**CSS**

**css\_1. Какие значения бывают у свойства display? В чём их особенность?**

Каждый HTML-элемент по умолчанию имеет свой тип отображения.

Свойство display (CSS) определяет тип отображения (display type) элемента, имеющий два основных свойства, определяющих генерацию блоков — внешний тип отображения определяет расположение блока в схеме потока (flow layout) и внутренний тип отображения определяет расположение дочерних элементов блока.

Значений очень много. Вот некоторые:

display: none

Полностью скрывает элемент со страницы, не удаляя его при этом из HTML-разметки.

Элемент удаляется из дерева оступности - это приводит к тому, что элемент и все его дочерние элементы больше не будут восприниматься технологиями чтения экрана.

display: block - сделает элемент блочным

display: inline - сделает элемент строчным

display: inline-block — элемент будет вести себя снаружи как строчный, внутри как блочный.

display: flex / inline-flex — элемент становится флекс-контейнером, ведёт себя как блочный или строчный, а вложенные элементы становятся флекс-элементами.

display: grid / inline-grid — элемент становится грид-контейнером. Снаружи грид-контейнер ведёт себя как блочный или строчный. Дочерние элементы начинают подчиняться правилам грид-раскладки.

display: table — элемент становится грид-контейнером. Снаружи грид-контейнер ведёт себя как блок. Дочерние элементы такого контейнера начинают подчиняться правилам грид-раскладки.

**css\_1.1 В чём разница между inline и inline-block?**

Размер inline элементов определяются содержимым - установить их через CSS-свойства нельзя. Так же не выйдет указать для них вертикальные margin и padding.

inline-block элементы также располагаются в потоке текста, но им можно устанавливать размеры, а так же любые значения margin и padding. Это позволит элементу отображаться в строке, но занимать пространство как блочному элементу.

**css\_2. Что делает свойство margin? Какие есть особенности у отрицательных значений? Что такое margin collapsing и в каких случаях это происходит?**

Задаёт размер внешнего отступа вокруг элемента. В качестве значения можно указать одно, два, три или четыре числа.

Отрицательные margin-left/top вместо отступа, наоборот, ставит элемент ближе к соседнему верхнему/левому элементу, причем при сдвиге соседние правые/нижние элементы занимают освободившееся пространство.

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



Margin collapsing - схлопывание внешних отступов - это такое поведение когда вертикальные отступы двух блочных элементов соприкасаются, они не складываются, а накладываются друг на друга, образуя общее пространство, равное большему из отступов.

Происходит в трех случаях:

- схлопываются отступы margin-top и margin-bottom соседних элементов.

Не произойдет:

- если у родительского элемента указано свойство display: grid или display: flex, то отступы дочерних элементов не схлопнутся.

- отступы абсолютно позиционированных элементов(position: absolute) никогда не схлопываются.

Проверить дома на компе

**css\_3. Что делает свойство vertical-align? Какие у него есть значения?**

**css\_4. Что такое padding?**

Свойство padding — это внутренний отступ — позволяет оттолкнуть контент от границ родительского элемента.

В качестве значения можно указать одно, два, три или четыре числа. Браузер будет читать это значение по-разному в зависимости от количества указанных значений.

* padding: 10px 15px 20px 30px — читается по часовой стрелке начиная с верхней границы. Сверху—> справа—> снизу—> слева.
* padding: 15px — для всех четырёх сторон 15 px.
* padding: 15px 20px — Первое значение для верхнего и нижнего отступов. Второе значение для боковых отступов.
* padding: 10px 5px 25px — первое значение для верхнего отступа, второе значение для боковых, последнее для нижнего отступа.

Вертикальные отступы не схлопываются так как margin.

**css\_5. Что такое box-sizing?**

CSS свойство которое определяет как вычисляется общая ширина и высота элемента. Может принимать 2 значения:

content-box - значение по умолчанию, свойства width и height включают исключительно контент, и не включают padding и border. Размеры элемента рассчитываются следующим образом: width = ширина контента, и height = высота контента. (Границы и внутренние отступы не включаются в вычисление.)

border-box - свойства width и height включают контент, внутренний отступ и границы. Размеры элемента рассчитываются следующим образом: width = граница + внутренний отступ + ширина контента, и height = граница + внутренний отступ + высота контента.

**css\_6. Шрифты.**

**css\_6.1. Как и в каких единицах можно задавать размер шрифта?**

CSS Свойство font-size определяет размер шрифта.

em -относительная единица, которая определяется по родительскому элементу. 1em — это стандартный размер текста, который установлен в браузере. Обычно он равен 16px. Соответственно, 2em будут равны 32px, а 0.5em — 8px. Если задаёшь font-size внутри другого элемента, то em будет рассчитан относительно этого элемента.

rem — работает похожим образом, что em, но в этом случае размер шрифта зависит не от родительского элемента, а от корневого html элемента. Другими словами, так можно не зависеть от размера родительского элемента.

% — значение в процентах, например 80%. Работает похожим образом, что и единица em. За 100% берётся размер текста в родительском элементе. Значение может быть только положительным.

px — можно задать значение в пикселях.

xx-small, x-small, small, medium, large, x-large, xx-large — абсолютные значения, которые зависят от настроек браузера и операционной системы. Обычно не используют.

larger, smaller — больший или меньший размер шрифта относительно родительского элемента. Обычно не используют.

**css\_6.2. В чём заключаются отличия у разных семейств шрифтов (с засечками, без них и моноширинные шрифты)? Как они называются на английском?**

serif— шрифтовое семейство с засечками, например, «Times». Символы имеют завершающие штрихи, расширяющиеся или сужающиеся концы, или имеют фактически зазубренные окончания.

sans-serif — шрифтовое семейство без засечек, например, «Arial»; Символы имеют гладкие окончания.

monospace— семейства моноширинных шрифтов с одинаковой шириной каждого символа. Похожи на текст, набранный на печатной машинке.

