Что такое такое tabindex?

Чем отличается тег a и button?

В чем разница между блочными, строчными и блочно-строчными элементвми?

Что такое касскадность в CSS?

Какие есть способы добавление стилей на страницу? И чем они отличаются?

За что отвечает свойство line-height? В каких единицах измерения можно узать узначения для него?

Для чего нужно указывать атрибут name в элементах формы?

В чкм отличие margin и padding?

Какие свойства CSS наследуются?

Что такое блочная модель CSS?

Как подключить скаченные шрифты?

Как сделать скругление границ?

Как задать фоновый цвет элементу?

В чем отличие em и rem?

Что такое vh и vw?

Как задать размер фоновому изображению?

В чем разница между cover и contain?

С помощью какого свойства можно задать, чтобы картинка оставалась на месте, а элемент скроллился?

Что такое псевдоклассы?

Что такое псевдоэлементы?

Какое свойство обязательно для псевдоэлментов before и after?

Как стилизовать placeholder?

Что такое BEM?

Что такое модификатор в BEM?

С помощью каких тегов создать таблицу?

Как очистить форму?

Как убрать пробелы между img?

Что такое flexbox?

Что такое специфичность в CSS?

В чем смысл таких тегов как section, header? Будет ли разница для конечного пользователя в разных тегах?

Откуда берутся базовые стили для тегов?

Что значит type submit?

Что такое позиционирование?

Что такое и зачем нужен iframe?

Как бороться с переполнением контентом?

Какие форматы изображений ты знаешь? Какие когда использовать?

Абсолютные и относительные единицы измерения?

Чем отличаются теги <div> и <span>? Когда и для чего их следует использовать?

Как работают атрибуты 'alt' и 'title' в теге <img>? Зачем они нужны и в чем разница между ними?

Что такое form в HTML? Какие элементы могут быть включены внутри тега <form>, и какие атрибуты

они могут иметь?

Какие есть способы создания анимаций в CSS? Какие свойства и функции используются для управления

анимацией?

Что за свойство outline?

Какие способы центрировть элемент вы знаете?

Что такое препроцессоры?

Что такое grid?

В чем разница между "сбросом" и "нормализацией" CSS? Что бы вы выбрали и почему?

Опишите z-индекс и то, как это работает?

Какие существуют способы визуально скрыть контент?

Что \* { box-sizing: border-box; } делает? В чем его преимущества?

В чем разница между селекторами "nth-of-type()" и "nth-child()"?

В чем разница между относительным, фиксированным, абсолютным и статически расположенным элементом?

Можете ли вы привести пример @media свойства?

Что такое такое tabindex?

Атрибут tabindex устанавливает порядок получения фокуса при переходе между элементами с помощью клавиши Tab . Переход происходит от меньшего значения к большему, например от 1 к 2,

затем к 3 и так далее. **Атрибут tabindex устанавливает порядок получения фокуса** при переходе между элементами с помощью клавиши Tab . Переход происходит от меньшего значения к большему,

например от 1 к 2, затем к 3 и так далее. При этом строгая последовательность

не важна, допускается пропускать какие-то числа и начинать с любой цифры

довательность не важна, допускается пропускать какие-то числа и начинать с любой цифры.  
  
Чем отличается тег a и button?

**Button = это кнопка. a - это текст ссылка.**

если тебе не важно нативное поведение button и a (а судя по вопросу тебе это не важно) и ты обрабатываешь события через js -то вообще пофиг что там будет. хоть div

кроме одного момента: SEO - робот обходит <a href=''>, а нажатия кнопок и js - нет

В плане внешнего вида - разницы вообще никакой.

Зависит от того, как вы будете реализовывать действие на нажатие элемента. Если уверены, что будет много джаваскрипта или же будет использован какой-нибудь фреймворк - то можно активно использовать button. Если же джаваскрипта мало или вовсе нет, то лучше через 'a'.

Чтобы таких вопросов возникало меньше, важно понимать, что каждый браузер имеет право интерпретировать каждый ваш тэг или любой html, css, js код по-своему. А функционал тэга 'a' - это соглашение в html.  
  
В чем разница между блочными, строчными и блочно-строчными элементвми?

Блочные элементы — отдельные структурные единицы, которые используются для создания структуры веб-страницы. Такие блоки имеют форму прямоугольника и ***занимают всю ширину экрана***.

Размеры блочных элементов могут быть изменены с помощью значений свойств width и height. При помощи свойства border можно задавать толщину, цвет и стиль линий границы, по умолчанию она невидима.

По умолчанию границы вплотную примыкают к контенту блоков, увеличить внутренний отступ можно с помощью значений свойства padding. Для просмотра стилей в редакторе ниже, смотрите вкладку CSS.При необходимости можно использовать свойство box-sizing, которое позволяет изменить алгоритм расчета ширины и высоты, либо учитывая padding и border при расчете, либо нет, в зависимости от значения свойства box-sizing.

Строчные элементы-Определить строчный элемент в инструментах разработчика просто, значение свойства display у таких элементов установлено как inline. Визуально же строчные элементы также отличаются. Они подобны строкам, **по умолчанию каждый новый элемент отрисовывается рядом с предыдущим**. То **есть главное отличие заключается в разных принципах расположения контента: в блочной модели — он вертикальный, а в строчной — горизонтальный.** Строчные элементы имеют характерные только им стили, например vertical-align (выравнивание по вертикале). Данное правило не работает в блочных элементах. В тоже время строчному элементу нельзя задать ширину и высоту, как это делается в блочных элементах. Тут ширина определяется шириной контента и значениями padding и border (если они есть).

**Что касается совместного использования, то внутри блочных элементов рекомендуется размещать элементы обоих типов, внутри строчных — только строчные.**

Что такое каскадность в CSS?  
  
CSS — это каскадные таблицы стилей (Cascading Style Sheets). **Каскадность — это главный принцип css — это приоритет одних правил/стилей над другими.** Когда одни стили перебивают другие. Это понятие также называется "специфичность селектора".

При вычислении приоритета браузер определяет «Вес» каждого CSS правила, который складывается из веса отдельных селекторов этого правила. Стили менее весомых правил перебиваются стилями более весомых. В результате элемент получает собранные стили от самых «весомых» правил. Все это и есть каскадность.  
  
  
Какие есть способы добавление стилей на страницу? И чем они отличаются?  
  
**Есть разные способы добавить стили на страницу. Можно написать их прямо в HTML в теге <style>. Можно подключить внешний CSS-файл. Можно задать стили с помощью JavaScript или оформить отдельный элемент при помощи инлайн-стилей**.

Писать стили сегодня принято в отдельном CSS-файле и затем подключать его с помощью тега <link>. А вот инлайн-стилей стараются избегать, хотя они тоже бывают полезными

От того, как вы подключаете стили на страницу, зависит, как быстро она отобразится, однако некоторые варианты подключения стилей могут привести к неожиданным последствиям. Например, если к элементу одновременно применить и внешние, и инлайн-стили с разными значениями CSS-свойств, то применятся инлайн-стили, так как у них более высокий приоритет. Разберёмся, какие у каждого способа достоинства и недостатки, и какой из них вам подойдёт

**Внешний файл со стилями**

При этом подходе CSS пишется в отдельном файле с расширением .css. Такие стили ещё называют связанными.

Для связывания страницы с файлом стилей используется тег <link> внутри <head>:

<head>

<link rel="stylesheet" href="main.css">

</head>

Атрибут rel="stylesheet" указывает, что этот файл является таблицей стилей. В href указывается относительный или абсолютный путь к CSS-файлу.

А вот какие стили содержатся в файле main.css:

h1 {

color: green;

font-size: 24px;

}

p {

font-size: 24px;

}

Ещё таким образом можно подключать файл стилей, который находится на другом сайте. Например, когда подключаете шрифты с Google Fonts.

Преимущества "Преимущества"

Можно использовать один CSS-файл для нескольких страниц. Изменение стилей в таком файле автоматически применится ко всем страницам, к которым он подключён.

При первой загрузке страницы файл со стилями кэшируется, и в следующие разы она открывается быстрее.

Во внешних стилях можно свободно использовать псевдоклассы и псевдоэлементы. Например, задавать интерактивные состояния отдельно выбранным кнопкам, стилизовать каждый чётный элемент списка и тому подобное.

Внешний CSS также позволяет использовать кастомные свойства.

Недостатки

Чтение стилей приостанавливает чтение кода страницы. Если файл, к которому вы обращаетесь, находится на отдалённом и недоступном сервере, это может повлиять на работоспособность остальных элементов страницы. Например на исполняемые скрипты. Подробнее можно почитать в статьте «Как браузер рисует страницы».

Встроенные стили

Вариант, когда CSS-свойства описываются в самом HTML-документе внутри тега <style> в шапке страницы:

<head>

<style>

h1 {

color: tomato;

}

p {

font-size: 24px;

}

</style>

</head>

Преимущества

Поскольку код находится непосредственно в HTML-файле, браузер не загружает сторонние файлы. Это позволяет отрисовать страницу быстрее.

Встроенные стили работают изолированно и применяются непосредственно к странице, на которой прописаны.

Можно использовать псевдоклассы и псевдоэлементы.

Недостатки

С каждым новым правилом вес HTML-файла будет увеличиваться и страница будет загружаться медленнее.

Со временем такие же стили могут понадобиться на других страницах, и CSS придётся дублировать.

