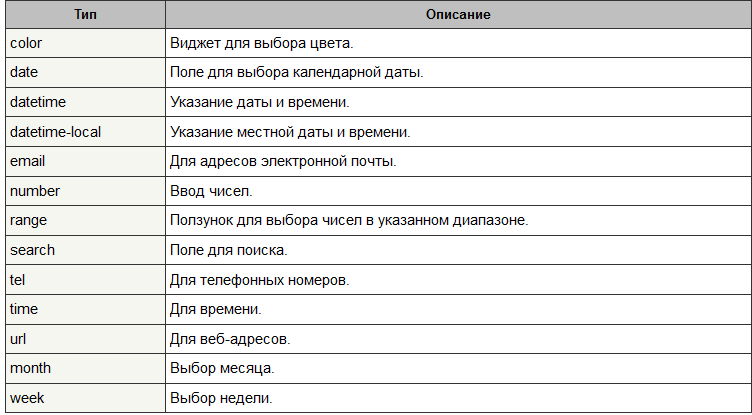
Для checkbox и radio уникально определяет каждый элемент, чтобы сервер мог однозначно установить, какой пункт выбрал пользователь.

Для файлового поля не оказывает влияния.

**Значения для атрибута type**



В HTML5 добавлены новые значения



Если в чекбоксе и радиокнопке не указать атрибут value, то при отправке формы значение будет установлено on.

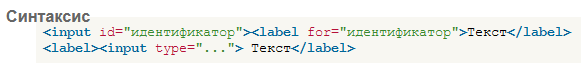
Браузер при отправке данных с формы не разрешает конфликт имен. Сервер получает строку, с параметрами в виде ключ=значение. Если в форме есть поля с одинаковыми именами, браузер отправит на сервер оба этих поля. Какой из них выбрать, или взять сразу оба решает сервер. Часто приоритетным является последнее полученное значение.

Тег **<select>** позволяет создать элемент интерфейса в виде раскрывающегося списка, а также список с одним или множественным выбором. Атрибут multiple позволяет одновременно выбрать несколько элементов. size – количество отображаемых строк списка. Внутри вложены теги <option>, представляющие пункты списка.

Тег **<optgroup>** представляет собой контейнер для тегов <option> и используется для группировки вариантов в выпадающем списке.

Поле **<textarea>** представляет область для ввода нескольких строк текста. Допустимо делать переносы строк. Атрибуты cols – ширина поля в символах, rows – высота поля в строках. Атрибут wrap указывает как переносится текст: soft – текст переносится на следующую строку, на сервер автоматический перенос не отправляется (только явный при нажатии Enter); hard – на сервер отправляется и автоматический перенос; off – текст не переносится автоматически, появляется полоса прокрутки.

Тег **<label>** - метка (подпись) элемента формы. Чаще всего используется с checkbox и radio. При нажатии на текст, будет изменяться значения чекбокса или радиокнопки.



Элемент **<fieldset>** предназначен для группировки элементов формы. Браузеры отображают его в виде рамки вокруг инпутов. Атрибут disabled блокирует поля в группе.

Тег <legend> определяет заголовок для группы <fieldset>.

Тег **<output>** HTML используется для вывода результатов вычисления, полученных, как правило, при помощи скрипта.

**ФЛЕКСБОКСЫ**

**Флексбоксы** (Flexbox, от flexible box — гибкие блоки) дают веб-разработчикам контроль над расположением элементов, их выравниванием и размерами внутри контейнера. Это позволяет выравнивать элементы по вертикали и горизонтали; менять порядок их появления; задавать направление, в котором выкладываются все элементы, и многое другое.

Чтобы объявить внешний элемент флекс-контейнером мы объявляем свойство *display: flex*. Теперь все дочерние элементы этого флекс-контейнера автоматически становятся флекс-элементами.

Флекс элементы увеличиваются и уменьшаются в зависимости от содержимого.

Flex-контейнер имеет 2 оси: **главная** (направление, в соответствии с которым располагаются все дочерние элементы контейнера) и **поперечная** (перпендикулярна главной. Вдоль этой оси работают вертикальные выравнивания).

Свойства флекс-элементов:

**flex-grow** – коэффициент роста флекс-элемента. Определяет насколько элемент будет вырастать относительно других флекс-элементов (после распределения свободного пространства). По умолчанию 0.

**flex**-**direction** – задает главную ось, то есть направление расположения элементов в контейнере.

* row – в строку слева направо;
* row-reverse – в строку слева направо;
* column – главная ось располагается вертикально сверху вниз;
* column-reverse – вертикально снизу вверх;

**justify-content** определяет то, как будут выровнены элементы вдоль главной оси.

* flex-start – у начала главной оси;
* flex-end – в конце главной оси;
* center – по центру главной оси;
* space-between – расстояния между соседними элементами одинаковые. Между элементами и краями флекс-контейнера отступов нет;
* space-around – расстояния между соседними элементами одинаковые. Между элементами и краями флекс-контейнера есть отступ, равный половине расстояния между соседними элементами;
* Space-evenly – расстояния между соседними элементами и краями флекс-контейнера одинаковые;

**align-items** – определяет то, как будут выровнены элементы вдоль поперечной оси.

* stretch – элементы растягиваются на всю “высоту” флекс контейнера;
* flex-start – элементы располагаются у начала поперечной оси;
* flex-end – элементы располагаются в конце поперечной оси;
* center – элементы располагаются по центру поперечной оси;
* baseline – элементы выравниваются по базовой линии текста внутри них.