**css\_6.3. Какие есть особенности подключения шрифтов через font-face?**

Директивой @font-face можно подключить любой шрифт из файла или url.

Директива @font-face по своей «анатомии» отличается от стандартных CSS-правил, где есть селектор и применяемые к нему правила. Основное отличие в том, что на месте селектора активируется функция, адресованная браузеру.

Внутри этого правила должно быть два обязательных свойства:

- font-family - название шрифта, которое будет использоваться в CSS. Название может отличаться от настоящего названия этого шрифта.

- src - ссылки на файлы, из которых браузер может взять шрифт.

Нужно указать ссылки на файлы в нескольких форматах, чтобы любой браузер — старый или новый — мог загрузить шрифт в понятном ему формате. Принято хранить и подключать шрифты в форматах WOFF, WOFF2, SVG.

Подключение из внешних файлов происходит при помощи конструкции url("ссылка-на-файл"). Несколько файлов можно подключить, перечислив эти конструкции несколько раз через запятую.

Если указаны ссылки на несколько файлов в разных форматах, то после url() оставляют браузеру подсказку в виде записи format("формат-подключаемого-файла").

Т.к. есть вероятность, что нужный шрифт уже есть на компьютере пользователя, то браузеру будет удобнее и быстрее взять шрифт из системы, чем читать файл. Для этого в качестве одного или нескольких значений у свойства src нужно указать конструкцию local("имя-шрифта-в-системе"). Нужно указывать локальные названия до ссылок на внешние файлы, чтобы браузер, найдя шрифт в системе под таким именем, даже не пошёл загружать и читать внешние файлы.

Для разных начертаний одного и того же шрифта(например bold и italic) нужно подключить разные файлы, содержащие эти начертания, и тут есть два подхода.

1. Использовать разные @font-face для подключения разных начертаний шрифта, называя их разными именами. Например, "Lexend Peta Regular" и "Lexend Peta Bold".



В коде, используем в разных местах разные начертания шрифтов:



2. Подключить все файлы начертаний, но использовать для них одно и то же имя, указав, для какого стиля и жирности этот файл, т.е. в @font-face дополнительно указать свойства font-style или font-weight.







Тогда не нужно будет переопределять шрифт для разных элементов, достаточно будет указывать нужный стиль или жирность шрифта.



@font-face нужно объявлять до обращения в CSS к этому шрифту. Принято подключать шрифты в самом начале файла стилей (но после всех @import).

**css\_7. Цвета.  
css\_7.1. Какие есть способы задать цвет? Как задать прозрачный цвет?**

**-** ключевым словом для цвета. Ключевые слова регистронезависимые.  
- ключевое слово transparent представляет собой полностью прозрачный цвет. Это делает фон позади цветного элемента полностью видимым. Технически transparent — это ярлык для rgba (0,0,0,0).

- ключевое слово currentColor устанавливает цвет, указанный в свойстве color элемента.

- цветовая модель RGB определяет цвет в соответствии с его красными, зелёными и синими компонентами. Дополнительный Альфа-компонент представляет прозрачность цвета.

Можно использовать функциональную нотацию (rgb(0 63 255)), или шестнадцатеричное представление (#ff0000).

В функциональной нотации, каждое из трёх значений отвечает за отдельный канал RGB и может быть записано числом от 0 до 255 или в процентах. Для добавления прозрачности, после записи каналов нужно поставить слэш и записать нужное значение от 0 до 1 или в процентах, например rgb(0 63 255 / 0.5) для полупрозрачного синего.

Раньше синтаксис RGB отличался от современного, всё ещё можно встретить его в коде или выбрать для лучшей кроссбраузерности. Для разделения каналов внутри функции нужно было использовать запятые rgb(0, 63, 255), а для добавления прозрачности — специальную функцию rgba(), которая принимала последним параметром прозрачность цвета, например rgba(255, 0, 0, 0.5).

В шестнадцатеричном представлении сплошные цвета записываются в формате #RRGGBB или в сокращённом #RGB (если символы каждой группы одинаковые).

Если нужно указать прозрачность, она добавляется в конце в HEX-формате #RRGGBBAA или #RGBA, например, #00990055 или #0905.

Раньше нельзя было задать прозрачность в HEX-цветах и приходилось использовать функцию rgba(), но сегодня у этого способа неплохая кроссбраузерность. Проблема одна: мало кто сходу сможет рассчитать 50% в шестнадцатеричном формате, поэтому для указания прозрачности цвета чаще используется функция rgb(), где прозрачность можно задать в дробях или процентах.

- Цветовая модель HSL. Описывает те же цвета, что и RGB, но иначе: H — Hue (оттенок), S — Saturation (насыщенность), L — Lightness (светлота). Например, hsl(120 100% 50%) для зелёного. Первое значение оттенка задаётся в градусах и его можно записать просто как 120 (как чаще всего и делают) или с указанием единиц поворота: deg, turn, grad, rad. Доступны значения от 0 до 360, если указываете поворот в градусах. Отрицательные значения не допускаются. Второе и третье значение указываются в процентах. Прозрачность добавляется так же, как в rgb, с помощью слэша со значением, например hsl(120 100% 50% / 0.5) полупрозрачный зелёный. Синтаксис hsl() отличался от современного точно так же, как rgb(): нужны были запятые и специальная функция hsla() для задания прозрачности цвета. Используйте старый синтаксис для лучшей совместимости.