Импорт CSS Скопировать ссылку "Импорт CSS"

Этот вариант тоже требует подключения внешнего файла стилей. Отличие в том, что этот файл содержит не весь CSS-код разом, а стили разбиты на отдельные файлы и подключаются с помощью директивы @import.

Посмотрим, как это делается. Даём ссылку на CSS-файл через тег <link>:

<link rel="stylesheet" href="main.css">

Затем в main.css импортируем несколько других CSS-файлов, header.css для шапки сайта, navbar.css для меню, и так далее:

@import url("header.css");

@import url("navbar.css");

@import url("footer.css");

@import url("grid.css");

Преимущества

По мере того как ваш проект растёт в размерах и сложности, поддержка CSS-файлов тоже усложняется. @import помогает разбить массивную таблицу стилей на более мелкие и понятные части.

Недостатки

Подключение стилей при помощи CSS-файла, в котором директивы @import подключают другие файлы, может значительно увеличить время отрисовки страницы. Браузер узнает о существовании других CSS-файлов слишком поздно, а если и в них окажутся импорты, то всё будет ещё хуже.

Если вам очень нужно подключить несколько файлов, лучше добавить несколько элементов <link>.

Иногда так всё же делают при разработке сайтов, чтобы не тратить время на сборку многих файлов и просто подключать их одним файлом с кучей импортов. Но перед публикацией сайта все эти импорты «склеиваются» в один файл для более эффективной загрузки.

Инлайн-стили "

В этом варианте элементам, которые необходимо оформить, добавляется атрибут style. Значением этого атрибута могут быть свойства и значения CSS, которые и применятся к элементу:

<h1 style="сolor: yellow; margin: 10px 20px">Заголовок</h1>

<p style="text-align: right; font-size: 12px">Текст</p>

Преимущества

Как и со встроенными стилями, браузеру не нужно запрашивать файл, что ускоряет рендеринг.

Не нужно писать селекторы, потому что стили применяются сразу к нужному элементу.

Удобно отлаживать код или делать быстрые правки прямо в админке, без разворачивания проекта.

Недостатки

Однако несмотря на преимущества, инлайн-стили — это антипаттерн. Недостатков у этого подхода слишком много.

**Приоритет**. У инлайн-стилей наивысший приоритет, их нельзя переопределить с помощью селекторов по идентификатору, классу или тегу во внешнем CSS. Единственный способ это сделать — добавить !important к значению свойства. Но этот приём тоже считается плохой практикой.

**Переиспользование**. При использовании инлайн-стилей мы будем вынуждены добавлять инлайн-стили к каждому элементу, который необходимо оформить. Если элементов на странице много, прописывать стили вручную станет неудобно.

Всё в одну кучу. Оформление при помощи инлайн-стилей нарушает принцип «разделения содержимого и оформления».

Псевдоклассы и псевдоэлементы. В инлайн-стилях нельзя использовать псевдоклассы и псевдоэлементы.

**Восприятие кода**. Если писать стили внутри атрибута style, то HTML становится трудно читать. Логическая структура перестаёт быть видна, стили размазываются по всему коду. И чем крупнее проект, тем сложнее управлять оформлением.  
  
За что отвечает свойство line-height? В каких единицах измерения можно юзать значения для него?  
  
**Свойство CSS line-height устанавливает величину пространства между строками**, например в тексте. **В блочных элементах оно указывает минимальную высоту блоков строк внутри элемента. В незамещаемых внутристрочных элементах —указывает высоту, которая используется для вычисления высоты блоков строк**.

Значения:

**normal** (пример)"normal;"Зависит от пользовательского браузера. Компьютерные браузеры (включая Firefox) используют значение по умолчанию 1.2в зависимости от элементов font-family.

**<число>** (пример)"2.5;"

Значением line-height будет результат умножения чисел (без имен) для размера элементов шрифта. Указанное число, по сути, множитель. В большинстве случаев это строгий способ указания значения line-height, потому что позволяет избежать непредвиденных результатов при наследовании.

**<величина>** (пример)"3em;"

Указанные <величина>использования при вычислении высоты блока строк. Значение, заданное в модулях em, может привести к непредвиденным результатам

**<процентное соотношение>** (пример)"150%;"

Относительно размера шрифта самого элемента. Относительно размера шрифта самого элемента. Вычисленное значение умножается <percentage>на вычисленный размер шрифта элемента. Процентные значения могут привести к неожиданным результатам

Для чего нужно указывать атрибут name в элементах формы?  
  
**Атрибут name (HTML тега <form>) определяет имя формы, которое используется для идентификации (задает имя для формы**). Атрибут может использоваться для ссылки на элементы в JavaScript, или ссылаться на данные формы после её отправки на сервер.

Обращаю Ваше внимание, что в XHTML атрибут name считается устаревшим. Используйте вместо него глобальный HTML атрибут id для идентификации формы.

В чем отличие margin и padding?  
  
**Падинги (свойство padding ) — это внутренние отступы от границ элемента до его содержимого. Маржины (свойство margin ) — это внешние отступы от границ элемента до соседних элементов.**

Создавать промежутки между элементами можно и тем, и другим способом, но если **padding – это отступ от содержимого до края блока, то margin – это расстояние от одного блока до другого, межблоковое пространство.**

Padding отделяет содержимое от границы блока, а margin создает промежутки между блоками

Для установки полей или отступов в CSS от каждой стороны элемента существуют следующие свойства:

Отступы: Поля:

padding-top: значение; margin-top: значение;

padding-right: значение; margin-right: значение;

padding-bottom: значение; margin-bottom: значение;

padding-left: значение; margin-left: значение;

Значения могут быть указаны в любых единицах CSS – px, em, % и т. д.

Также существует очень удобная вещь как сокращенная запись margin и padding CSS. Если вам необходимо задать поля или отступы для всех четырех сторон элемента, совсем необязательно записывать свойство для каждой стороны по отдельности. Всё делается проще: для margin и padding можно указывать сразу 1, 2, 3 или 4 значения. От количества значений зависит, как распределяются настройки:

4 значения: задаются отступы для всех сторон элемента в такой последовательности: сверху, справа, снизу, слева:

padding: 2px 4px 5px 10px;

3 значения: задается отступ сначала для верхней стороны, потом одновременно для левой и правой, а затем – для нижней:

padding: 3px 6px 9px;

2 значения: задаются отступы сначала одновременно от верхней и нижней стороны, а затем – одновременно для левой и правой:

padding: 6px 12px;

1 значение: задаются одинаковые отступы для всех сторон элемента:

padding: 3px;

Те же правила касаются и свойства margin CSS. Обратите внимание на то, что для margin можно использовать и отрицательные значения (например, -3px), которые иногда бывают весьма полезными.

**Схлопывание margin**

Представьте ситуацию: два блочных элемента находятся друг над другом и им заданы поля margin. Для верхнего блока установлено значение margin: 60px, а для нижнего – margin: 30px. Логично было бы предположить, что два граничащих поля двух элементов просто соприкоснутся и в итоге промежуток между блоками будет равен 90 пикселям.

Однако дела обстоят по-другому. На самом деле в такой ситуации проявляется **эффект, который называют схлопыванием, когда из двух примыкающих полей элементов выбирается наибольший по размеру**. В нашем примере итоговый промежуток между элементами будет равен 60 пикселям.

**Схлопывание полей margin в CSS**

Расстояние между блоками равно большему из значений

**Схлопывание margin работает только для верхних и нижних полей элементов** и не относится к полям с правой и левой сторон. Окончательная величина промежутка высчитывается в разных ситуациях по-разному:

Когда оба значения margin положительные, итоговый размер поля будет равен бóльшему значению.

Если одно из значений является отрицательным, то для вычисления размера поля нужно получить сумму значений. Например, при значениях 20px и -18px размер поля будет равен:

20 + (-18) = 20 — 18 = 2 пикселя.

Если оба значения отрицательные, сравниваются модули этих чисел и выбирается число, бóльшее по модулю (следовательно, меньшее из отрицательных чисел). Пример: необходимо сравнить значения полей -6px и -8px. Модули сравниваемых чисел равны 6 и 8 соответственно. Отсюда следует, что 6 < 8, то есть, -6 > -8. Итоговый размер поля равен -8 пикселей.

В случае, когда значения указаны в разных единицах CSS, они приводятся к одной, после чего сравниваются и выбирается бóльшее значение.

Размер margin для дочерних элементов определяется еще более интересно: если у потомка поле margin больше, чем у родителя, то приоритет отдается ему. В этом случае размеры верхнего и нижнего полей родителя станут такими, как задано у потомка. При этом расстояния между родителем и потомком не будет.

Какие свойства CSS наследуются?  
**Наследуемые свойства**

**Свойства шрифта:** font и его «производных»: font-style, font-variant, font-weight, font-stretch, font-size и font-family; color и line-height.

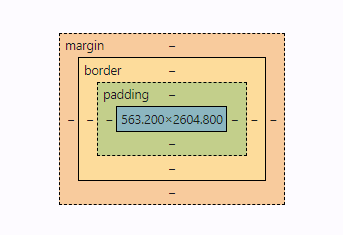
**Свойства межбуквенных и «межсловных» расстояний:** letter-spacing, word-spacing и white-space.

**Параметры текста:** text-align, text-indent, text-shadow, text-transform;

оформление пунктов списков: list-style, list-style-type, list-style-position.

