**1.Понятие веб-технологий. Чем веб отличается от других сетевых служб?**

Под «веб-технологиями» обычно подразумеваются языки разметки, языки программирования, базы данных и другие технологии, которые позволяют создавать веб-сайты, приложения и магазины. Есть два типа веб-технологий. Первый, front-end, - отвечает за внешний вид сайта, и виден в браузере пользователя. Технологии включают HTML, CSS, JavaScript. Второй - back-end, работает на сервере и используется для обработки данных. Обычно невидимо для пользователя, видны только введенные данные или действия, выполненные на веб-сайте, и результат этих данных или действий. Популярным языком программирования является PHP, созданный для использования на веб-сайтах. Однако есть более универсальные языки - Python, Java, C #, C ++.

Сервер - это устройство, которое предоставляет услуги другим устройствам. При создании веб-сайтов обычно используется серверы внешнего поставщика услуг. Когда пользователь вводит адрес веб-сайта, устройство отправляет на сервер, на котором размещена эта страница, запрос на ее загрузку. Если связь по линии клиент-сервер установлена, сервер отправит код веб-сайта на компьютер. Веб-браузер обработает его, и через некоторое время появится готовый сайт.

Сетевая служба - совокупность серверной и клиентской частей ОС, предоставляющих доступ к конкретному типу ресурса компьютера через сеть. Сетевая служба предоставляет пользователям сети некоторый набор услуг. Эти услуги иногда называют также сетевым сервисом. Веб-служба, веб-сервис — идентифицируемая уникальным веб-адресом (URL-адресом) программная система со стандартизированными интерфейсами.

Классификация сетевых сервисов

Наиболее известными среди сетевых сервисов являются (сервисы 1 поколения): •электронная почта (E-mail), обеспечивающая возможность обмена сообщениями одного человека с одним или несколькими абонентами; •сервис FTP – система файловых архивов, обеспечивающая хранение и пересылку файлов различных типов; •World Wide Web (WWW) – гипертекстовая (гипермедиа) система, предназначенная для интеграции различных сетевых ресурсов в единое информационное пространство; •сервис DNS, или система доменных имен, обеспечивающий возможность использования для адресации узлов сети мнемонических имен вместо числовых адресов; •сервис IRC, предназначенный для поддержки текстового общения в реальном времени (chat); и др.

Преимущества: 1)Веб-службы обеспечивают взаимодействие программных систем независимо от платформы. Windows-C#-клиент может обмениваться данными с Java-сервером, работающим под Linux. 2)Веб-службы основаны на базе открытых стандартов и протоколов. Достигается простота разработки и отладки веб-служб благодаря использованию XML. 3)Использование интернет-протокола обеспечивает HTTP-взаимодействие программных систем через межсетевой экран. Это значительное преимущество, по сравнению с CORBA, DCOM или Java RMI. С другой стороны, веб-службы не привязаны намертво к HTTP — могут использоваться и другие протоколы.

Недостатки: 1)Меньшая производительность и больший размер сетевого трафика по сравнению с технологиями RMI, CORBA, DCOM за счёт использования текстовых XML-сообщений. Однако на некоторых веб-серверах возможна настройка сжатия сетевого трафика.2)Аспекты безопасности. Ответственные веб-службы должны использовать кодирование, возможно — требовать аутентификации пользователя. Достаточно ли здесь применения HTTPS, или предпочтительны такие решения, как XML Signature, XML Encryption или SAML — должно быть решено разработчиком.

WEB и HTTP сервисы очень сходны между собой, но все же в них есть кардинальные отличия. Сходны они тем, что: Предназначены для доступа к базе данных снаружи, Работают посредством веб-технологий, поверх протокола TCP, Оба поддерживают технологии XML и JSON.

Различия: веб🡪http; Сервисно-ориентированный🡪ресурсно-ориентированный

Есть описание процедур (wsdl) 🡪нет описания (только документация); Есть проверка типов данных (xdto) 🡪нет проверки типов (есть заголовки, параметры и тело запроса); Преимущественно xml🡪преимущественно json

**3.История развития веб-технологий.**

**Развитие HTML:** Первая версия языка разметки гипертекста была разработана в 1991 году сотрудниками CERN. За его основу взят SGML, стандартный язык структурной разметки. Разметка была логической, то есть не несла никакой информации о внешнем виде документа, лишь указывала границы и соподчинение его составных частей. HTML 1.0 не был стандартизирован, так как на данный момент существовал только один браузер, Mosaic. В 1995 году появилась первая спецификация для языка гипертекстовой разметки, HTML 2.0. В ней появилась возможность вставлять рисунки, гиперссылки и формы отправки информации на сервер. С появлением спецификации HTML 3.2 начался подъем в web-дизайне. Были реализованы таблицы, разметки математических формул, обтекание текста вокруг объектов. Самым важным обновлением в данной версии HTML была поддержка каскадных таблиц стиля CSS, позволявших производить графическое описание документа отдельно. Версия HTML 4.01 была стандартизирована 24 декабря 1999 года. Она отличается законченностью и полнотой, поддерживает усовершенствованную версию CSS. Появился язык **XML**, сочетающий в себе простоту HTML, логику разметки SGML и удовлетворяющий требованиям Интернета. Он используется для создания многих web-приложений, в том числе и для мобильных устройств.

**Развитие web-серверов:** Изначально, основой Всемирной паутины были web-сервера CERN httpd, написанные Тимом Бернерсом-Ли на языке программирования Си. Сервер NCSA httpd появился после CERN, когда возникла потребность в небольшом и быстром web-сервере. В 1995 году Брайан Белендорф объединил эти патчи и создал первую версию сервера **Apache**, который по настоящее время занимает лидирующую позицию по популярности. Одновременно с появлением Apache, компания Microsoft выпустила коммерческий web-сервер Internet Information Server (IIS), как надстройку для операционной системы Windows NT 3.51. В этой версии отсутствовали многие возможности, но имелись стандартные веб-службы, FTP и Gopher. В 2003 году вышел web-сервер IIS6 в составе операционной системы Windows Server 2003. В данной версии произошли значительные изменения: добавлена поддержка программной платформы .NET и применен новый подход с точки зрения безопасности.