- Цветовая модель HWB. H — hue (оттенок), W — whiteness (количество белого), B — blackness (количество чёрного). Первое значение оттенка задаётся в градусах и его можно записать просто как 120 (как чаще всего и делают) или с указанием единиц поворота: deg, turn, grad, rad. Доступны значения от 0 до 360, если указываете поворот в градусах. Второе и третье значения указываются в процентах. Допустимо указывать прозрачность цвета четвёртым параметром через слэш. hwb(45 45% 20% / 0.6);

**css\_7.2. Как в другом css-свойстве получить текущий цвет?**

Ключевое слово currentColor.

**css\_7.3. Какие цветовые модели допустимо использовать в CSS?**

RGB, HSL, HBW, HEX.

**css\_7.4. Какую роль играет альфа-канал? Как им пользоваться?**

Альфа-канал используется для управления прозрачностью (непрозрачностью) элементов. Альфа-канал устанавливается четвертым значением в функции RGB, RGBA, HSL, HLSA, HBW, можно установить и в HEX кодах - он добавляется в конце в HEX #RRGGBBAA или #RGBA, например, #00990055 или #0905.

**css\_8. Что такое поток документа?**

Это принцип организации элементов на странице при отсутствии стилей. Даже если к странице не подключено никаких стилей, к каждому элементу всё равно будут применяться CSS-правила, «зашитые» в движке браузера. На основании этих правил условно все элементы на странице можно разделить на две категории: блочные (block) и строчные (inline). Поменять стандартное поведение можно при помощи CSS-свойства display. Если вообще не применять никаких стилей, браузер формирует из элементов **нормальный поток.** Поведение блочных элементов в нормальном потоке отличается от поведения строчных.

Правила расположения строчных и блочных элементов в нормальном потоке называются **контекстом форматирования**. Блочные элементы участвуют в формировании **блочного** контекста форматирования. Строчные элементы формируют **строчный** контекст форматирования. Расположение элементов в контексте форматирования зависит от направления письма для конкретного языка. Например, тексты на европейских языках мы читаем и пишем слева направо сверху вниз. Это означает, что по умолчанию контекст форматирования располагает блочные элементы сверху вниз, а строчные — слева направо. Но, например, в случае с восточноазиатскими языками, такими как китайский, японский и корейский, когда используется вертикальное письмо, мы видим совершенно другую картину: блочные элементы будут располагаться справа налево, а строчные — сверху вниз.

Блочные элементы в нормальном потоке располагаются друг под другом, всегда занимая всю доступную ширину родителя. Высота блочного элемента по умолчанию равна высоте его содержимого.

Даже если ширина блочного элемента явно задана и позволяет разместить справа ещё один такой элемент, поток всё равно продолжит выстраивать их друг под другом.



Строчные элементы располагаются друг за другом, как слова в предложении. В зависимости от направления письма в конкретном языке элементы могут располагаться слева направо (например, в русском языке), справа налево (как, например, в иврите) и даже сверху вниз (как иероглифы и знаки слоговых азбук в японском вертикальном письме). Ширина и высота строчного элемента равна ширине и высоте содержимого. В отличие от блочного элемента, мы не можем управлять шириной и высотой строчного элемента через CSS. Несколько строчных элементов будут стремиться уместиться на одной строке, насколько хватает ширины родителя. Если ширины родителя не хватает, то лишний текст строчного элемента переносится на следующую строку.

В рамках блочного контекста форматирования, в соответствии со спецификацией соприкасающиеся отступы «схлопываются».

Если первому дочернему элементу в блоке задан верхний отступ или последнему элементу — нижний, то эти отступы «выпадают» во внешний мир из своего родителя.

Выпадение отступов из родителя можно предотвратить несколькими способами, например:

- Задать родителю вертикальный внутренний отступ padding-top или padding-bottom.

- Задать родителю верхнюю или нижнюю рамку. Рамка может быть прозрачной, главное, чтобы она была.

- Задать родителю свойство overflow со значением, отличным от visible.

- Переопределить родителю свойство display на flow-root, либо на flex или grid (но в последнем случае его потомки будут раскладываться уже не по правилам нормального потока, а по правилам флексбоксов и грид-раскладки соответственно).

Выход из потока????????????????????

**css\_9. Что делает свойство float? Почему родитель элемента с float может схлопываться?**

**css\_10. Что делает свойство position?**

Static - значение по умолчанию. Статичное позиционирование. Любой элемент, добавленный на страницу, будет иметь это значение и будет расположен в нормальном потоке документа согласно контексту форматирования родительского элемента. Свойства left, right, top, bottom и z-index игнорируются.

Relative - элемент позиционируется так же, как и статичный, но его отображение может быть смещено относительно своего начального положения при помощи свойств left, right, top, bottom или inset. Это смещение чисто визуальное и не затрагивает положение соседних элементов, кроме случая, когда элемент выходит за границы родителя, имеющего возможность прокрутки.

На странице элемент будет занимать столько же места, как если бы он имел статичное позиционирование.

Absolute - элемент убирается из основного потока документа. То есть перестаёт влиять на положение окружающих элементов и на размер родителя.

Элемент позиционируется относительно ближайшего позиционированного предка. То есть браузер идёт вверх по дереву элементов и ищет ближайшего опорного родителя. И затем располагает наш элемент относительно этого родителя.

Если элемент был частью строчного контекста форматирования, он приобретает блочный контекст форматирования. К нему становится применима блочная модель.

Если элемент был блочным и занимал всю ширину своего родителя, то теперь его ширина будет определяться шириной контента.

Отступы элемента с position: absolute не схлопываются с отступами соседних элементов.