**Внешний вид курсора**: cursor и отображение содержимого элемента visibility.

Замечание

Ключевое слово **inherit** позволяет нам явно задать наследование. Это работает и на наследуемых, и на ненаследуемых свойствах..  
  
  
Что такое блочная модель CSS?  
**Блочная модель — правила, по которым браузер определяет размер элемента на странице, его ширину и высоту.**Блочная модель состоит из нескольких CSS-свойств, влияющих на размеры элемента:

width — ширина элемента;

height — высота элемента;

padding — внутренние отступы от контента до краёв элемента;

border — рамка, идущая по краю элемента;

margin — внешние отступы вокруг элемента.  
  
Как подключить скаченные шрифты?  
  
Например, у нас есть шрифт Raleway.ttf и мы хотим использовать его во всех заголовках h1 нашего сайта. Для этого выполняем следующие действия:

В корневой папке сайта создаём папку fonts и копируем туда наш Raleway.ttf;

В самом низу файла стилей style.css прописываем правило:

@font-face {

// название шрифта(чтобы не запутаться лучше указать название папки со шрифтом)

font-family: "RalewayRegular";

// адрес шрифта и тип

src: url("../fonts/RalewayRegular.ttf") format("truetype");

font-style: normal;

font-weight: normal;

}

А также в файле стилей задаём правило для всех заголовков:

h1{

font-family: "RalewayRegular";

}

Теперь все заголовки 1-го уровня на сайте отображаются с нужным нам шрифтом.

Если вы хотите подключить несколько шрифтов или их начертаний, то просто добавьте их под предыдущим:

@font-face {

font-family: "RalewayRegular";

src: url("../fonts/RalewayRegular.ttf") format("truetype");

font-style: normal;

font-weight: normal;

}

@font-face {

font-family: "RalewayBold";

src: url("../fonts/RalewayBold.ttf") format("truetype");

font-style: normal;

font-weight: normal;

}

Вы могли заметить, что есть разные форматы шрифтов - .ttf, .woff, .eot, .svg и другие. Еще существует формат для современных браузеров .woff2,

Обычно каждый шрифт подключается сразу в 3-х форматах. Это сделано, чтобы шрифт правильно отображался во всех браузерах в том числе и старых. Выглядит это так:

@font-face {

font-family: "RalewayRegular";

src: url("../fonts/RalewayRegular/RalewayRegular.eot");

src: url("../fonts/RalewayRegular/RalewayRegular.eot?#iefix") format("embedded-opentype"),

url("../fonts/RalewayRegular/RalewayRegular.woff") format("woff"),

url("../fonts/RalewayRegular/RalewayRegular.ttf") format("truetype");

font-style: normal;

font-weight: normal;

}

Здесь следует обратить внимание на порядок подключения — это важно!

Не будем вдаваться в подробности, просто запомните вид этой конструкции.

Если у вас есть шрифт только в одном формате, то воспользуйтесь различными конверторами шрифтов.  
  
  
Как сделать скругление границ?  
  
  
**CSS-свойство border-radius помогает скруглить углы элемента**. Оно особенно полезно для стилизации кнопок, форм, карточек товаров и других элементов сайта.

Если укажете **одно значение, оно применится ко всем углам элемента**. Например, так из квадрата можно сделать круг

Пропишете **два значения — первое применится к левому верхнему и правому нижнему углу, а второе — к правому верхнему и левому нижнему**.

Укажете **три значения — первое применится к левому верхнему углу, второе — к правому верхнему и левому нижнему, третье — к правому нижнему**.

А если пропишете **четыре значения, первое применится к левому верхнему углу, второе — к правому верхнему, третье — к правому нижнему, четвёртое — к левому нижнему**. Просто запомните, что значения распределяются по часовой стрелке, начиная от левого верхнего угла.

Чтобы скруглить только один угол, используйте свойства:

border-top-left-radius — для верхнего левого угла;

border-top-right-radius — для верхнего правого угла;

border-bottom-left-radius — для нижнего левого угла;

border-bottom-right-radius — для нижнего правого угла.  
  
  
  
  
  
  
Как задать фоновый цвет элементу?

**При помощи свойства background-color можно задать цвет фона элемента**. Задать фоновый цвет можно любому элементу — строчному (inline), блочному (block) или строчно-блочному (inline-block)

\* Свойство фона не наследуется.

\* Значение по умолчанию — прозрачный фон: transparent.

\* Фон нельзя задать частично. Блок заливается указанным цветом полностью.

\* В качестве значения можно указать только один цвет.

\* Изменение цвета фона можно анимировать при помощи свойства transition  
  
В чем отличие em и rem?

Относительные единицы измерения: определяются на основании каких-либо других величин, т.е размеров других указанных величин или элементов.

**em — размер** шрифта **относительно размера** шрифта **родительского элемента** и **rem — размер** шрифта **относительно размера** шрифта **корневого элемента**

**em в CSS — относительная единица измерения, используемая для определения размера элементов на веб-странице, в основном размера шрифта**. Поскольку это относительная единица измерения — 1 em равен размеру шрифта установленном в родительском элементе. Это означает, что если вы установите размер шрифта в родительском <div> в 20px и установите размер шрифта дочернего <div> равный 2em, то размер шрифта в дочернем элементе будет равен 40px.

**rem — ещё одна относительная единица измерения в CSS, которая расшифровывается как "root em" — "корневой em**". Поскольку мы знаем, что em равен размеру шрифта родительского элемента, мы можем сделать вывод, что "корневой em" относится к размеру шрифта корневого элемента, которым обычно является элемент <html>.

Как и em, rem наследует свой размер от родительского элемента, но родительский элемент на который смотрит rem, это не вышестоящий <div> или <section>, а самый первый элемент окружающий весь код, то есть элемент <html>. Давайте рассмотрим пример, используя предыдущий код. Тот же HTML код, только с дополнительным <div>.

В CSS единица измерения rem относится только к корневому элементу документа, а em относится непосредственно к родительскому элементу. Это означает, что размеры em наследуются от родительских элементов, а размеры rem только от корневого элемента.

Когда использовать em, а когда rem в CSS

rem рекомендуется использовать для глобальных значений CSS, таких, как font-size, margin и padding, особенно если вы хотите задать один font-size для всего документа и обеспечить его равномерное масштабирование, без влияния родительских элементов.

em подходит для значений, специфичных для конкретного элемента и его дочерних элементов. Это позволяет создать последовательный и гибкий макет, который хорошо адаптируется к разным размерам экрана и шрифта.

**Возможные проблемы при использовании em и rem в CSS**

**Сложные вычисления**: использование em и rem может привести к сложным вычислениям, особенно когда задействованы вложенные элементы. Это может затруднить точное прогнозирование и контроль размера элементов на странице.

**Проблемы наследования**: поскольку em относятся к размеру шрифта родительского элемента, могут возникнуть трудности с пониманием и контролированием того, как размеры наследуются на странице. Это может привести к неожиданным результатам и потребовать дополнительной отладки для устранения.

**Проблемы с производительностью**: В некоторых редких случаях использование em и rem может отрицательно сказаться на производительности, особенно в сочетании со сложными вычислениями или чрезмерным наследованием на странице.

Что такое vh и vw?  
**vw - относительная единица ширины окна,vh - относительная единица высоты окна.**

Приёмы отзывчивого веб-дизайна базируются на использовании процентных значений. Но проценты далеко не лучшее решение для каждого случая, так как они вычисляются относительно размеров ближайшего родительского элемента. Поэтому, если вы хотите использовать высоту и ширину окна браузера, лучше воспользоваться единицами vh и vw. Например, если высота окна браузера равна 900px, то 1vh будет равен 9px. Аналогично, если ширина окна браузера равна 1600px, то 1vw будет равен 16px.

Так как ширина элемента, указанная с помощью 100vw больше ширины области просмотра, то для создания полноэкранных фоновых изображений лучше использовать ширину 100%, которая будет равна ширине корневого элемента html.

**Vh-Эквивалентно 1% высоты окна браузера.**

**Vw-Эквивалентно 1% ширины окна браузера.**

Эти единицы вычисляются относительно размеров окна браузера. Для настольных компьютеров ширина окна браузера больше ширины области просмотра (добавляется ширина скроллбара), поэтому если для элемента установить ширину 100vw, то он выйдет за пределы html.

Как задать размер фоновому изображению?

**Значение background-size в CSS позволяет задавать размер фонового изображения**. Изображение может быть оставлено в исходном размере, растянуто, или подогнано под размеры доступного пространства.

Примечание: Если значение этого свойства не задано в сокращённом свойстве background, которое применяется к элементу после CSS-свойства background-size, то значение этого свойства затем сбрасывается до исходного значения c помощью сокращённого свойства.  
  
В чем разница между cover и contain?  
Свойство background-size имеет два зарезервированных значения:

**cover — Изображение масштабируется так, чтобы занять весь блок. В этом случае изображение может быть обрезано**

**contain — Изображение масштабируется так, чтобы максимально покрыть область блока, но не обрезать само изображение**

Важно: при создании страниц и добавлении фоновых изображений важно следить, чтобы исходное изображение имело достаточное разрешение. При попытке растянуть небольшое изображение мы увидим пикселизованную картинку, что не добавляет странице привлекательности

С помощью какого свойства можно задать, чтобы картинка оставалась на месте, а элемент скроллился?  
  