**Развитие языков web-программирования:** Web-программирование – это раздел программирования, ориентированный на разработку динамических web-приложений. Языки web-программирования делятся на серверные и клиентские. Для связи с сервером используется интерфейс CGI. В 1994 году. PHP/FI был размещен в Сети для всеобщего использования, и началось его повсеместное распространение. К концу 1997 года РНР использовался более чем на пятидесяти тысячах сайтов. Так как исходный код интерпретатора был открыт, то энтузиасты стали заниматься его доработкой, и летом 1998 года появился РНРЗ – разработка Зива Сураски и Энди Гутманса. РНРЗ был создан практически «с нуля», так как его авторы сочли код предыдущих версий недостаточно эффективным.

Разработка первого клиентского языка велась в 1992-1995 годах компанией Nombas. Этот язык, названный Cmm («Си-минус-минус») не получил широкого распространения, так как сценарии, написанные на нем работали только в 16-битовом Netscape Navigator под управлением Windows. В это же время группа разработчиков компании Netscape под руководством Брендона Айха создала язык LiveScript, который впоследствии был переименован в JavaScript и стандартизован Европейской компьютерной ассоциацией. В 1996 году компания Microsoft выпустила технологию ASP, позволяющую подключать программные модули во время процесса формирования web-страницы. Относительная популярность ASP основана на простоте используемых при этом языков Visual Basic Script и JScript. Разработчики могут писать код для ASP.NET, используя практически любые языки программирования, входящие в комплект программной платформы .NET (C#, Visual Basic.NET, и JScript .NET). В 2005 году была разработана технология AJAX. Это новый подход к построению интерактивных пользовательских интерфейсов web-приложений, заключающийся в «фоновом» обмене данными браузера с сервером.

**2.Основные сетевые протоколы, использующ в вебе.**

**Понятие** Сетевой протокол — набор правил, определяющий принципы взаимодействия устройств в сети. Чтобы отправка и получение информации прошли успешно, все устройства-участники процесса должны принимать условия протокола и следовать им. Для взаимодействия протоколов между собой существует модель OSI. Модель OSI позволяет разным системам связи коммуницировать между собой по общепринятым стандартам.

## TCP/IP — модель, на которой работает сеть Интернет

|  |  |
| --- | --- |
| Модель OSI | TCP/IP |
| Прикладной уровень | Прикладной уровень |
| Уровень представления |
| Сеансовый уровень |
| Транспортный уровень | Транспортный уровень |
| Сетевой уровень | Межсетевой уровень |
| Канальный уровень | Канальный уровень |
| Физический уровень |

Модель TCP/IP помогает понять принцип работы и взаимодействия узлов в сети Интернет. Ее название включает в себя два основных протокола, на которых построен интернет. TCP/IP или протокол управления передачей (данных)/интернет-протокол.

Протоколы транспортного уровня: краткое описание

Internet Protocol (IP) — это наиболее простой протокол, объединивший отдельные компьютеры в глобальную сеть. Главная задача - маршрутизация дейтаграмм — определение маршрута следования пакетов по узлам сети. Минусы протокола - низкая надежность. Он не определяет факт передачи пакета и не контролирует целостность данных. IP просто осуществляет пересылку пакета, где необходимо определить, на какой порт отправить пакет. Для этого протокол имеет свою систему адресации. В качестве адресов выступает 32-битные (IPv4) или 128-битные (IPv6) адреса. Недостаток протокола — в сложности сетевого администрирования. Один из основных протоколов, который работает поверх IP, — это протокол TCP, из-за чего его часто обозначают как TCP/IP. TCP помогает устройствам в сети обмениваться сообщениями. Свойства протокола TCP: **Система нумерации сегментов** **Управление потоком**. **Контроль ошибок**. **Порт источника и порт назначения.** SSH использует 22й порт, HTTP — 80.

В отличие от протокола ТСР User Datagram Protocol обеспечивает передачу данных без получения подтверждения от пользователя о результате действия. Достигается большая скорость работы и передачи данных в ущерб надежности и безопасности. UDP работает путем сбора данных в UDP-пакете и добавления в пакет собственной информации заголовка. Тот включает четыре поля: номер порта источника, номер порта назначения, длина заголовка и контрольная сумма блока.

Протокол SCTP обеспечивает надежную последовательную передачу данных. Поддерживает многоадресное соединение, когда один или оба конечных узла могут состоять из более чем одного IP-адреса. Это обеспечивает прозрачное переключение между резервными сетевыми путями. SCTP работает поверх бесконтактной пакетной сети (IP) и поддерживает передачу данных в случаях с одним или несколькими IP.

Real-time Transfer Protocol — это протокол, который используется при передаче потокового аудио и видео. RTP применяется в совокупности с протоколом управления RTCP. Когда RTP транслирует медиа, RTCP применяется при анализе статистик QoS и обеспечивает синхронизацию разных потоков.

## Протоколы межсетевого уровня: краткое описание

ICMP — протокол управляющих сообщений в сети. Задача протокола — диагностика проблем при взаимодействии устройств. Основная цель ICMP — сообщать об ошибках. Как и в случае UDP, протокол ICMP можно использовать для сетевых атак.

Open Shortest Path First используется для поиска наилучшего пути между исходным и конечным маршрутизатором. На основе собранной информации он вычислит наилучший кратчайший путь к каждой доступной подсети/сети, используя алгоритм под названием Shortest Path First (SFP).

## Протоколы прикладного уровня: краткое описание

FTP — это клиент-серверный протокол, который использует два канала для передачи данных: командный, управляющий процессом передачи, и транспортный, непосредственно передающий информацию. FTP-сессии работают в двух режимах: 1)При активном режиме сервер после инициализации, путем вызова командного канала, открывает транспортный канал и начинает передачу данных. 2)При пассивном режиме сервер при помощи командного канала отправляет клиенту данные, требующиеся для открытия канала передачи данных.

Domain Name System преобразует домены в IP-адреса, чтобы сделать возможной загрузку интернет-ресурса через браузер.