Fixed – используется, когда требуется позиционировать элемент не относительно родителя, а относительно окна браузера вне зависимости от вложенности. Элемент позиционируется относительно окна браузера, за исключением случаев, если один из родителей имеет значения свойств transform, perspective или filter, отличные от none. В этом случае блок становится опорным, и позиционирование будет производиться уже относительно него, а не окна браузера.

Элемент убирается из основного потока документа. То есть перестаёт влиять на положение окружающих элементов и на размер родителя.

Sticky - Это свойство применяется, когда нужно зафиксировать какой-либо элемент не сразу, а при прокрутке родителя до какого-то известного положения.

position: sticky; будет работать только тогда, когда на родительском элементе не установлено свойство overflow со значением hidden, scroll или auto.

Свойство position: sticky; не будет работать, если родительский элемент имеет меньшую высоту, чем «прилипающий» элемент. В таком случае, «прилипающий» элемент просто останется на своем месте.

Для корректной работы position: sticky; необходимо указать положение «прилипающего» элемента относительно окна просмотра с помощью свойств top, right, bottom или left. Например, top: 0; означает, что элемент будет «прилипать» к верхней части окна просмотра.

Элемент остаётся «приклеенным» во время прокрутки родителя до тех пор, пока не «встретит» противоположную границу своего родителя.

**css\_10.1 Какие элементы называют позиционированными?**

Элементы, у которых свойство position установлено в значение отличное от static.

Эти элементы считаются позиционированными, т.к. их расположение в документе управляется через свойства top, bottom, left, right, а также через свойство z-index.

Элемент со значением position: static не является позиционированным элементом. Элементы с любым другим значением свойства position создают внутри себя контекст наложения и становятся опорными.

**css\_10.2 Относительно чего происходит позиционирование элементов при разных значениях свойства position?**

Relative – относительно своего начального расположения.

Absolute - позиционируется относительно ближайшего позиционированного предка.

Fixed - позиционируется относительно окна браузера, за исключением случаев, если один из родителей имеет значения свойств transform, perspective или filter, отличные от none. В этом случае блок становится опорным, и позиционирование будет производиться уже относительно него, а не окна браузера.

Sticky -

**css\_11. Что делает свойство z-index?**

CSS-свойство z-index определяет положение позиционированного элемента и его дочерних элементов по оси z отвечающее за глубину, а также дает возможность создания локального контекста наложения. Перекрывающие элементы с большим значением z-index будут накладываться поверх элементов с меньшим z-index.

Значением свойства z-index может быть либо auto, либо целое число.

auto - элемент не будет создавать нового локального контекста наложения. Порядок наложения блока в текущим контексте наложения будет равен 0.

целое число - порядок наложения блока, заданный в виде целого отрицательного или положительного числа в текущем контексте наложения.

При этом элемент создаёт новый локальный контекст наложения. Таким образом значения z-index дочерних элементов внутри нового контекста наложения не будут сравниваться со значениями z-index элементов за пределами этого блока.

Чем выше значение z-index, тем выше элемент в *стопке* наложения.

Если z-index не указан, то блоки накладываются друг на друга в том порядке, в котором указаны в разметке. Чем ниже в разметке, тем выше в стопке. А если указан, то стопка блоков формируется согласно значению этого свойства.

Элементы могут сортироваться, находясь на одном уровне внутри контекста наложения. Элементы внутри блока никогда не могут быть выше или ниже своего родителя.

**css\_12. Контекст наложения.**

**css\_12.1 Что такое?**

Контекст наложения (stacking context) — это концепция трёхмерного расположения HTML-элементов вдоль оси Z по отношению к пользователю, находящемуся перед экраном.

В стандартной ситуации тот элемент, который находится ниже в HTML-разметке будет ближе к пользователю.

Допустим, если есть 2 блока:



И для наглядности сдвинуть второй блок чуть вверх и вправо:



Будет явно видно, что второй блок перекрывает собою первый, а значит находится выше в *стопке* элементов по оси *z*:



**css\_12.2 Как влияет на расположение элементов с заданным значением z-index?**

Z-index изменяет стандартные правила браузера и устанавливает свой порядок наложения элементов друг на друга.

**css\_12.3 Какие свойства создают новый контекст наложения?**

- корневой элемент (<html>),

- элемент позиционирован абсолютно (position: absolute) или относительно (position: relative) со свойством z-index, значение которого не auto,

- флекс-элемент со свойством z-index, значение которого не auto и чей родительский элемент имеет свойство display: flex или display: inline-flex,

- элементу задано свойство opacity со значением меньше 1,

- элементу задано свойство transform со значением не none,

- элементу задано свойство mix-blend-mode со значением не normal,

- элементу задано свойство filter со значением не none,

- элементу задано свойство isolation со значением isolate,

- элемент с position: fixed,

- элементу задано свойство will-change или аналогичный атрибут,

- элементу задано свойство -webkit-overflow-scrolling со значением touch.

Остальные элементы, не создающие собственный контекст наложения, используют родительский контекст.

**css\_13 Flex?**

**css\_13.1 Что такое flexbox?**

Это CSS-модуль для создания гибких и адаптивных макетов. Он помогает адаптировать веб-страницы под разные устройства и удобно располагать элементы. Это инструмент для построения сеток на странице. Свойство основано на концепции гибкого макета, где элементы могут растягиваться или сжиматься, чтобы использовать доступное пространство на странице наиболее эффективно.

**css\_13.2 Существует ли инлайновый flexbox?**

Да. Он реализуется с помощью значения display: inline-flex. В отличие от display: flex, который создает блочный flex-контейнер, inline-flex создаёт инлайновый flex-контейнер. Это означает, что инлайновый flex-контейнер занимает только столько места, сколько необходимо его содержимому.