Свойство background-attachment устанавливает, будет ли прокручиваться фоновое изображение вместе с содержимым элемента. Изображение может быть зафиксировано и оставаться неподвижным, либо перемещаться совместно с документом. В CSS3 можно указать несколько значений для ряда фоновых изображений, перечисляя значения через запятую.

background-attachment: fixed | scroll | inherit

fixed Делает фоновое изображение элемента неподвижным.

scroll Позволяет перемещаться фону вместе с содержимым.

inherit Наследует значение родителя.

Что такое псевдоклассы?

**Псевдокласс — это селектор, который выбирает элементы, находящиеся в специфическом состоянии**, например, они являются первым элементом своего типа, или на них наведён указатель мыши. Они обычно действуют так, как если бы вы применили класс к какой-то части вашего документа, что часто помогает сократить избыточные классы в разметке и даёт более гибкий, удобный в поддержке код.

Псевдоклассы — это ключевые слова, которые начинаются с двоеточия :pseudo-class-nam  
  
Что такое псевдоэлементы?

**Псевдоэлемент в CSS — это ключевое слово, добавляемое к селектору, которое позволяет стилизовать определённую часть выбранного элемента**. Например, псевдоэлемент ::first-line может быть использован для изменения шрифта первой строки абзаца.

В отличие от псевдоэлементов, псевдоклассы могут быть использованы для стилизации элемента на основе его состояния

Примечание: Как правило, следует использовать двойное двоеточие (::) вместо одинарного (:). В этом состоит различие между псевдоклассами и псевдоэлементами. Однако, так как это различие не присутствовало в старых версиях спецификации W3C, большинство браузеров поддерживают оба синтаксиса для псевдоэлементов.  
  
Какое свойство обязательно для псевдоэлментов before и after?

Каждый из этих псевдоэлементов должен включать **специальное свойство content** , внутри которого и указывается, что должно быть внутри. Без этого свойства псевдоэлементы ::before и ::after работать не будут  
  
Как стилизовать placeholder?

Атрибут placeholder применяется для призыва к действию внутри пустых элементов input и textarea.

В IE и старых firefox (до 18-го) placeholder считается псевдоклассом, а в новых firefox, webkit и blink – псевдоэлементом. ::placeholder.

Нужно сказать, что поддерживаются не все возможные css свойства. Большинство современных браузеров позволяют изменить:

font (и сопутствующие свойства)

background (и сопутствующие свойства)

color

word-spacing

letter-spacing

text-decoration

vertical-align

text-transform

line-height

text-indent

text-overflow

opacity

Подсказка-плейсхолдер не должна выступать как замена тегу <label>. Основное назначение плейсхолдера — дать подсказку к правильному заполнению поля. Дизайнеры зачастую любят экономить место в формах и не рисуют лейблы у полей, заменяя их подсказками. Это не лучшая практика, поэтому если вам попадётся такой макет, обговорите с дизайнером возможность добавления лейблов.

Желательно стилизовать подсказку таким образом, чтобы визуально она отличалась от введённого текста. Как правило, её делают полупрозрачной, либо просто светлее вводимого текста.  
  
Что такое BEM?

**БЭМ (Блок, Элемент, Модификатор) — компонентный подход к веб-разработке. В его основе лежит принцип разделения интерфейса на независимые блоки. Он позволяет легко и быстро разрабатывать интерфейсы любой сложности и повторно использовать существующий код, избегая «Copy-Paste».**

Блок

Функционально независимый компонент страницы, который может быть повторно использован. В HTML блоки представлены атрибутом class.

Особенности:

Название блока характеризует смысл («что это?» — «меню»: menu, «кнопка»: button), а не состояние («какой, как выглядит?» — «красный»: red, «большой»: big).

Пример

<!-- Верно. Семантически осмысленный блок `error` -->

<div class="error"></div>

<!-- Неверно. Описывается внешний вид -->

<div class="red-text"></div>

Блок не должен влиять на свое окружение, т. е. блоку не следует задавать внешнюю геометрию (в виде отступов, границ, влияющих на размеры) и позиционирование.

В CSS по БЭМ также не рекомендуется использовать селекторы по тегам или id.

Таким образом обеспечивается независимость, при которой возможно повторное использование или перенос блоков с места на место

**Вложенность**

Блоки можно вкладывать друг в друга. Допустима любая вложенность блоков.

Элемент

Составная часть блока, которая не может использоваться в отрыве от него.

Особенности:

Название элемента характеризует смысл («что это?» — «пункт»: item, «текст»: text), а не состояние («какой, как выглядит?» — «красный»: red, «большой»: big).

Структура полного имени элемента соответствует схеме: имя-блока\_\_имя-элемента. Имя элемента отделяется от имени блока двумя подчеркиваниями (\_\_).

**Принципы работы с элементами: Вложенность Принадлежность Необязательность**

**Вложенность**

Элементы можно вкладывать друг в друга.

Допустима любая вложенность элементов.

Элемент — всегда часть блока, а не другого элемента. Это означает, что в названии элементов нельзя прописывать иерархию вида block\_\_elem1\_\_elem2.

Модификатор

Cущность, определяющая внешний вид, состояние или поведение блока либо элемента.

Особенности:

Название модификатора характеризует внешний вид («какой размер?», «какая тема?» и т. п. — «размер»: size\_s, «тема»: theme\_islands), состояние («чем отличается от прочих?» — «отключен»: disabled, «фокусированный»: focused) и поведение («как ведет себя?», «как взаимодействует с пользователем?» — «направление»: directions\_left-top).

Имя модификатора отделяется от имени блока или элемента одним подчеркиванием (\_).

**Типы модификаторов**

Булевый

Используют, когда важно только наличие или отсутствие модификатора, а его значение несущественно. Например, «отключен»: disabled. Считается, что при наличии булевого модификатора у сущности его значение равно true.

Структура полного имени модификатора соответствует схеме:

имя-блока\_имя-модификатора;

имя-блока\_\_имя-элемента\_имя-модификатора.

Пример

<!-- Блок `search-form` имеет булевый модификатор `focused` -->

<form class="search-form search-form\_focused">

<input class="search-form\_\_input">

<!-- Элемент `button` имеет булевый модификатор `disabled` -->

<button class="search-form\_\_button search-form\_\_button\_disabled">Найти</button>

</form>

Ключ-значение

Используют, когда важно значение модификатора. Например, «меню с темой оформления islands»: menu\_theme\_islands.

Структура полного имени модификатора соответствует схеме:

имя-блока\_имя-модификатора\_значение-модификатора;

имя-блока\_\_имя-элемента\_имя-модификатора\_значение-модификатора.

Пример

<!-- Блок `search-form` имеет модификатор `theme` со значением `islands` -->

<form class="search-form search-form\_theme\_islands">

<input class="search-form\_\_input">

<!-- Элемент `button` имеет модификатор `size` со значением `m` -->

<button class="search-form\_\_button search-form\_\_button\_size\_m">Найти</button>

</form>

<!--

Невозможно одновременно использовать два одинаковых модификатора

с разными значениями

-->

<form class="search-form

search-form\_theme\_islands

search-form\_theme\_lite">

<input class="search-form\_\_input">

<button class="search-form\_\_button

search-form\_\_button\_size\_s

search-form\_\_button\_size\_m">

Найти

</button>

</form>

**Принципы работы с модификаторами**

Модификатор нельзя использовать самостоятельно

С точки зрения БЭМ-методологии модификатор не может использоваться в отрыве от модифицируемого блока или элемента. Модификатор должен изменять вид, поведение или состояние сущности, а не заменять ее.

Пример

<!-- Верно. Блок `search-form` имеет модификатор `theme` со значением `islands`-->

<form class="search-form search-form\_theme\_islands">

<input class="search-form\_\_input">

<button class="search-form\_\_button">Найти</button>

</form>

<!-- Неверно. Отсутствует модифицируемый класс `search-form` -->

<form class="search-form\_theme\_islands">

<input class="search-form\_\_input">

<button class="search-form\_\_button">Найти</button>

</form>  
  
Что такое модификатор в BEM?

**Cущность, определяющая внешний вид, состояние или поведение блока либо элемента.**

Особенности:

Название модификатора характеризует внешний вид («какой размер?», «какая тема?» и т. п. — «размер»: size\_s, «тема»: theme\_islands), состояние («чем отличается от прочих?» — «отключен»: disabled, «фокусированный»: focused) и поведение («как ведет себя?», «как взаимодействует с пользователем?» — «направление»: directions\_left-top).

Имя модификатора отделяется от имени блока или элемента одним подчеркиванием (\_).

**Типы модификаторов**

Булевый

Используют, когда важно только наличие или отсутствие модификатора, а его значение несущественно. Например, «отключен»: disabled. Считается, что при наличии булевого модификатора у сущности его значение равно true.

Структура полного имени модификатора соответствует схеме:

имя-блока\_имя-модификатора;

имя-блока\_\_имя-элемента\_имя-модификатора.

Пример

<!-- Блок `search-form` имеет булевый модификатор `focused` -->

<form class="search-form search-form\_focused">

<input class="search-form\_\_input">

<!-- Элемент `button` имеет булевый модификатор `disabled` -->

<button class="search-form\_\_button search-form\_\_button\_disabled">Найти</button>

</form>

Ключ-значение

Используют, когда важно значение модификатора. Например, «меню с темой оформления islands»: menu\_theme\_islands.