HTTP является основой интернета и используется для загрузки веб-страниц с использованием гипертекстовых ссылок. Относится к прикладным протоколам и работает поверх других уровней стека сетевых протоколов. Каждый HTTP-запрос включает в себя ряд закодированных данных, содержащих различную информацию: версию HTTP, URL-адрес, метод HTTP-запроса, заголовок — он передает основную информацию о запросе и содержит пары ключ-значение, тело запроса. После получения запроса сервер должен дать ответ. В его стандартную структуру входят: код состояния, заголовок и тело ответа. Код состояния HTTP-запроса — это трехзначные коды, которые, как правило, указывают на успешность его выполнения. 1xx\* Информация, 2хх Успешность выполнения, 3хх Перенаправление, 4xx Ошибка клиента, 5xx Ошибка сервера.

SSH — это защищенный протокол, который используется как основное средство подключения к серверам. С помощью него при подключении к серверу пользователь входит в существующую учетную запись, где выполняются все отправленные команды. Данное соединение реализовано по схеме «клиент-сервер».

**4.Клиентская часть веб-приложений. Назначение, технологии, схема работы.**

**Клиентская часть веб-приложения** (Front-end) – интерактивная часть программы, исполняемая в веб-браузере на компьютере, смартфоне или планшете пользователя. Она реализует пользовательский интерфейс веб-приложения и загружается на устройства в виде динамических веб-страниц. Веб-приложения запускаются на любых устройства и операционных системах, где есть интернет-браузеры.

Фронтенд-разработчик работает со следующими компонентами:

- Фреймворки (React.js, AngularJS и т. д.). Платформы, дающие фронтенд-разработчику основу для создания веб-приложений. В них присутствуют определенные и реализованные функции с классами. В фреймворках можно добавлять свой код к предложенному.

- Cascading Style Sheets (CSS). Язык стилизации внешнего вида веб-страницы. При помощи CSS клиент (браузер) «понимает», каким именно образом нужно отображать графические элементы на сайте. При помощи CSS задаются цвета, шрифты, расположения блоков и т. д. CSS-код помогает адаптировать веб-приложение, чтобы оно одинаково отображалось на разных устройствах.

- HyperText Markup Language (HTML). Язык разметки всех элементов страницы, позволяющий также прописать особенности их взаимодействия друг с другом.

- JavaScript. Язык, способный «оживить» веб-страницы. Цели создания таких скриптов — отклики на действия посетителей, обработка перемещения курсора мыши, нажатий на клавиши, отправленных запросов, загрузки данных без перезагрузки самой страницы и т. д.

- Сопутствующие системы. Понимание работы инструментов для контроля версий (GitHub, Git, CVS и т. д.), минимальное владение графическими редакторами по типу Photoshop, Illustrator, знание английского языка.

## Варианты взаимодействия frontend и backend

Отправка HTTP-запроса на сервер, где она ищется, встраивается в шаблон и возвращается пользователю в читаемом виде HTML-страницы.

Есть еще инструментарий под названием AJAX. В таком случае запрос отправляется при помощи JavaScript, который подключен в веб-обозревателе. Ответ возвращается в XML или JSON, а с чтением этих форматов отлично справляется JS.

Существуют одностраничные сайты, загружающие данные без предварительного обновления страницы. Осуществляется данная операция при помощи AJAX или фреймворков Angular и Ember.

Подключаемые библиотеки Ember или React предназначены для использования приложения одновременно на сервере и в клиентской части. Две рассматриваемые сферы связываются через AJAX и HTML-код с обработкой на сервере.

**5.Серверная часть веб-приложений. Назначение, технологии, схема работы.**

Back-end разработка фокусируется на серверных аспектах веб-сайта или веб-приложения. Этот тип разработки связан с архитектурой веб-сайта, написанием сценариев и взаимодействием с базами данных. Внутренний код обеспечивает связь между браузерами и информацией из баз данных. Back-end разработчики сосредотачиваются на том, как функционирует веб-сайт, они могут работать с API, кодом, который взаимодействует с базами данных, библиотеками, архитектурой данных и т. Д.

### Серверное программирование и языки сценариев

Существует два разных типа языков программирования серверной части: функциональные языки и объектно-ориентированные (ООП) языки. Первые используют декларативный синтаксис и делают упор на выражение. Они создают чистые функции и избегают общих состояний и изменяемых данных. Объектно-ориентированные языки используют классы и объекты. Это позволяет структурировать программы в простые многократно используемые фрагменты кода. Можно использовать фрагменты кода для создания отдельных экземпляров объектов. Популярные серверные языки включают: Python, Java, JavaScript, .NET, PHP, R, Haskell, SQL, Clojure

Фреймворки бэкэнда упрощают и делают более удобным создание согласованных и стабильных бэкэндов. Популярные фреймворки включают: Django, Ruby on Rails, Flask, Asp .NET, Spring Boot

Бэкэнд веб-сайта состоит из серверов, приложений и баз данных. Базы данных важны, потому что они позволяют веб-сайтам и приложениям обрабатывать пользовательские данные. Системы управления базами данных (СУБД) действуют как интерфейс между базой данных и ее пользователями. СУБД позволяет пользователям получать доступ, обновлять и управлять информацией. Есть два основных типа баз данных: SQL и NoSQL. Базы данных SQL структурированы, а базы данных NoSQL не структурированы.

**Схема работы Backend и общая логика сайта вообще сводятся к трем пунктам: 1)**Получение-обработка данных от пользователя (ввод поискового запроса). 2) Обработка данных на сервере (поиск данных в базе, любые выборки). 3)Получение ответа от сервера и перевод информации в удобочитаемый вид (вывод результата).

**backend-разработчику нужно: 1)**Поддерживать веб-ресурс, бизнес-логику проекта и бэкэнд-системы в целом. 2)Создавать программный интерфейс веб-ресурса для его взаимодействия с другими платформами. 4)Создавать единую библиотеку элементов для фронтенда и дизайнеров. 5) Адаптировать систему под разные платформы с помощю CSS3 и HTML5. 6) Оптимизировать существующие веб-приложения для повышения быстродействия.