**css\_13.3 Что такое flex-container и flex-items?**

Флекс-контейнер: элемент, к которому применяется свойство display: flex. Вложенные в него элементы подчиняются правилам раскладки флексов.

Флекс-элемент: непосредственные потомки флекс-контейнера.

**css\_13.4 Как выравнивать элементы с помощью flexbox?**

Выравнивать элементы можно по основной и поперечной осям, применяя свойства или к флекс-контейнеру, или к флес-элементу.

Основная ось: основная направляющая флекс-контейнера, вдоль которой располагаются флекс-элементы.

Поперечная (побочная, перпендикулярная) ось: ось, идущая перпендикулярно основной.

***justify-content*** – применяется к контейнеру и выравнивает флекс-элементы внутри флекс-контейнера по основной оси.

Значения:

start — элементы прижимаются к тому краю, откуда начинается чтение на том языке, на котором отображается сайт.

end — элементы прижимаются к краю, противоположному началу направления чтения на языке сайта.

flex-start — элементы прижимаются к краю, от которого начинается основная ось.

flex-end — элементы прижимаются к краю, у которого основная ось заканчивается.

left — элементы прижмутся к левому краю родителя.

right — элементы прижмутся к правому краю родителя.

center — элементы выстраиваются по центру родителя.

space-between — крайние элементы прижимаются к краям родителя, оставшиеся выстраиваются внутри контейнера равномерно, так, чтобы между ними были одинаковые отступы.

space-around — свободное пространство делится поровну между элементами и по половине от этой доли размещается по бокам от каждого элемента. Таким образом, между соседними элементами будет равное расстояние, а снаружи крайних элементов — по половине этого расстояния.

space-evenly — свободное место будет распределено так, чтобы расстояние между любыми двумя элементами было одинаковым и расстояние от крайних элементов до края было таким же.

***align-items***- применяется к контейнеру и выравнивает флекс-элементы внутри флекс-контейнера по поперечной оси.

Значения:

stretch (значение по умолчанию) — элементы растягиваются вдоль поперечной оси так, чтобы заполнить всего родителя.

flex-start или start — элементы выстраиваются у начала поперечной оси. Разница между ними лишь в том, что второе значение «уважает» направление чтения выбранного языка.

flex-end или end — элементы выстраиваются у конца поперечной оси. Разница между первым и вторым значениями аналогична предыдущему пункту.

center — элементы выстраиваются по центру поперечной оси.

baseline — элементы выравниваются по базовой линии текста. «Базовая линия» — baseline — воображаемая линия, проходящая по нижнему краю знаков шрифта.



***align-content*** - применяется к контейнеру и распределяет свободное пространство по поперечной оси между рядами флекс-элементов.  Если размер родителя по поперечной оси позволяет, то при помощи align-content можно распределять строчки элементов: по верхнему краю, по нижнему, по центру или равномерно. align-items выравнивает элементы внутри одной строки флекс-контейнера по поперечной оси, а align-content выравнивает все строки флекс-контейнера по поперечной оси, когда контейнер имеет больше, чем одну строку. Если контейнер имеет только одну строку, align-content не будет влиять на выравнивание.

align-items -Применяется к одной строке элементов.

align-content - Применяется ко всем строкам флекс-контейнера, если их больше одной, работает только, если flex-wrap установлено в wrap, то есть строки должны быть развернуты.

Значения:

stretch (значение по умолчанию) — ряды растягиваются одинаково, так, чтобы занять всё доступное пространство родителя.

flex-start / start — все ряды располагаются у начала поперечной оси. Первое значение не зависит от направления чтения текущего языка, в отличие от второго.

flex-end / end — все ряды располагаются у конца поперечной оси. end «уважает» направление чтения текущего языка.

center — ряды выравниваются по центру родителя.

space-between — первый ряд прижимается к началу поперечной оси, последний — к концу поперечной оси, а остальные располагаются так, чтобы свободное пространство было поделено на отступы между ними равномерно.

space-around — отступы у каждого ряда равнозначны отступам у любого другого ряда.

space-evenly — отступы между рядами и от краёв родителя одинаковые.

***align-self –*** применяется к флекс-элементу, и позволяет выровнять его по поперечной оси иначе, чем задано у родительского элемента, переопределяя значение свойства align-items. Если у какого-либо flex-элемента margin в поперечной оси выставлен в auto, то align-self игнорируется. Возможные значения совпадают с align-items.

**css\_13.5 Принципы работы flex-grow, flex-shrink. Каким образом рассчитывается занимаемое и свободное пространство при использовании этих свойств?**

flex-grow определяет как много свободного пространства во flex-контейнере должно быть назначено текущему элементу (flex grow factor — «коэффициент увеличения flex»). В значение принимает число. Чтобы посчитать flex grow, нужно:

1. Рассчитать свободное место - вычесть суммарную ширину элементов (с учетом flex-basis) из ширины контейнера.

2. Рассчитать долю свободного места - разделите свободное место на сумму значений flex-grow всех элементов.

3. Рассчитать итоговый размер - умножить долю свободного места на значение flex-grow конкретного элемента и прибавьте к его базовому размеру.

Пример:

Предположим, есть контейнер шириной 900px, и в нем три блока с базовыми размерами 200px каждый. Один блок имеет flex-grow: 1, второй flex-grow: 1, и третий flex-grow: 2.

Свободное место: 900px (контейнер) - (200px \* 3) (элементы) = 300px.

Доля свободного места: 300px / (1 + 1 + 2) = 300px / 4 = 75px.

Итоговые размеры:

Блок 1: 200px + (75px \* 1) = 275px.