Структура полного имени модификатора соответствует схеме:

имя-блока\_имя-модификатора\_значение-модификатора;

имя-блока\_\_имя-элемента\_имя-модификатора\_значение-модификатора.

Пример

<!-- Блок `search-form` имеет модификатор `theme` со значением `islands` -->

<form class="search-form search-form\_theme\_islands">

<input class="search-form\_\_input">

<!-- Элемент `button` имеет модификатор `size` со значением `m` -->

<button class="search-form\_\_button search-form\_\_button\_size\_m">Найти</button>

</form>

<!--

Невозможно одновременно использовать два одинаковых модификатора

с разными значениями

-->

<form class="search-form

search-form\_theme\_islands

search-form\_theme\_lite">

<input class="search-form\_\_input">

<button class="search-form\_\_button

search-form\_\_button\_size\_s

search-form\_\_button\_size\_m">

Найти

</button>

</form>

**Принципы работы с модификаторами**

Модификатор нельзя использовать самостоятельно

С точки зрения БЭМ-методологии модификатор не может использоваться в отрыве от модифицируемого блока или элемента. Модификатор должен изменять вид, поведение или состояние сущности, а не заменять ее.

Пример

<!-- Верно. Блок `search-form` имеет модификатор `theme` со значением `islands`-->

<form class="search-form search-form\_theme\_islands">

<input class="search-form\_\_input">

<button class="search-form\_\_button">Найти</button>

</form>

<!-- Неверно. Отсутствует модифицируемый класс `search-form` -->

<form class="search-form\_theme\_islands">

<input class="search-form\_\_input">

<button class="search-form\_\_button">Найти</button>

</form>

С помощью каких тегов создать таблицу?

**В HTML для создания таблиц существует набор семантических тегов: <table> <thead> <tbody>**

Таблица состоит из строк и столбцов ячеек, которые могут содержать текст и рисунки. Для добавления таблицы на веб-страницу используется тег <table>. Этот элемент служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются соответственно с помощью тегов <tr> и <td>. Таблица должна содержать хотя бы одну ячейку (пример 1). Допускается вместо тега <td> использовать тег <th>. Текст в ячейке, оформленной с помощью тега <th>, отображается браузером шрифтом жирного начертания и выравнивается по центру ячейки. В остальном, разницы между ячейками, созданными через теги <td> и <th> нет.

Пример

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Тег table</title>

</head>

<body>

<table border="1">

<tr>

<th>Ячейка 1</th>

<th>Ячейка 2</th>

</tr>

<tr>

<td>Ячейка 3</td>

<td>Ячейка 4</td>

</tr>

</table>

</body>

</html>

**Основные теги, используемые при создании таблицы:**

<table> — определяет начало и конец таблицы. Всё содержимое таблицы должно находиться между <table></table>.

<thead> — определяет заголовок таблицы. Заголовок может содержать одну или несколько строк, в которых могут использоваться теги <th> для определения заголовков столбцов.

<tbody> — определяет тело таблицы. Тело содержит одну или несколько строк, в которых могут использоваться теги <td> для определения содержимого ячеек.

<tfoot> — определяет нижний колонтитул таблицы. Колонтитул может содержать одну или несколько строк, в которых могут использоваться теги <td> для определения содержимого ячеек.

<tr> — определяет строку таблицы. Каждая строка должна находиться между тегами <tbody>, <thead> или <tfoot>.

<th> — определяет заголовок столбца или строки таблицы. Используется внутри тегов <thead> и <tr>.

<td> — определяет содержимое ячейки таблицы. Используется внутри тегов <tbody>, <tfoot> и <tr>.

<caption> — определяет заголовок таблицы, который будет размещен над таблицей. Используется внутри тега <table

Как очистить форму?

Кнопка сброса полей формы

**Тег input с атрибутом type в значении reset создает кнопку сброса полей HTML формы**. После нажатия на такую кнопку поля формы приобретут тот вид, который они имели при заходе на страницу (до внесенных пользователем изменений).

**Тег button с атрибутом type в значении reset создает кнопку** **сброса полей HTML формы** для использования в HTML формах, а также просто как элемент дизайна (в этом случае ее часто вкладывают в ссылку).  
  
Как убрать пробелы между img?

Изображения являются строчными элементами, поэтому написание в коде подряд несколько тегов <img> автоматически выстроит картинки по горизонтали. Тем не менее, с помощью стилей этим построением можно управлять более изящно.

<body>

<p>

<img src="images/thumb1.jpg" alt="Фотография 1" width="120" height="120">

<img src="images/thumb2.jpg" alt="Фотография 2" width="120" height="120">

<img src="images/thumb3.jpg" alt="Фотография 3" width="120" height="120">

<img src="images/thumb4.jpg" alt="Фотография 4" width="120" height="120">

</p>

</body>

Изображения выстраиваются в одну строку по горизонтали, а если некоторые из них не помещаются по ширине в окне, то переносятся на следующую строку

Обратите внимание на пробел между фотографиями, он появляется за счет переноса тега <img> в коде на новую строку. **Чтобы убрать пустой промежуток достаточно написать теги <img> в примере в одну строку**.

Для регулирования горизонтальных и вертикальных отступов между фотографиями применяется стилевое свойство margin-right и margin-bottom. В примере 2 кроме отступов к фотографиям также добавлена рамка и цветная область вокруг изображений.  
**Свойство gap CSS задаёт отступы между колонками и рядами. Является сокращением для свойств row-gap и column-gap**.

Что такое flexbox?

**Flexbox (Flexible Box) - это модуль CSS, который предоставляет более эффективный способ управления распределением элементов в контейнере, а также их выравниванием и распределением**

**Flexbox определяет набор CSS свойств для контейнера (flex-контейнер) и его дочерних элементов (flex-блоков).** Первое, что нужно сделать – это указать контейнеру display:flex или display:inline-flex.

Одним из основных понятий в fleхbox являются оси.

Главной осью flex-контейнера является направление, в соответствии с которым располагаются все его дочерние элементы.

Поперечной осью называется направление, перпендикулярное главной оси.

Главная ось в ltr локали по умолчанию располагается слева направо. Поперечная – сверху вниз. Направление главной оси flex-контейнера можно задавать, используя базовое css свойство flex-direction.  
  
  
Что такое специфичность в CSS?

**Специфичность - это способ, с помощью которого браузеры определяют, какие значения свойств CSS наиболее соответствуют элементу и, следовательно, будут применены. Специфичность основана на правилах соответствия, состоящих из селекторов CSS различных типов**.

Как вычисляется специфичность?

Специфичность представляет собой вес, придаваемый конкретному правилу CSS. Вес правила определяется количеством каждого из типов селекторов в данном правиле. Если у нескольких правил специфичность одинакова, то к элементу применяется последнее по порядку правило CSS. Специфичность имеет значение только в том случае, если один элемент соответствует нескольким правилам. Согласно спецификации CSS, правило для непосредственно соответствующего элемента всегда будет иметь больший приоритет, чем правила, унаследованные от предка.

Примечание: Взаимное расположение элементов в дереве документа не влияет на специфичность.

Типы селекторов

В следующем списке типы селекторов расположены по возрастанию специфичности:

селекторы типов элементов (например, h1) и псевдоэлементов (например, ::before).

селекторы классов (например, .example), селекторы атрибутов (например, [type="radio"]) и псевдоклассов (например, :hover).

селекторы идентификаторов (например, #example).

Универсальный селектор (\*), комбинаторы (+, >, ~, '``') и отрицающий псевдокласс (:not()) не влияют на специфичность. (Однако селекторы, объявленные внутри :not(), влияют)

Стили, объявленные в элементе (например, style="font-weight:bold"), всегда переопределяют любые правила из внешних файлов стилей и, таким образом, их специфичность можно считать наивысшей.

**Важное исключение из правил - !important**

Когда при объявлении стиля используется модификатор !important, это объявление получает наивысший приоритет среди всех прочих объявлений. Хотя технически модификатор !important не имеет со специфичностью ничего общего, он непосредственно на неё влияет. Поскольку !important усложняет отладку, нарушая естественное каскадирование ваших стилей, он не приветствуется и следует избегать его использования. Если к элементу применимы два взаимоисключающих стиля с модификатором !important, то применён будет стиль с большей специфичностью.

Несколько практических советов:

Всегда пытайтесь использовать специфичность, а !important используйте только в крайних случаях

Используйте !important только в страничных стилях, которые переопределяют стили сайта или внешние стили (стили библиотек, таких как Bootstrap или normalize.css)

Никогда не используйте !important, если вы пишете плагин или мэшап.

Никогда не используйте !important в общем CSS сайта.

Вместо !important можно:

Лучше использовать каскадные свойства CSS

Использовать более специфичные правила. Чтобы сделать правило более специфичным и повысить его приоритет, укажите один элемент или несколько перед нужным вам элементом  
  
В чем смысл таких тегов как section, header? Будет ли разница для конечного пользователя в разных тегах?  
  
**Семантическая вёрстка — подход к разметке, который опирается не на содержание сайта, а на смысловое предназначение каждого блока** и логическую структуру документа. Даже в этой статье есть заголовки разных уровней — это помогает читателю выстроить в голове структуру документа. Так и на странице сайта — только читатели будут немного другими.