## Варианты взаимодействия frontend и backend

1. Отправка HTTP-запроса на сервер, где он ищется, встраивается в шаблон и возвращается пользователю в читаемом виде HTML-страницы.
2. AJAX. В таком случае запрос отправляется при помощи JS. Ответ возвращается в XML или JSON, а с чтением этих форматов справляется JS.
3. Одностраничные сайты, загружающие данные без предварительного обновления страницы. Осуществляется данная операция при помощи AJAX или фреймворков Angular и Ember.

Подключаемые библиотеки Ember или React предназначены для использования приложения одновременно на сервере и в клиентской части. Эти сферы связываются через AJAX и HTML-код с обработкой на сервере.

**8.URL и URI. Понятие, различия, структура, примеры, использование.**

URL🡪URL-адрес — это идентификатор, который сообщает вашему компьютеру о доступе к ресурсу на веб-сервере. Унифицированный указатель ресурса или URL является подмножеством URI, что означает, что все URL также являются URI. URL предоставляет схему и местоположение ресурса. Ресурс URL находится в сети, что означает, что к нему можно получить доступ из локальной сети, глобальной сети, сети Интернет. URL-адреса, обычно используются в веб-браузерах и приложениях для указания удаленного ресурса, такого как веб-страница, API и т. д.

URL содержит следующую информацию: Протокол, который используется для доступа к ресурсу – http, https, ftp; Расположение сервера с использованием IP-адреса или имени домена - например, wiki.merionet.ru - это имя домена. https://192.168.1.17 - здесь ресурс расположен по указанному IP-адресу; Номер порта на сервере. Например, http://localhost: 8080, где 8080 - это порт; Точное местоположение в структуре каталогов сервера. Например - https://wiki.merionet.ru/ip-telephoniya/ - это точное местоположение, если пользователь хочет перейти в раздел про телефонию на сайте.; Необязательный идентификатор фрагмента.

Синтаксис: [protocol]://www.[domain\_name]:[port 80]/[path or exaction resource location]?[query]# [fragment]

URI🡪 Унифицированный идентификатор ресурса — это идентификатор ресурса. URI может относиться к веб-странице, книге или документу. URL-адреса представляют собой тип URI, поскольку они позволяют идентифицировать определённый ресурс в Интернете. Но URI не ограничиваются веб-разработкой. URI является наиболее общим идентификатором ресурса, где его можно использовать для различных ресурсов, таких как веб-страница, QR-код и т. д.

URI содержит в себе следующие части: **Схема (scheme)** - показывает на то, как обращаться к ресурсу, чаще всего это сетевой протокол (http, ftp, ldap); **Иерархическая часть (hier-part)** - данные, необходимые для идентификации ресурса (например, адрес сайта); **Запрос (query)** - необязательные дополнительные данные ресурса (например, поисковой запрос); **Фрагмент (fragment)** – необязательный компонент для идентификации вторичного ресурса ресурса (например, место на странице) Общий синтаксис URI выглядит так: URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]

### URI или URL

* URL – это локатор, тогда как URI является идентификатором. URL специфичен для веб-ресурсов и есть универсальный URI, который содержит URL. URL должен иметь протокол, такой как http, ftp, tel и т. Д., тогда как URI не должен иметь протокол. И URL, и URI могут использоваться для относительных и абсолютных ссылок. И URL, и URI могут принимать параметры запроса.  И URL, и URI могут иметь фрагментированные идентификаторы. Согласно определению W3C, URL и URI – это одно и то же, и они могут быть взаимозаменяемыми.

|  |  |
| --- | --- |
| **URI** | **URL** |
| Схема может быть любой - протокол, имя, спецификация и тд | Схема всегда протокол, http, https, ftp, LDAP и так далее |
| цель - идентифицировать ресурс, отличить его от других, используя местоположение или имя. | Основная цель - получить адрес или местоположение ресурса. |
| Пример: contact: +1 883-345-1111, urn:isbn:1234567890 | Пример: https://wiki.merionet.ru/servernye-resheniya/36/vse-chto-vam-nuzhno-znat-pro-devops/?f=0 |
| Используется в файлах HTML, XML и библиотек тегов для идентиф ресурсов и двоич файлов. | URL используется для поиска только веб-страниц |

**96.ORM. Понятие, назначение, использование, примеры.**

Объектно-реляционное отображение (ORM) - это метод, который позволяет запрашивать и манипулировать данными из базы данных с использованием объектно-ориентированной парадигмы. Говоря об ORM, большинство людей ссылаются на библиотеку, которая реализует технику объектно-реляционного сопоставления, отсюда и фраза "ORM".

Библиотека ORM — это совершенно обычная библиотека, написанная на выбранном вами языке, которая инкапсулирует код, необходимый для манипулирования данными, поэтому вы больше не используете SQL; вы взаимодействуете непосредственно с объектом на том же языке, который используете.

Например, вот совершенно воображаемый случай с псевдоязыком:

У вас есть класс книг, вы хотите получить все книги, автором которых является "Линус". Вручную, вы бы сделали что-то вроде этого:

book\_list = new List(); sql = "SELECT book FROM library WHERE author = 'Linus'";

data = query(sql); while (row = data.next())

{ book = new Book(); book.setAuthor(row.get('author'); book\_list.add(book);}

С библиотекой ORM это будет выглядеть так: book\_list = BookTable.query(author="Linus");

**6.Необходимое программное обеспечения для работы веб-приложений.**

Веб-приложение — это полноценная программа, доступ к которой пользователь получает через интернет, то есть она не требует установки на устройство. Веб-приложение интерактивно и позволяет пользователям взаимодействовать с разными элементами: например, оставить заявку на покупку товара, оформить покупку или прокомментировать пост.

Любое веб-приложение, если оно работает в браузере, работает на 3 основных технологиях: HTML, CSS и JS.

Любое сложное приложение — не только картинка в браузере, это ещё и данные, которые пользователи используют или создают. Эти данные нужно уметь хранить, обрабатывать и выводить. Хранением и обработкой данных обычно занимается сервер или бэкенд.