Блок 2: 200px + (75px \* 1) = 275px.

Блок 3: 200px + (75px \* 2) = 350px.

flex-shrink — свойство CSS, которое определяет фактор сжатия flex-элемента(shrinking factor). Если в контейнере не хватает места для расположения всех элементов без изменения размеров, то свойство flex-shrink указывает, в каких пропорциях элемент будет уменьшаться.

Значение по умолчанию — 1, означающее, что все элементы будут сжиматься пропорционально. Значением может быть любое целое положительное число (включая 0).

Чтобы посчитать фактор сжатия необходимо:

1. Посчитать общий вес. Для этого нужно умножить значение flex-shrink каждого элемента на его размер (либо flex-basis, либо его width), и сложить получившиеся значения.

2. Посчитать сколько места не хватает. Для этого нужно вычесть сумму ширин всех элементов и ширины контейнера.

3. Посчитать сокращения для каждого элемента – пространство которого не хватает умножить на flex-shrink, затем результат умножить на ширину элемента, и результат разделить на общий вес.

4. Отнять получившиеся значение от исходной ширины элемента.

Например:

Есть два элемента, у каждого есть изначальная ширина 400 px и 800 px. Итого 1200 px. А ширина флекс-контейнера всего 800 px. Для первого элемента flex-shrink 2, для второго 1.

Общий вес: 400px \* 2 + 800px \* 1 = 1600px.

Не хватает места: 400px + 800px = 1200px – 800px(ширина контейнера) = 400px.

Сокращения для каждого элемента:

400px(которых не хватает) \* 2 \* 400px(ширина элемента) / 1600(общий вес) = 200px

400px(которых не хватает) \* 1 \* 800px(ширина элемента) / 1600(общий вес) = 200px

Отнять сокращение от исходной ширины элемента

400px – 200px = 200px.

800px – 200px = 600px.

Эта ширина будет у исходных элементов для ширины контейнера в 800px.

**css\_14. Что делает border и какие значения может принимать?**

Свойство border отвечает за отрисовку рамки - видимой границы блока. Само свойство border является шорткатом (короткой записью) и объединяет в себе значение для нескольких полных свойств: border-width, border-style, border-color. Значения можно писать в любом порядке, интерпретатор браузера разберётся. Главное, разделять их пробелами. Но общепринятым является порядок: ширина → стиль → цвет рамки. Можно рисовать фигуры рамками.

**css\_14.1 border-style**

Задает стиль рамки элемента. Свойство border-style является шорткатом для свойств: border-bottom-style, border-left-style, border-right-style, border-top-style. Может принимать значения:

* none — отключает рамку (значение по умолчанию).
* solid — сплошная рамка. Этот стиль используется чаще всего.
* dotted — рамка состоит из точек с пробелами между ними.
* dashed — рамка состоит из коротких чёрточек с пробелами между ними.
* double — рамка состоит из двух сплошных линий с небольшим нерегулируемым отступом между ними.

Есть еще и другие варианты рамок объемные с тенями.

**css\_14.2 border-color**

Управляет цветом рамки. Можно писать одно, два, три или четыре значения, разделяя их пробелами. В зависимости от этого получится разный результат:

Одно значение - цвет рамки со всех сторон одинаковый.

Два значения - первое устанавливает цвет рамки сверху и снизу, второе — слева и справа.

Три значения - первое значение устанавливает цвет рамки сверху, второе — слева и справа, а третье — снизу.

Четыре значения — четыре значения, устанавливает для каждой стороны свой цвет, поочерёдно: для верхней, правой, нижней и левой рамки.

Значение по умолчанию соответствует значению color у элемента.

Разрешается любое доступное обозначение цвета в вебе, а также ключевые слова transparent (для прозрачной рамки) или inherit (для наследования значения родителя).

**css\_14.3 border-width**

Управляет толщиной рамки элемента. Для того чтобы увидеть работу свойства, необходимо помимо толщины рамки добавить ещё и значение свойства border-style.

Значением может быть число с любыми единицами измерения, доступными в вебе, кроме процентов. Разрешается указывать одно, два, три или четыре значения. Для задания толщины можно использовать переменные - thin, medium или thick, значение переменных определяется браузерами.

**css\_14.4 border-radius**

Задаёт радиус закругления каждого из углов элемента. Значение в процентах будет высчитываться от размеров элемента: горизонтальные размеры будут высчитываться от ширины элемента, а вертикальные — соответственно, от высоты. Это свойство является шорткатом для четырёх свойств border-top-left-radius, border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius.

Свойство border-radius может быть задано как:

одно, два, три или четыре значения - для задания обычного радиуса углов.

одна, две, три или четыре пары значений разделённые "/". Используется для задания эллиптического скругления.

**css\_14.5 Можно ли задать стили для каждой рамки отдельно (например, отдельно для левой и для правой рамок)?**

Можно! При помощи следующих свойств: border-top, border-right, border-bottom, border-left. Все четыре рамки могут быть разными по толщине, цвету и стилю.

**css\_15 Внешние рамки (outline).**

**css\_15.1 В чем отличие от обычных рамок?**

**css\_15.2 Что делают и какие значения могут принимать свойства:**

**outline-style** - задаёт стиль обводки. Значения такие же, как у свойства border-style.

**outline-color** - задаёт цвет обводки элемента. Принимает любое доступное значение цвета в вебе, включая ключевые слова transparent, currentColor и invert — ключевое слово, устанавливающее противоположный цвету фона элемента цвет обводки. Удобно для достижения максимальной контрастности.