**Чтобы сделать сайт доступным.** Зрячие пользователи могут без проблем с первого взгляда понять, где какая часть страницы находится — где заголовок, списки или изображения. Для незрячих или частично незрячих всё сложнее. Основной инструмент для просмотра сайтов не браузер, который отрисовывает страницу, а скринридер, который читает текст со страницы вслух.

Этот инструмент «зачитывает» содержимое страницы, и семантическая структура помогает ему лучше определять, какой сейчас блок, а пользователю понимать, о чём идёт речь. Таким образом семантическая разметка помогает б\_о\_льшему количеству пользователей работать с вашим сайтом. Например, наличие заголовков помогает незрячим в навигации по странице. У скринридеров есть функция навигации по заголовкам, что ускоряет знакомство с информацией на сайте.

**Чтобы сайт был выше в поисковиках.** Поисковики не разглашают правила ранжирования, но известно, что наличие семантической разметки страниц помогает поисковым ботам лучше понимать, что находится на странице, и в зависимости от этого ранжировать сайты в поисковой выдаче.

емантика прописана в стандартах. Многие разработчики по старинке пользуются конструкциями типа <div id="nav"> для обозначения навигации или других структурных элементов страницы. Тем временем в стандарте HTML есть несколько семантических тегов, которые рекомендуется использовать для разметки страниц вместо <div> и span. В спецификации для каждого семантического элемента описана его роль.

Ну и представьте, насколько проще читать <nav></nav> вместо <div class="nav"></div>.

Основные семантические теги HTML

Среди «старых» тегов из ранних версий HTML тоже есть семантические — например, тег <p>, который обозначает параграф. При этом теги <i> или <b> не семантические, потому что они не добавляют смысла выделенному тексту, а просто определяют его внешний вид.

Но в актуальной версии HTML Living Standard есть семантические теги почти для всех основных частей сайта, и лучше пользоваться ими. Вот несколько примеров семантических тегов.

**<article>**

Значение: независимая, отделяемая смысловая единица, например комментарий, твит, статья, виджет ВК и так далее.

Особенности: желателен заголовок внутри.

Типовые ошибки: путают с тегами <section> и <div>.

**<section>**

Значение: смысловой раздел документа. Неотделяемый, в отличие от <article>.

Особенности: желателен заголовок внутри.

Типовые ошибки: путают с тегами <article> и <div>.

**<aside>**

Значение: побочный, косвенный для страницы контент.

Особенности: может иметь свой заголовок. Может встречаться несколько раз на странице.

Типовые ошибки: считать <aside> тегом для «боковой панели» и размечать этим тегом основной контент, который связан с окружающими его элементами.

**<nav>**

Значение: навигационный раздел со ссылками на другие страницы или другие части страниц.

Особенности: используется для основной навигации, а не для всех групп ссылок. Основной является навигация или нет — на усмотрение верстальщика. Например, меню в подвале сайта можно не оборачивать в <nav>. В подвале обычно появляется краткий список ссылок (например, ссылка на главную, копирайт и условия) — это не является основной навигацией, семантически для такой информации предназначен <footer> сам по себе.

Типовые ошибки: многие считают, что в <nav> может быть только список навигационных ссылок, но согласно спецификации там может быть навигация в любой форме.

**<header>**

Значение: вводная часть смыслового раздела или всего сайта, обычно содержит подсказки и навигацию. Чаще всего повторяется на всех страницах сайта.

Особенности: этих элементов может быть несколько на странице.

Типовые ошибки: использовать только как шапку сайта.

**<main>**

Значение: основное, не повторяющееся на других страницах, содержание страницы.

Особенности: должен быть один на странице, исходя из определения.

Типовые ошибки: включать в этот тег то, что повторяется на других страницах (навигацию, копирайты и так далее).

**<footer>**

Значение: заключительная часть смыслового раздела или всего сайта, обычно содержит информацию об авторах, список литературы, копирайт и так далее. Чаще всего повторяется на всех страницах сайта.

Особенности: этих элементов может быть несколько на странице. Тег <footer> не обязан находиться в конце раздела.

Типовые ошибки: использовать только как подвал сайта.

Откуда берутся базовые стили для тегов?

По умолчанию, все HTML-теги имеют своё стандартное оформление. **Это оформление задают браузеры, причём каждый браузер может задать свой стиль.** Из-за этого одна и та же вёрстка может выводиться по разному, в зависимости от браузера, операционной системы и системных настроек.  
  
Что значит type submit?

Тип submit (от англ. "submit" ‒ «отправить») **создаёт кнопку отправки данных формы**. При активации данной кнопки происходит отправка всех данных формы на сервер. Это значение по умолчанию, если атрибут не указан или если атрибут динамически изменен на пустое или недопустимое значение.  
  
Что такое позиционирование?  
**Под позиционированием подразумевают расположение элемента в заданной системе координат. Можно выделить четыре типа позиционирования: нормальное, абсолютное, относительное и фиксированное**. В зависимости от выбранного типа, установленного через свойство position, будет изменяться и система координат.

Вместе со свойством position целесообразно использовать следующую комбинацию свойств (или лишь несколько из них): left, top, right, bottom и z-index, с их помощью элемент можно позиционировать более точно и в определенных случаях абсолютно независимо от соседних элементов. Свойства left, top, right, bottom отвечают за перемещение элемента от его первоначального положения в четырех направлениях: вправо, вниз, влево, вверх соответственно. Свойство z-index отвечает за перемещение элемента на другую плоскость (или уровень) и может принимать значения из диапазона от 1 до N.

Наиболее часто свойство position используется для определения взаимного расположения слоев

Что такое и зачем нужен iframe?  
  
Тег iFrame — элемент HTML, позволяющий встраивать на веб-страницу документы, видео и интерактивные медиафайлы и прочие части содержимого из других источников.Фронтенд-разработчики используют элемент для встраивания в страницы HTML-документов, которые, как правило, расположены на других сайтах. Это статические страницы, виджеты, апплеты, формы, видеоролики. Возможность использования iFrame предусмотрена во многих конструкторах сайтов, например в Google Sites.

Использование іFrame экономит время разработчика и упрощает хранение мультимедийных файлов: вместо того чтобы создавать видеоплеер с нуля и размещать видеоматериалы на собственном хостинге, можно загрузить все ролики на YouTube-канал и добавить их на страницу с помощью фреймов.  
  
  
Как бороться с переполнением контентом?

Переполнение контента внутри блока — распространенное явление при верстке контента. Переполнением считается ситуация, при которой контент внутри контейнера больше, чем размер самого контейнера. Распространенный случай такого поведения — использование контейнера с фиксированными значениями высоты и ширины.

Интересно: использование фиксированных значений высоты и ширины в большинстве случаев не является хорошей практикой. Так можно достаточно быстро сверстать блок по макету, но одновременно с этим отнимается возможность расширения функционала. Любой отход от изначального контента может привести к проблемам, связанным с выходом контента из контейнера. Используйте фиксированные значения высоты и ширины там, где это предполагается в дизайне или для создания специфичной функциональности.Например большая часть описания вышла за пределы блока. Браузеры в данном случае считают контент важнее, чем контейнер, в котором он лежит, и не скрывают текст. Это не лишено смысла, ведь главная часть любой страницы — ее контент. Без него вся страница не будет иметь никакого смысла

Такая ситуация называется переполнением, и CSS позволяет управлять им. Для этого существует три свойства:

overflow-x — управление переполнением по горизонтали.

overflow-y — управление переполнением по вертикали.

overflow — сокращенная запись двух предыдущих свойств. Если указать внутри только одно значение, то оно применится к двум осям одновременно. Наиболее распространенный вариант использования.

По умолчанию свойство имеет значение visible, которое и указывает на то, что при переполнении контент должен отрисовываться вне своего родителя. В противовес visible есть значение hidden. Его задача обратна — скрыть контент, который выходит за пределы своего родителя. При этом доступ к такому контенту теряется. При использовании свойства overflow важно помнить, что это свойство не является наследуемым, поэтому его необходимо указывать у каждого блока, с которым происходит переполнение. В дальнейшем вы увидите примеры таких реализаций.

Хоть теперь верстка не «сломалась» от переполнения, но прочитать описание профессии невозможно. Не хватает какой-нибудь полосы прокрутки внутри блока. Свойство overflow позволяет добавить полосу прокрутки в такой блок. Для этого может использоваться два значения:

Scroll auto  
  
  
Какие форматы изображений ты знаешь? Какие когда использовать?  
Существует различное множество форматов изображений: **jpg, jpeg, png, gif, raw, svg, bmp, ico, tiff, webp** и др. Каждый из них, по сути, отличается друг от друга лишь алгоритмом сжатия данных и возможностью поддержания прозрачности. И, между прочим, далеко не каждый из этих форматов поддерживается всеми браузерами.

Для простоты можно выделить 3 самые распространённые группы изображений:

Изображения без прозрачности

Изображения с прозрачностью

Иконки

Наиболее распространённые форматы

**JPG (JPEG)** .— наиболее оптимальный

Поддерживает 16 млн. цветов. Имеет небольшой размер файла.

**PNG .**— много весит

Поддерживает 16 млн. цветов. Имеет большой размер файла. Поддерживает прозрачность.

**GIF .**— низкое качество

Поддерживает всего 256 цветов. Имеет небольшой размер файла. Поддерживает прозрачность.

**WebP .** — набирающий популярность формат

Поддерживает 16 млн. цветов. Имеет минимальный размер файла. На данный момент имеет неполную поддержку браузерами.