## Клиент-серверная архитектура:

Архитектура — это описание системы на самом высоком уровне. При описании архитектуры мы не вдаёмся в подробности конкретного модуля, а скорее описываем их взаимодействие между собой поведение каждого из них. Клиент-серверная архитектура описывает, как взаимодействуют между собой клиент (фронтенд) и сервер (бэкенд).

Клиент : Клиент обращается с запросами к серверу. Роль клиента для сервера: сообщить серверу, что нужно сделать с данными, которые хранятся в базе, или с данными, которые он передаёт. Роль клиента для пользователя: представить данные в удобном виде и предоставить механизмы для их обновления. Для веба клиент почти всегда браузер.

Сервер: Сервер принимает запросы от клиента. Его роль в том, чтобы сохранять информацию от клиента в базе данных, обрабатывать её и предоставлять к ней доступ по некоторым правилам. На сервере помимо общения с клиентом могут запускаться какие-то фоновые задачи, например, индексирование информации в базе данных для более быстрого поиска, или запуск автоматических email-рассылок.

### База данных: База данных (БД) — это хранилище всей пользовательской и служебной информации. Роль - обеспечивать быстрый и бесперебойный доступ к этой информации и собственно хранение.В современных приложениях между клиентом и сервером общение строится именно на данных, а не на отрендеренных кусках разметки. Чаще всего для такого общения выбирают [JSON](https://doka.guide/tools/json/).

Сейчас большая часть приложений работает так:

1. Клиент делает первичный запрос на сервер.
2. Сервер отвечает HTML-страницей, иногда с набором каких-то данных внедрённых в виде JS-объекта в конце страницы.
3. Пользователь совершает какое-то действие, например, просит отсортировать таблицу.
4. Клиент в ответ на это действие решает, какой запрос отправить на сервер, строит этот запрос и отправляет его.
5. Сервер принимает этот запрос, обрабатывает его и отправляет на клиент порцию новых данных.
6. Клиент принимает данные и перерисовывает часть страницы по ним сам. То есть он уже не заменяет один кусок разметки другим готовым, а рисует разметку сам.

Плюсов такого общения (когда передаются только данные) несколько: Сервер и клиент становятся независимыми друг от друга. Сервер может ничего не знать об устройстве страниц, ему достаточно лишь уметь работать с БД и обрабатывать данные (первичная отрисовка может быть сделана самим сервером с помощью SSR). Количество информации, которое приходится передавать и принимать, меньше — а это уменьшает объём трафика. Логика приложения на сервере может быть проще, потому он и клиент становятся менее зависимы друг от друга в плане формата данных.

То есть для работы веб-приложений необходим работающий интернет, сервер с самим приложением и устройство клиента.

**7.Необходимое программное обеспечение для веб-разработки.**

Существует множество способов писать код для веб-приложений: от текстовых редакторов до облачных сред разработки.

## Текстовые редакторы для веб-разработки

[Vim](https://www.vim.org/) — расширенная версия [Vi](https://ru.wikipedia.org/wiki/Vi" \t "_blank), текстового редактора в UNIX. Основные возможности: бесконечная история отмены, обширное количество плагинов, поддержка сотен языков программирования и форматов файлов, мощный поиск и замена, интеграция со сторонними инструментами.

[Visual Studio Code](https://code.visualstudio.com/) — легкий, но мощный редактор исходного кода. В изначальной конфигурации используется для редактирования кода на JavaScript, TypeScript и Node.JS, а с помощью расширений поддерживает C++, C#, Python и PHP. Visual Studio Code, дописывает названия объявленнных переменных, функций и модулей, а также делает ссылку на соответствующий раздел документации. Возможна отладка кода напрямую из редактора, запуск приложения для отладки и присоединение к запущенным приложениям.

## Настольные интегрированные среды разработки (IDE)

[IDE NetBeans](https://netbeans.org/) — среда с открытым исходным кодом, мировым сообществом пользователей и разработчиков. С её помощью можно быстро и легко разрабатывать настольные, мобильные и веб-приложения на Java, JavaScript, HTML5, PHP, C/C++ и других языках. Netbeans предоставляет из коробки анализатор и редактор кода на Java, а также ряд новых инструментов для HTML5 и JavaScript, в том числе для Node.js, KnockoutJS и AngularJS. NetBeans делает отступы, дополняет слова и скобки, делает [синтаксическое](https://netbeans.org/kb/74/php/editorguide_ru.html#SyntaxHighlighting) и [семантическое](https://netbeans.org/kb/72/ide/javascript-editor_ru.html#semantic_highlighting) выделение исходного кода. С ним легко рефакторить код. Также NetBeans оснащён сниппетами, советами и генераторами кода.

## Облачные IDE🡪 Большинство десктопных приложений перешли в «облако», облачные среды также становятся популярными среди программистов. Не каждый сразу доверяет облачным IDE, но популярные инструменты вроде Github и Pastebin помогают привыкнуть к тому, что исходный код хранится не на локальной машине, а на стороннем сервере.

## Локальное окружение разработки

**MAMP** Аббревиатура [MAMP](https://www.mamp.info/ru/) расшифровывается как Macintosh, Apache, MySQL и PHP. После установки этого решения получите работоспособный веб-сервер. Ключевая особенность — простой и понятный интерфейс.

## Редакторы кода

**Visual Studio Code** [VSCode](https://code.visualstudio.com/) — это редактор с открытым исходным кодом, разработанный корпорацией Microsoft. Visual Studio Code по умолчанию поддерживает JavaScript, Node.js и TypeScript. Но при этом экосистема расширений настолько богата, что вы сможете найти утилиты для поддержки практически любого языка. В VSCode реализована первоклассная интеграция с другими продуктами Microsoft, в первую очередь с GitHub. Visual Studio Code полностью бесплатен и идеально подходит для большинства разработчиков.

## Системы контроля версий

Централизованное хранение кода, документирование изменений и предоставление возможности командной разработки — все эти задачи охватывают системы управления версиями или VCS. Git — ключевой инструмент любого современного разработчика. Простыми словами, это инструмент для документирования изменений, которые вы вносите в код проекта, и сохранения его в репозиториях. А еще это удобный инструмент совместной разработки.