**outline-width** - указывает какой толщины должна быть обводка. Для того чтобы свойство заработало, нужно написать свойство outline-style и задать любое значение, кроме none. В отличие от border-width, нельзя устанавливать границу для каждой стороны, свойство применяется одновременно для всех сторон. Можно использовать для указания толщины обводки любые единицы измерения или ключевые слова.

**outline-offset** - устанавливает расстояние между фактическим краем элемента и внутренним краем обводки, заданной при помощи outline. В качестве значения можно указать положительное или отрицательное число в любых доступных единицах измерения. Если значение отрицательное, то обводка будет отображаться внутри элемента. Если свойство outline-offset не прописано или ему задано значение 0, то обводка будет располагаться вплотную к рамке (свойство border) или границе элемента. Расстояние между границей элемента или его рамкой и обводкой прозрачное. Нет возможности его закрасить.

**outline** - шорткат — короткое свойство для записи сразу нескольких значений. Внутри себя оно содержит свойства outline-width, outline-style и outline-color. Можно задать одно, два или три значения. Без указания outline-style обводка не будет отображаться. Исключение составляют элементы input, для которых браузеры определяют стандартный стиль.

Если задано только одно значение, то цвет будет совпадать с цветом текста (подражание ключевому слову currentColor), а размер будет соответствовать ключевому слову medium (что, в свою очередь, равно 3 пикселям). Порядок следования значений не имеет значения.

**css\_16 Тени.**

**css\_16.1 Что делает свойство text-shadow? Как им пользоваться?**

Свойство даёт возможность добавить тень буквам в тексте. Можно задавать несколько теней для одного элемента, перечисляя их через запятую. Если определены больше, чем одна тень, тени накладываются друг на друга в порядке обратном порядку их определения, что значит, первая тень окажется поверх последующих.

Принимает:

Два или три значения размера с единицами измерения + цвет.

Если задано только два значения, то они интерпретируются как смещение по осям x и y. Смещения по осям *x* и *y* — обязательные значения для тени. Могут принимать любые числовые значения, в том числе отрицательные. Значение по умолчанию равно 0 для обеих осей. Если первое значение положительное, то тень будет справа от буквы, если отрицательное — слева. Если второе значение положительное, то тень будет снизу, если отрицательное — сверху.

Если задано и третье значение тоже, то оно отвечает за размытие тени. Радиус размытия — необязательное, положительное числовое значение с единицами измерения. По умолчанию значение 0, что делает его указание необязательным. Если не указываете его или пишете 0, то край тени будет резким, без размытия. Чем больше значение, тем шире область размытия и тем светлее сама тень.

Цвет тени в любом доступном формате. Может быть указан до числовых значений размеров, так и после. Но не между значениями. А то браузер запутается. Цвет — вроде бы опциональное, но на самом деле обязательное значение цвета тени. Почему? Если не указывать цвет, то решение остаётся за браузером. Как правило, браузер возьмёт значение свойства color текста, которому прописываете тень. Но Safari отрисует прозрачную тень. Если нужен цвет тени, совпадающий с цветом текста, можно указать явно ключевым словом currentColor.

text-shadow: 5px 5px #2e9aff;



**css\_16.2 Что делает свойство box-shadow? Как им пользоваться?**

Добавляет тень к элементу. Тень будет совпадать с формой блока. Если вы сделали круглый блок при помощи, например, border-radius, то тень тоже будет круглой. Если не менять форму элемента, то тень будет прямоугольной.

Можно задавать несколько теней для одного элемента, перечисляя их через запятую.

Принимает:

Два, три или четыре значения размера с единицами измерения + цвет + inset.

Два значения расшифровываются как смещение по оси x и по оси y - обязательные.

Если задано третье значение, то оно интерпретируется как радиус размытия.

Если задано четвёртое значение, то оно отвечает за радиус распространения. Радиус распространения — необязательное, любое числовое значение с единицами измерения. По умолчанию равно 0, размеры тени совпадают с размерами элемента. Если указано отрицательное значение, то тень будет меньше, если положительное, то тень будет больше.

Дополнительно (необязательно) можно указать ключевое слово inset, которое превратит тень из внешней во внутреннюю.

Не обязательно, цвет тени в любом доступном формате цвета(но по факту обязательно также как и text-shadow).

Порядок наложения нескольких теней такой же, что и у текстовых теней: первая указанная тень будет поверх остальных.

**css\_16.3 Как сделать так, чтобы у одного элемента было несколько теней?**

Перечислить тени через запятую.

**css\_16.4 Есть ли возможность задавать параметры теней отдельно (например, задать отдельно цвет или размер тени)?**

**css\_17 Фоны. Что делают и какие значения могут принимать свойства:**

**background-image**

Устанавливает одно или несколько фоновых изображений для элемента. Изображения рисуются в слоях контекстов наложения одно поверх другого. Первый слой выводится так, чтобы он был ближе всего к пользователю.

**React**

**rct\_1. Что такое React?**

Это JavaScript-библиотека для создания пользовательских интерфейсов. React использует компонентную архитектуру, декларативный стиль программирования и быстрые, удобные для разработчика абстракции DOM, iOS, Android и др. React обеспечивает высокую производительность благодаря виртуальной модели DOM. Плюс огромное сообщество, которое предоставляет множество различных готовых компонентов.

**rct\_1.1. Что такое ReactDOM?**

Библиотека, которая содержит методы для рендеринга DOM, а также рендеринга на стороне сервера.

**rct\_2. Что такое декларативное UI программирование? Чем оно отличается от императивного подхода?** 

Императивное программирование — оно же «приказывающее» или «повелительное» — это парадигма (стиль написания исходного кода компьютерной программы), в которой описываются последовательные команды, которые должен совершить процессор, чтобы получить необходимый нам результат. Т.е. это такой стиль программирования, при котором описывается, как добиться желаемого результата.