**SVG** . — наиболее оптимальный

Векторная графика (при увеличении размера изображение не теряет качество). Имеет минимальный размер файла.

**ICO .** — устаревший формат

Используется сейчас исключительно для favicon. Имеет большой размер afqkf

Абсолютные и относительные единицы измерения?

Абсолютные единицы измерения привязаны к настоящим физическим размерам. Среди них пиксели, сантиметры, миллиметры, дюймы. Относительные единицы измерения описывают значения, которые зависят от других значений. Среди них em , rem , vh , ex и другие единицы измерения.  
  
Чем отличаются теги <div> и <span>? Когда и для чего их следует использовать?  
  
Tеги div и span имеют большую разницу между собой, хотя на первый взгляд это вовсе не заметно. **Разница заключается в том, что тег div является блочным элементов, а тег span – строковым.**

Блочный элемент – элемент, который является блоком. Каждый блок без использования дополнительных стилей будет отображаться с новой строки, так как в ширину они по умолчанию занимают 100% ширины экрана.

Строковый элемент – элемент, который не является блоком, а соответственно не нуждается в дополнительных пропусках строчек или же каких-то отступах.

Оба тега используются вместе со стилями и использовать их без стилей не является лучшей затеей.  
Главное отличие между тегами div и span заключается в том, как они отображаются на странице. Тег div обычно используется для создания больших блоков содержимого, таких как разделы, заголовки или нижние колонтитулы. Эти блоки могут содержать другие элементы и управлять их расположением и стилями с помощью CSS.

С другой стороны, тег span обычно используется для форматирования и стилизации отдельных слов или фраз внутри текстового контента. Вы можете применить стили к тегу span, чтобы выделить определенное слово жирным шрифтом или изменить его цвет. Тег span позволяет точнее управлять внешним видом отдельных элементов внутри текста.

Таким образом, выбор между тегами div и span зависит от конкретных задач и нужд проекта. Если вам нужно создать большой блок контента с возможностью управления его стилями и расположением, используйте тег div. Если вам нужно выделить или стилизовать отдельные элементы внутри текста, воспользуйтесь тегом span.  
  
Как работают атрибуты 'alt' и 'title' в теге <img>? Зачем они нужны и в чем разница между ними?  
  
При использовании тега <img> для создания изображений используйте атрибут **alt, чтобы предоставить обязательное описание, которое гарантирует доступность содержимого для скринридеров и в ситуациях, когда изображения не загружаются. Атрибут title предоставляет дополнительную информацию и всплывающую подсказку, расширяя контекст изображения**. Оба эти атрибута важны для SEO и удобства использования, но alt имеет особое значение в связи со стандартами доступности

Учет контекста

В зависимости от контекста использования изображения, следует определить степень важности атрибутов alt и title. Для интернет-магазина важны подробные описания alt, в то время как для блога всплывающие подсказки title могут стимулировать вовлечённость читателей.

Избегайте дублирования

Не стоит дублировать текст в alt и title. Это создает лишнюю нагрузку на ассистивные технологии. Тексты должны дополнять друг друга, а не повторяться.

Сокращайте текст всплывающих подсказок

Убедитесь, что текст всплывающей подсказки в title максимально релевантен и конкретен. Чем проще и яснее, тем лучше.

Обеспечьте одинаковое восприятие всплывающих подсказок в разных браузерах

Используйте title, чтобы гарантировать однородное впечатление от взаимодействия пользователей с вашим сайтом в разных браузерах. Разные браузеры могут по-разному интерпретировать alt при взаимодействии с всплывающими подсказками, что может привести к недоразумениям.

Что такое form в HTML? Какие элементы могут быть включены внутри тега <form>, и какие атрибуты

они могут иметь?

**Тег <form> устанавливает форму на веб-странице. Форма предназначена для обмена данными между пользователем и сервером.** Область применения форм не ограничена отправкой данных на сервер, с помощью клиентских скриптов можно получить доступ к любому элементу формы, изменять его и применять по своему усмотрению.

Документ может содержать любое количество форм, но одновременно на сервер может быть отправлена только одна форма. По этой причине данные форм должны быть независимы друг от друга.

Для отправки формы на сервер используется кнопка Submit, того же можно добиться, если нажать клавишу Enter в пределах формы. Если кнопка Submit отсутствует в форме, клавиша Enter имитирует ее использование.

Когда форма отправляется на сервер, управление данными передается программе, заданной атрибутом action тега <form>. Предварительно браузер подготавливает информацию в виде пары «имя=значение», где имя определяется атрибутом name тега <input>, а значение введено пользователем или установлено в поле формы по умолчанию.

Допускается внутрь контейнера <form> помещать другие теги, при этом сама форма никак не отображается на веб-странице, видны только ее элементы и результаты вложенных тегов.

Атрибуты

**accept-charset**

Устанавливает кодировку, в которой сервер может принимать и обрабатывать данные.

**action**

Адрес программы или документа, который обрабатывает данные формы.

**autocomplete**

Включает автозаполнение полей формы.

**enctype**

Способ кодирования данных формы.

**method**

Метод протокола HTTP.

**name**

Имя формы.

**novalidate**

Отменяет встроенную проверку данных формы на корректность ввода.

**target**

Имя окна или фрейма, куда обработчик будет загружать возвращаемый результат.

Также для этого тега доступны универсальные атрибуты и события.

Закрывающий тег-Обязателен.  
  
  
Какие есть способы создания анимаций в CSS? Какие свойства и функции используются для управления

анимацией?

CSS-анимации позволяют анимировать переходы от одной конфигурации CSS стилей к другой. CSS-анимации состоят из двух компонентов: стилевое описание анимации и набор ключевых кадров, определяющих начальное, конечное и, возможно, промежуточное состояние анимируемых стилей.

Есть три преимущества CSS-анимации перед традиционными способами:

Простота использования для простых анимаций; вы можете создать анимацию, не зная JavaScript.

Анимации будут хорошо работать даже при умеренных нагрузках системы. Простые анимации на JavaScript, если они плохо написаны, часто выполняются плохо. Движок может использовать frame-skipping и другие техники, чтобы сохранить производительность на таком высоком уровне .

Позволяет браузеру контролировать последовательность анимации, тем самым оптимизируя производительность и эффективность браузера. Например, уменьшая частоту обновления кадров анимации в непросматриваемых в данный момент вкладках.

Конфигурирование анимации

**Чтобы создать CSS-анимацию вы должны добавить в стиль элемента, который хотите анимировать, свойство animation или его подсвойства**. Это позволит вам настроить ускорение и продолжительность анимации, а также другие детали того, как анимация должна протекать. Это не поможет вам настроить внешний вид анимации, который настраивается с помощью @keyframes, рассматриваемой далее в Определение последовательности анимации с помощью ключевых кадров.

Свойство animation имеет следующие подсвойства:

**animation-name**

Определяет имя @keyframes, настраивающего кадры анимации.

**animation-duration**

Определяет время, в течение которого должен пройти один цикл анимации.

**animation-timing-function**

Настраивает ускорение анимации.

**animation-delay**

Настраивает задержку между временем загрузки элемента и временем начала анимации.

**animation-iteration-count**

Определяет количество повторений анимации; вы можете использовать значение infinite для бесконечного повторения анимации.

**animation-direction**

Даёт возможность при каждом повторе анимации идти по альтернативному пути, либо сбросить все значения и повторить анимацию.

**animation-fill-mode**

Настраивает значения, используемые анимацией, до и после исполнения.

**animation-play-state**

Позволяет приостановить и возобновить анимацию.

Определение последовательности анимации с помощью ключевых кадров

После того, как вы настроили временные свойства (продолжительность, ускорение) анимации, вы должны определить внешний вид анимации. Это делается с помощью двух и более ключевых кадров после @keyframes. Каждый кадр описывает, как должен выглядеть анимированный элемент в текущий момент.

В то время, как временные характеристики (продолжительность анимации) указываются в стилях для анимируемого элемента, ключевые кадры используют <percentage>, чтобы определить стадию протекания анимации. 0% означает начало анимации, а 100% её конец. Так как эти значения очень важны, то для них придумали специальные слова: from и to.  
  
Что за свойство outline?  
  
**CSS-свойство outline-style задаёт стиль обводки элемента. Обводка — линия вокруг элемента за пределами рамки (border).**

Значения

**Auto**-Оставляет стиль обводки на усмотрение браузера.

**None-**Отсутствие обводки. Значение outline-width в таком случае будет 0.

**Dotted-**Обводка в виде линии из точек.

**Dashed-**Обводка в виде пунктирной линии.

**Solid-**Обводка в виде сплошной линии.

**Double-**Обводка из двух сплошных линий. Значение outline-width — это сумма двух линий и расстояния между ними.

**Groove-**Обводка выглядит так, словно она выгравирована на странице.

**Ridge-**Противоположность groove: обводка полностью объёмная, словно выдавлена из страницы.

**Inset-**Обводка выглядит так, словно блок погружается в страницу.

**Outset-**Противоположность inset: обводка выглядит так, словно приподнимает блок на странице.

Какие способы центрировть элемент вы знаете?

**Горизонтальное выравнивание**

Выравнивание по центру строкового или строково-\* элемента

Выравнивать по центру строковые элементы внутри блочного родительского элемента можно достаточно просто:

.center-children {

text-align: center;

}

Это будет работать для inline, inline-block, inline-table, inline-flex, и т. д.