## Фреймворки и библиотеки

Фреймворки — это крупные фрагменты заранее написанного кода, которые помогут вам быстрее начать работу над своими проектами. Фреймворк определяет структуру, задает правила и предоставляет необходимый набор инструментов для разработки. Библиотека — это набор уже готовых функций, классов и объектов для решения каких-то задач.

**React.js** [React.js](https://reactjs.org/) — популярная библиотека JavaScript, разработанная компанией Facebook. Если нужен современный и динамичный пользовательский интерфейс, стоит обратить внимание на React. Он использует расширенный синтаксис языка JavaScript, называемый JSX, для создания элементов. React — это мощное решение и он должен быть в вашем списке программ для изучения, особенно если заниматься фронтендом.

**Bootstrap** Адаптированность и быстрая работа на мобильных устройствах — стандартное требование к современному веб-проекту. Учитывая, что пользователи чаще обращаются к мобильным браузерам, а не к десктопным, разработчикам требуются инструменты для разработки сайтов, ориентированных именно на мобильные устройства. [Bootstrap](https://getbootstrap.com/)— отличный инструмент с открытым исходным кодом для веб-разработки с помощью HTML, CSS и JS. Свежая версия фреймворка выпущена под девизом: «Mobile First», что означает приоритет адаптации под мобильные устройства. Практически весь продукт был переделан с нуля, чтобы внедрить поддержку низких разрешений экрана и масштабирования.

**9.Схема соединения по протоколу HTTP.**

## Что такое HTTP

Hypertext Transfer Protocol, HTTP (протокол передачи гипертекста) — это протокол [прикладного уровня](https://hackware.ru/?p=6290#tcp/ip_osi) передачи данных, изначально — в виде гипертекстовых документов в формате HTML. HTTP — это протокол общего назначения без сохранения состояния, который можно использовать для других целей, а также с использованием расширения методов запроса, кодов ошибок и заголовков. По сути это протокол связи на основе TCP/IP, который используется для доставки данных по Всемирной паутине. Порт по умолчанию — TCP 80, но можно использовать и другие порты. Он обеспечивает стандартизированный способ взаимодействия компьютеров друг с другом. Спецификация HTTP определяет, как данные запросов клиентов будут создаваться и отправляться на службу, и как серверы отвечают на эти запросы.

## Базовая архитектура

Протокол HTTP — это протокол запроса/ответа, основанный на архитектуре клиент/сервер, где веб-браузер, роботы, поисковые системы и т. д. действуют как HTTP-клиенты, а веб-сервер действует как сервер. Клиент: HTTP-клиент через TCP/IP соединение отправляет запрос на сервер, используя один из методов запросов, URI и версию протокола, за которым следует MIME-подобное сообщение, содержащее модификаторы запроса, информацию о клиенте и возможное содержимое тела. Типичные HTTP клиенты — веб-браузеры. Сервер: HTTP-сервер, прослушивающий порт, на который пришёл запрос, отвечает строкой состояния, включающую версию протокола сообщения и код успеха или ошибки, за которыми следует MIME-подобное сообщение, содержащее информацию о сервере, метаинформацию объекта и возможное содержимое тела объекта.

HTTP-сообщение состоит из:1) **Стартовая строка** (англ. Starting line) — определяет тип сообщения; 2) **Заголовки** (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения; после каждой строки следует символ CRLF 3) **Пустая строка** (то есть строка, в которой ничего не предшествует CRLF), она обозначает конец полей заголовка 4) **Тело сообщения** (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой. Тело сообщения может отсутствовать, но стартовая строка и заголовок являются обязательными элементами.

Синхронизированный поток

Процесс отправки запроса на сервер можно разбить на несколько составных частей:1.В первую очередь осуществляется DNS-запрос, который должен преобразовать адрес сайта из URI формата в IP (числовая форма URI). Именно такой формат адреса используется в Всемирной сети. 2.После определения IP устанавливается связь между сервером и HTTP клиентом.3.Пересылка запроса.4.Задержка, в которую входит пересылка информации на сервер, ее обработка и отправка ответа на запрос. 5.Получение ответа на запрос

Заметка. Связь сохраняется только в пределах одного веб-ресурса. При переходе на другой хост связь разрывается и первый этап снова становится составляющей частью процесса обработки запроса. Такая форма работы сводит шансы сайтов к нулю в конкурентной борьбе с десктопными приложениями. Отсюда выплывает и первый способ ускорить работу сайта – нужно минимизировать количество обращений к серверу, прописанных в коде.

Параллельное HTTP соединение: Чтобы решить проблему большого времени ожидания и прерывания связи с хостом, была создана параллельная схема связи между клиентом и сервером. Можно одновременно установить соединение с несколькими хостами. Разработчики стандарта HTTP 1.1 советуют подключать не более 2 каналов соединения одновременно. Но следует учитывать, что спецификация вышла давно. Сейчас браузеры легко поддерживают связь с 4 каналами одновременно по умолчанию, а если порыться в настройках клиента, то этот показатель можно увеличить до 8. Каждый канал работает по старой схеме соединения, но рост их количества привел к существенным изменениям в плане времени загрузки ресурса.

Конвейерное HTTP соединение: С развитием технологий существенно начал развиваться и процесс взаимодействия сервера и браузера. Существенным прорывом в этом вопросе стало создание конвейерной схемы отправки запросов на сервер (в оригинале - HTTP pipelining). Согласно этой схеме, стало возможным по одному каналу отправлять несколько запросов, не дожидаясь ответа на них. В свою очередь сервер стал отправлять ответы на каждый запрос в порядке очереди. Благодаря этому нововведению стало также возможным сокращение количества TCP/IP-пакетов. Таким образом, можно в один такой пакет поместить несколько HTTP-запросов. Вследствие этого улучшится не только работа протокола, но и повысится эффективность функционирования сети Интернет в целом.

