**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\_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.

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

Управляет толщиной рамки элемента. 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.

**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.