Выравнивание по центру блочного элемента

Выравнять по центру блочный элемент вы можете установив свойствам margin-left и margin-right значение auto (у него также должна быть задана ширина, иначе элемент просто займёт всю ширину контента и не будет необходимости в выравнивании). Чаще всего это делается с помощью сокращённого свойства, вот так:

.center-me {

margin: 0 auto;

}

Это будет работать независимо от того какая ширина установленная у блочного элемента или его родителя.

Обратите внимание, что вы не сможете обтекать элемент выравненный по центру таким способом. Однако и для этого есть трюк.

Выравнивание по центру нескольких блочных элементов

Если есть два или более блочных элементов, которые вам нужно выравнять горизонтально по центру в пределах одной строки, то возможно вам лучше изменить их тип отображения. Ниже приведён пример изменения их типа отображения на inline-block и пример с Flexbox:

Если эти несколько блоков расположены на разных строках, то техника с автоматическими отступами по прежнему прекрасна:

**Вертикальное выравнивание**

Вертикальное выравнивание по центру в CSS немного сложнее.

Выравнивание по центру строкового или строково-\* элемента

Выравнивание элемента с одной строкой

Иногда строковые/текстовые элементы могут выравниваться по центру просто потому что у них одинаковое значения для свойств padding-top и padding-bottom.

.link {

padding-top: 30px;

padding-bottom: 30px;

}

Если для вас по каким-то причинам padding - это не вариант, а вы пытаетесь выровнять по центру некоторый текст и точно знаете, что он не будет обёрнут чем-то ещё, то есть есть небольшой трюк, суть которого заключается в том чтобы установить свойство line-height равным значению высоты элемента.

.center-text-trick {

height: 100px;

line-height: 100px;

white-space: nowrap;

}

Выравнивание многострочного элемента

Одинаковые отступы сверху и снизу могут создать эффект выравнивания по центру и для многострочных элементов, но если это не сработает, то возможно что элемент в котором находится текст является ячейкой таблицы, либо такое его поведение явно задано со помощью CSS. В данном свойство vertical-align обрабатывает это, в отличии от того что оно обычно обрабатывает выравнивание элементов внутри строки.

Если это что-то похожее на таблицу, возможно вы сможете использовать Flexbox ? Отдельный Flex-элемент может быть легко расположен по центру внутри Flex-контейнера.

.flex-center-vertically {

display: flex;

justify-content: center;

flex-direction: column;

height: 400px;

}

Помните, что это на самом деле актуально когда у контейнера есть фиксированная высота (px, %, и другие), вот почему здесь у контейнера задана высота.

Если обе техники приведённые выше нельзя применить, вы можете использовать технику «призрачного элемента», в которой псевдоэлемент с высотой в 100% размещается внутри контейнера и тех выравнивается вертикально по центру него.

.ghost-center {

position: relative;

}

.ghost-center::before {

content: " ";

display: inline-block;

height: 100%;

width: 1%;

vertical-align: middle;

}

.ghost-center p {

display: inline-block;

vertical-align: middle;

}

Выравнивание по центру блочного элемента

Выравнивание элемента с известной высотой

Довольно часто мы не можем знать высоту, по многим причинам: если изменяется ширина, текст может форматироваться и изменять высоту. Разная стилизация текста может изменять высоту. Разный размер шрифта может изменить высоту. Элементы с фиксированным соотношением сторон, такие как изображения, могут изменять высоту при изменении собственных размеров и т.д.

Но если вы знаете высоту элемента, то можете выравнивать по центру вот так:

.parent {

position: relative;

}

.child {

position: absolute;

top: 50%;

height: 100px;

margin-top: -50px; /\* account for padding and border if not using box-sizing: border-box; \*/

}

Выравнивание элемента с неизвестной высотой

Даже если вы не знаете высоту элемента, его всё ещё можно протолкнуть на 50% высоты его родителя, а после поднять на 50% его собственной высоты:

.parent {

position: relative;

}

.child {

position: absolute;

top: 50%;

transform: translateY(-50%);

}

Выравнивание элемента с помощью Flexbox

Это можно очень просто сделать с помощью Flexbox.

.parent {

display: flex;

flex-direction: column;

justify-content: center;

}

**Горизонтальное и вертикальное выравнивание**

Вы можете комбинировать представленные выше техники для того чтобы получить идеально выравненные по центру элементы. Но обычно их можно разделить на 4 группы:

Выравнивание элемента с фиксированной шириной и высотой

Использование отрицательных отступов (maring) равных половине этой ширины и высоты после абсолютного позиционирования элемента 50% / 50% выравняют элемент по центру. У этого способа достаточно хорошая кроссбраузерная поддержка:

.parent {

position: relative;

}

.child {

width: 300px;

height: 100px;

padding: 20px;

position: absolute;

top: 50%;

left: 50%;

margin: -70px 0 0 -170px;

}

**Выравнивание элемента с неизвестной шириной и высотой**

Если вы не знаете ширину или высоту элемента, то можете использовать свойство transform и отрицательный translate в 50% в обоих направлениях (в зависимости от текущей ширины / высоты элемента) до центра:

.parent {

position: relative;

}

.child {

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

}

**Выравнивание элемента с помощью Flexbox**

Чтобы выровнять элемент по центру с помощью Flexbox вам нужно использовать два свойства центрирования:

.parent {

display: flex;

justify-content: center;

align-items: center;

}

**Выравнивание элемента с помощью CSS Grid**

Это просто небольшой трюк (присланный Lance Janssen), который в большей степени будет работать для одного элемента:

body, html {

height: 100%;

display: grid;

}

span { /\* thing to center \*/

margin: auto;

}  
  
Что такое препроцессоры?

**Препроцессоры CSS - это инструменты, которые расширяют стандартный язык таблиц стилей CSS, добавляя дополнительные функции, которые облегчают и ускоряют процесс написания стилей для веб-страниц. Они позволяют использовать переменные, вложенные правила**, миксины (повторно используемые блоки стилей) и другие возможности, которых нет в обычном CSS.

Что такое grid?  
  
**Grid (сетка) - это CSS-модуль, который предоставляет разработчикам удобный способ создания двумерной сетки для размещения элементов на веб-странице**. С помощью Grid можно легко создавать сложные макеты, управлять распределением элементов по строкам и столбцам, а также их выравниванием.

В чем разница между "сбросом" и "нормализацией" CSS? Что бы вы выбрали и почему?  
  
Сброс (reset) сбрасывает все стили элементов.

Normalize приводит стили к единому виду во всех браузерах.

Я бы выбрал Normalize из-за кроссбраузерности.

Опишите z-индекс и то, как это работает?  
  
**Любой элемент в HTML документе может быть либло на переднем, либо на заднем плане. Если свойства z-index и позиционирование не заданы явно, то порядок наложения равен порядку следования элементов в HTML.**

**Если позиционирование элементов (и их детей) указано явно, то такие элементы будут перекрывать собой элементы без явно заданного свойства позиционирования.**

**Если задан z-index. Во-первых, z-index учитывается только на явно позиционированных элементах. Если попытаться установить z-index на непозиционированный элемент, то ничего не произойдет. Во-вторых, значения z-index могут создавать контекст наложения**.

Контекст наложения

Элементы с общими родителями, перемещающиеся на передний или задний план вместе, известны как контекст наложения.

Контекст наложения имеет свой корневой элемент в HTML структуре. В момент формирования нового контекста на элементе все дочерние элементы также попадают в этот контекст и занимают своё место в порядке наложения. Если элемент располагается в самом низу одного контекста наложения, то никаким образом не получится отобразить его над другим элементом в соседнем контексте наложения, располагающимся выше по иерархии, даже с установленным z-index равным миллиону.

Новый контекст может быть сформирован в следующих случаях:

Если элемент — корневой элемент документа

Если элемент позиционирован не статически и его значение z-index не равно auto.

Если элемент имеет прозрачность менее 1.

Порядок наложения:

Background и border элемента.

Позиционированные элементы и его дети с z-index < 0.

Элементы блочного уровня в нормальном потоке, то есть у которых position = static.

Плавающие(float) элементы.

inline-элементы.

Элементы с z-index = 0 или auto. Элементы со свойством opacity < 1

Какие существуют способы визуально скрыть контент?  
  
visibility: hidden

width: 0px; height: 0px;

Что \* { box-sizing: border-box; } делает? В чем его преимущества?  
По умолчанию все элементы имеют box-sizing: content-box. Размеры блока рассчитываются из размеров контента.

box-sizing: border-box меняет то, как расчитываются width и height. Border и padding включаются в расчёт. Высота будет состоять из: высота контента + вертикальные padding'и + ширина вертикальных border. Ширина будет состоять из: ширина контента + горизонтальные padding'и + ширина горизонтальных border.

В чем разница между селекторами "nth-of-type()" и "nth-child()"?

В чем разница между относительным, фиксированным, абсолютным и статически расположенным элементом?

Можете ли вы привести пример @media свойства?  
  
Медиа запросы поволяют адаптировать страницу для различных типов устройств, таких как: принтеры, речевые браузеры, устройства Брайля, телевизоры и так далее.

1)all - Подходит для всех устройств.

2)print - Для принтеров.

3)screen - Предназначен в первую очередь для цветных компьютерных экранов.

4)speech - Предназначен для синтезаторов речи.