**12 Методы HTTP. Использование в современных API.**

REST — передача состояния представления. Технология позволяет получать и модифицировать данные и состояния удаленных приложений, передавая HTTP-вызовы через интернет или другую сеть. Серверное приложение дает доступ к своим данным клиентскому приложению по определенному URL. **Application Programming Interface (API)**, программный интерфейс приложения — это набор инструментов, который позволяет одним программам работать с другими. API учитывает, что программы могут быть написаны на различных языках программирования и работать в разных ОС. REST API позволяет использовать для общения между программами **протокол HTTP**, с помощью которого мы получаем и отправляем большую часть информации в интернете. **Чаще всего применяют:** Для связи мобильных приложений с серверными. Для построения микросервисных серверных приложений. Это архитектурный подход, при котором большие приложения разбиваются на много маленьких частей. Для предоставления доступа к программам сторонних разработчиков.

## Методы HTTP: основа работы REST API

Hypertext Transfer Protocol, HTTP (протокол передачи гипертекста) — это протокол [прикладного уровня](https://hackware.ru/?p=6290#tcp/ip_osi) передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в наст вр используется для передачи произвольных данных. Построение API-системы с использованием ресурсов, HTTP и различных запросов к ним как раз и будет Representational State Transfer (REST API) — передачей состояния представления. Определение операций API с точки зрения методов HTTP. Протокол HTTP определяет несколько методов, назначающих запросу семантическое значение.

## Методы GET: Успешное выполнение метода GET обычно возвращает код состояния HTTP 200 (ОК). Если ресурс не найден, метод должен вернуть код 404 (не найдено). Если запрос был выполнен, но в HTTP-ответе нет тела респосе, то он должен вернуть код состояния HTTP 204 (нет содержимого). Операция поиска, возвращающая совпадения, может быть реализована с так.

### Методы POST: Если метод POST создает новый ресурс, он возвращает код состояния HTTP 201 (создано). Текст ответа содержит представление ресурса. Если метод выполняет определенную обработку, но не создает новый ресурс, он может вернуть код состояния HTTP 200 и содержать результат операции в тексте ответа. Если клиент помещает недопустимые данные в запрос, сервер должен вернуть код состояния HTTP 400 (неверный запрос). Текст ответа может содержать дополнительные сведения об ошибке или ссылку на универсальный код ресурса (URI), где можно получить более подробную информацию.

### Методы PUT: Если метод PUT создает новый ресурс, он возвращает код состояния HTTP 201 (создано), как и метод POST. Если метод обновляет имеющийся ресурс, он возвращает коды состояния 200 (ОК) или 204 (нет содержимого). В некоторых случаях обновить имеющийся ресурс невозможно.

### Методы PATCH: С помощью запроса PATCH клиент отправляет набор обновлений в имеющийся ресурс в виде документа с исправлениями. Сервер обрабатывает документ с исправлениями, чтобы выполнить обновление. Документ с исправлениями не описывает весь ресурс, а только набор применяемых изменений. Вероятно, наиболее распространенный формат данных для веб-API — JSON.

Результат конкретного запроса должен зависеть от того, является ли целевой ресурс коллекцией или отдельным элементом. Различия между POST, PUT и PATCH могут запутать новичков.

Запрос POST создает ресурс. Сервер назначает URI для нового ресурса и возвращает этот URI клиенту. В модели REST запросы POST постоянно применяются к коллекциям. Запрос POST может использоваться для отправки данных для обработки в имеющийся ресурс без создания нового.

Запрос PUT создает ресурс или обновляет имеющийся ресурс. Если ресурс с таким URI уже существует, он заменяется. В противном случае создается новый ресурс, если сервер поддерживает это..

Запрос PATCH выполняет частичное обновление имеющегося ресурса. Это может быть более эффективно, чем использование PUT, так как клиент отправляет только изменения, а не все представление ресурса. С технической точки зрения PATCH также может создать новый ресурс, если это поддерживается сервером.

**10.HTTP-запросы. Структура, примеры, использование, методы.**

HTTP запросы - это сообщения, отправляемые клиентом, чтобы инициировать реакцию со стороны сервера. Их стартовая строка состоит из трёх элементов: [*Метод HTTP*](https://developer.mozilla.org/ru/docs/Web/HTTP/Methods), глагол (например, [GET](https://developer.mozilla.org/ru/docs/Web/HTTP/Methods/GET), [PUT](https://developer.mozilla.org/ru/docs/Web/HTTP/Methods/PUT) или [POST](https://developer.mozilla.org/ru/docs/Web/HTTP/Methods/POST)) или существительное (например, [HEAD](https://developer.mozilla.org/ru/docs/Web/HTTP/Methods/HEAD) или [OPTIONS](https://developer.mozilla.org/ru/docs/Web/HTTP/Methods/OPTIONS)), описывающие требуемое действие. Цель запроса, обычно [URL](https://developer.mozilla.org/ru/docs/Glossary/URL), или абсолютный путь протокола, порт и домен обычно характеризуются контекстом запроса. Формат цели запроса зависит от используемого HTTP-метода. Версия HTTP, определяющая структуру оставшегося сообщения, указывая, какую версию предполагается использовать для ответа.

HTTP-клиент отправляет HTTP-запрос на сервер в форме сообщения запроса («request message»), которое включает в себя следующие элементы: 1)Строка запроса 2) Ноль или более полей заголовка (общие заголовки, заголовки запроса и заголовки сущности), за которыми следует символ CRLF 3) Пустая строка (то есть строка, в которой ничего не предшествует CRLF), указывающая конец полей заголовка 4)Необязательно тело сообщения

**Строка запроса:** Строка запроса начинается с токена метода, за которым следует URI и версия протокола, и заканчивается CRLF. Элементы разделяются пробелами. 1Метод URI Версия HTTP

**Метод запроса:** Метод HTTP (англ. HTTP Method) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Метод Описание

GET🡪Используется для запроса содержимого указанного ресурса. С помощью метода GET можно начать какой-либо процесс. В тело ответного сообщения следует включить информацию о ходе выполнения процесса. Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»: GET /path/resource?param1=value1&param2=value2 HTTP/1.1

HEAD🡪Аналогичен методу GET, за исключением, что в ответе сервера отсутствует тело. Запрос обычно применяется для извлечения метаданных, проверки наличия ресурса и чтобы узнать, не изменился ли он с момента последнего обращения. При несовпадении метаданных ресурса с информацией в кэше — копия ресурса помечается как устаревшая.