Языки: C, C++;

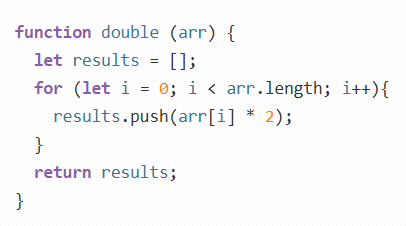
Декларативное программирование — это парадигма (стиль написания исходного кода компьютерной программы), программирования, в которой описывается ожидаемый результат, а не способ его получения.

Языки: SQL, HTML.

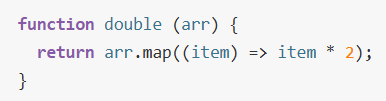
Неважно, как будут выбраны пользователи из Мексики и как браузер распарсит article. Важно, что мы получим мексиканских пользователей и новый header и paragraph на сайте, не углубляясь в инструкции.

Например: Дан массив чисел, написать функцию, которая вернет массив, где каждое число -> [1, 2, 3] -> [2, 3, 6]

Императивный стиль:



Декларативный стиль:



**rct\_3. Что такое React Element? Как его можно создать? Из чего он состоит?** 

React-элемент — это не то, что будет на экране, а описание того, что будет на экране. React-элемент описывает узел DOM в виде javascript-объекта. Объект состоит из свойств type(тип тега), props(атрибуты), children(дочерние узлы), и др.

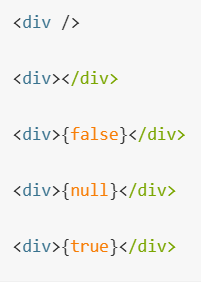
Создать можно:

* React.createElement(type, props, children)
* Используя синтаксис JSX.

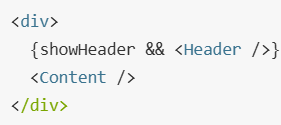
Если коротко: элементы являются экземплярами компонентов, которым могут передаваться свойства.

**rct\_3.1 Что будет, если передать булевые, null или undefined значения в children?**

Они игнорируются. Их можно использовать в качестве потомков, но данные значения не отображаются при рендеринге. Следующие JSX выражения будут отображены одинаково:



Вместе с условными выражениями это можно использовать для условного отображения элементов React:



**rct\_3.2 Что такое React Component?** 

React-компонент — это функция или класс, который принимает входные данные (опционально) и возвращает React-элемент — то есть, javascript-объект. Пользовательский интерфейс создается с помощью комонентов.

**rct\_4 Что такое JSX и зачем он нужен?**

**JSX представляет собой синтаксический сахар на базе JavaScript, позволяющий**

**записывать элементы React на JavaScript в HTML-подобной записи с <>.**

JavaScript XML (JSX) — это синтаксическое расширение JavaScript предоставляющее «синтаксический сахар» для вызовов функций и построения объектов, особенно React.createElement() (иначе говоря, он упрощает процесс написания кода, сохраняя его функциональность). JSX ускоряет создание элементов React, делая их более компактными, а также упрощает их чтение. JSX создан только для разработчиков. Сам по себе он никак не ускоряет и не улучшает веб-приложения. JSX преобразуется в тот же код, который получается и без JSX.

Основные плюсы:

- Улучшенное взаимодействие с разработчиком (DX, Developer Experience) — код проще читать благодаря XML-подобному синтаксису, который хорошо подходит для написания вложенных декларативных структур.

- Улучшенные сообщения об ошибках — React по умолчанию предполагает использование JSX и выводит полезные сообщения об ошибках. Если JSX не используется, эти сообщения будут запутывать разработчика, так как в них будет упоминаться не тот синтаксис, который он реально использует.

- Ускорение кода — при преобразовании JSX в JavaScript транспилятор одновременно оптимизирует код, благодаря чему итоговый код JavaScript исполняется быстрее, чем если бы он был введен вручную.

- Повышение эффективности работы команд — неопытным разработчикам (например, веб-дизайнерам) проще менять код, потому что JSX напоминает разметку HTML, которая им уже знакома.

- Снижение количества синтаксических ошибок — разработчики набирают на клавиатуре меньше кода, а это значит, что они сделают меньше ошибок.

Процесс преобразования JSX в JavaScript, называется транспиляция. Транспиляция, а не компиляцией, потому что код транслируется с одного исходного языка (JSX) на другой исходный язык (JavaScript). В свою очередь, полученный код JavaScript интерпретируется «реальным» компилятором, который выполняет код. Транспиляция — всего лишь преобразование синтаксиса, а не интерпретация кода.

При выполнении приложения React браузер видит только команды React.createElement, необходимые для генерирования нужной структуры. JSX существует только в редакторе. Транспилятор преобразует файлы, содержащие JSX, в «чистый» JavaScript с многочисленными вызовами React.createElement().

Процесс транспиляции заключается в следующем: исходный код переписывается, чтобы получить такие же результаты, но уже с помощью синтаксиса, понимаемого устаревшими браузерами. Любой новый синтаксис для браузера, который его не поддерживает, просто недействителен и выдает ошибку разбора.

**rct\_x Что такое stateful и stateless компоненты?**

**rct\_x.x Что делает хук useState? Как обновить состояние?**

Позволяет добавлять состояние в функциональные компоненты. React хранит это состояние между рендерами. Вызов useState возвращает массив с двумя элементами - текущее значение состояния и функцию для его обновления. Единственный аргумент useState — это начальное состояние. Чтобы обновить состояние нужно вызвать функцию обновления состояния с новым значением или коллбэком. После этого React сделает повторный рендер, в котором использует уже новое значение.