Распределение по главной оси задается для всех элементов флекс-контенера. Задать отдельному элементу отличное поведение нельзя.

Поперечное выравнивание можно задать каждому элементу отдельно, используя свойство **align-self.**

Если флекс-элементов в контейнере станет больше, чем может уместиться в один ряд, то:

1. Они будут сжиматься до минимально возможной ширины;
2. Даже если им задать ширину, механизм флексбокса может ее уменьшить;
3. Если они перестанут помещаться в контейнер, то выйдут за его пределы, но продолжат располагаться в один ряд.

Чтобы этого не происходило, нужно воспользоваться свойством флекс-контейнера **flex-wrap**:

* nowrap (по умолчанию) – перенос-элементов на новую строку запрещен;
* wrap – перенос разрешен. Ряды элементов располагаются вдоль поперечной оси;
* wrap-reverse – ряды располагаются в обратном порядке (против направления поперечной оси)

**GRID**

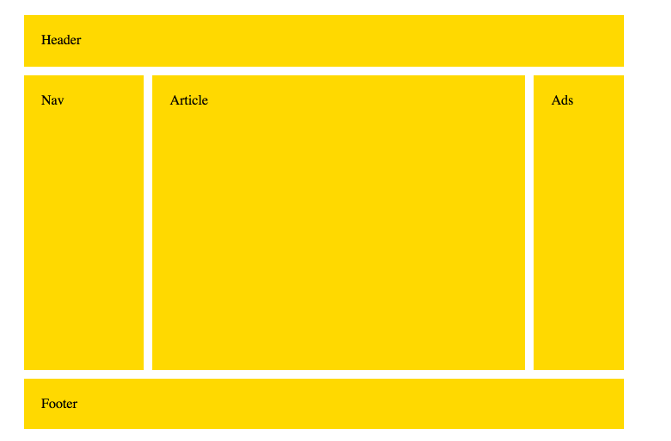
Новая модель для создания шаблонов, оптимизированная для создания двумерных макетов. Он работает по системе сеток. Grid это набор пересекающихся горизонтальных и вертикальных линий, которые создают размерность и позиционируют систему координат для контента в самом grid-контейнере.

Чтобы элемент стал grid-контейнером нужно указать ему свойство **display: grid**.

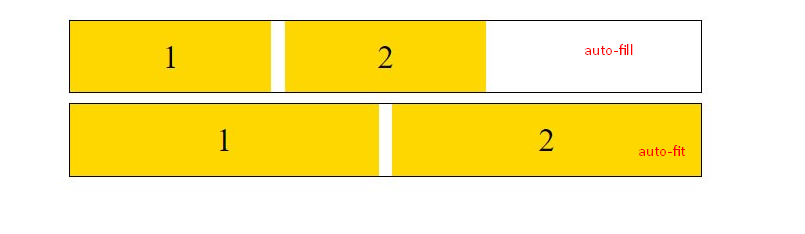
Шаблон можно описывать с помощью грид-областей, и область назначать конкретному элементу.

Например:

body {  
 display: grid;  
 grid-template-areas:   
 "header header header"  
 "nav article ads"  
 "footer footer footer";  
 grid-template-rows: 60px 1fr 60px;  
 grid-template-columns: 20% 1fr 15%;  
 grid-gap: 10px;  
 height: 100vh;  
 margin: 0;  
}  
header, footer, article, nav, div {  
 padding: 20px;  
 background: gold;  
}  
#pageHeader {  
 grid-area: header;  
}  
#pageFooter {  
 grid-area: footer;  
}  
#mainArticle {  
 grid-area: article;  
}  
#mainNav {  
 grid-area: nav;  
}  
#siteAds {  
 grid-area: ads;  
}



Для создания адаптивного грида используются свойства:

* auto-fill – заставит треки повторятся такое количество раз, пока они не заполнят контейнер  
    
  (тут колонкам выдается минимум 150px, и максимум 1fr)
* auto-fit – работает практически также, как и auto-fill. Разница только в том, что auto-fit растягивает элементы для заполнения пространства.  
  

Гриды позволяют просто описывать медиазапросы, путем переназначения grid-template-areas.

Существуют **Неявные гриды**. Они автоматически гинерируются самим грид контейнером, чтобы не сломать сетку. Например, когда мы явно объявили 2 строки и 2 колонки, а элементов, будет создан неявный грид чтобы уместить два дополнительных элемента.

Свойство grid-auto-flow определяет, как будут добавляться неявные гриды(row, column).

Грид-контейнеры можно вкладывать друг в друга. При этом вложенный не наследует свойства родителя.

Грид Элементы можно расставлять явно по сетке, указывая начальную и конечную grid-линию.