POST🡪Применяется для передачи пользовательских данных заданному ресурсу, в том числе выгрузки файлов на веб-сервер. С помощью метода POST обычно загружаются файлы на сервер. В отличие от метода GET, метод POST не считается идемпотентным, то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться очередная копия этого комментария).

DELETE🡪Удаляет указанный ресурс.

CONNECT🡪Преобразует соединение запроса в прозрачный TCP/IP-туннель, обычно чтобы содействовать установлению защищённого SSL-соединения через нешифрованный прокси.

OPTIONS🡪Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. В ответ серверу следует включить заголовок Allow со списком поддерживаемых методов. Также в заголовке ответа может включаться информация о поддерживаемых расширениях.

PATCH🡪Аналогично PUT, но применяется только к фрагменту ресурса.

**URI** — это Uniform Resource Identifier, то есть универсальный идентификатор ресурса, который определяет ресурс, к которому следует применить запрос. Метод и описание

Звёздочка \* используется, когда HTTP-запрос применяется не к определённому ресурсу, а к самому серверу, и разрешён только в том случае, если используемый метод не обязательно применим к ресурсу. Например: OPTIONS \* HTTP/1.1

Абсолютный URI используется при отправке HTTP-запроса к прокси. Прокси-сервер запрашивается для пересылки запроса или обслуживания его из действительного кеша и возврата ответа. Например: GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1

Наиболее распространённая форма URI — это та, которая используется для идентификации ресурса на исходном сервере или шлюзе. Например, клиент, желающий получить указанный выше ресурс непосредственно с исходного сервера, создаст TCP-соединение с портом 80 хоста «www.w3.org» и отправит строки: GET /pub/WWW/TheProject.html HTTP/1.1

Host: www.w3.org

Обратите внимание, что путь не может быть пустым; если в исходном URI ничего нет, он ДОЛЖЕН быть указан как "/" (корень сервера)

**Поля заголовка запроса**

Поля заголовка запроса позволяют клиенту передавать на сервер дополнительную информацию о запросе и о самом клиенте. Эти поля действуют как модификаторы запроса. Доступны следующие важные поля заголовка запроса, которые можно использовать в зависимости от необходимости.

Примеры сообщений запроса. HTTP-запрос для получения страницы robots.txt с веб-сервера, запущенного на suay.ru.

|  |  |
| --- | --- |
|  | GET /robots.txt HTTP/2 /// Host: suay.ru  User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:82.0) Gecko/20100101 Firefox/82.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8  Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate, br  Connection: keep-alive /// Cookie: \_ga=............................  Upgrade-Insecure-Requests: 1  If-Modified-Since: Tue, 02 Jun 2020 09:00:16 GMT  If-None-Match: "966122f-10f-5a71623d78800"  Cache-Control: max-age=0 |

Здесь в запросе мы не отправляем какие-либо данные на сервер, поскольку мы просто хотим получить документ. Здесь используется общий заголовок **Connection**, а остальные заголовки являются заголовками запроса.

**11.HTTP-ответы. Структура, примеры, поля ответа, использование.**

После получения и интерпретации сообщения запроса сервер отвечает сообщением ответа HTTP, его структура: 1)Строка состояния 2)Ноль или более полей заголовка (общие заголовки, заголовки ответа и заголовки сущности), за которыми следует символ CRLF 3)Пустая строка (строка, в которой ничего не предшествует CRLF), указывающая на конец полей заголовка 4)Необязательно тело сообщения

**Строка состояния сообщения🡪** Строка состояния, состоящая из версии протокола, за которым следует числовой код состояния и связанная с ним текстовая фраза. Элементы разделяются пробелами. HTTP-версия Код состояния Причина-фраза

**Версия HTTP:** Сервер, поддерживающий HTTP версии 1.1, вернет следующую информацию о версии: HTTP/1.1

**Код состояния🡪**Клиент узнаёт по коду ответа о результатах его запроса и определяет, какие действия ему предпринимать дальше. Набор кодов состояния является стандартом, и они описаны в соответствующих документах RFC. Код состояния представляет собой трехзначное целое число, где первая цифра код состояния определяет класс ответа, а последние две цифры уточняют его. Первая цифра имеет 5 значений:

1xx🡪Информационный: Информирование о процессе передачи. 2xx🡪Успех:Информирование о случаях успешного принятия и обработки запроса клиента. 3xx🡪Перенаправление: сообщает клиенту, что для успешного выполнения операции необходимо сделать другой запрос. 4xx🡪Ошибка клиента: Указание ошибок со стороны клиента. 5xx🡪Ошибка сервера:Информирование о случаях неудачного выполнения операции по вине сервера.

**Поля заголовка ответа:** Поля заголовка ответа позволяют серверу передавать дополнительную информацию об ответе, которую нельзя поместить в строку состояния. Эти поля заголовка предоставляют информацию о сервере и о дальнейшем доступе к ресурсу, URI которого мы указали.

**Примеры сообщений ответа🡪**Ниже приведён пример сообщения HTTP-ответа, показывающего состояние ошибки, когда веб-сервер не может найти запрошенную страницу:HTTP/1.1 404 Not Found

Date: Sun, 18 Oct 2012 10:36:20 GMT /// Server: Apache/2.2.14 (Win32)

Content-Length: 230 /// Connection: Closed

Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html><head>

<title>404 Not Found</title> </head><body>

<h1>Not Found</h1>

<p>The requested URL /t.html was not found on this server.</p>

</body></html>

**13.Общая характеристика языка HTML. Назначение, структура. Понятие тега. Виды тегов.**

**HTML** ( *HyperText Markup Language* — «язык [гипертекстовой](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82) разметки») — стандартизированный язык гипертекстовой разметки документов для просмотра [веб-страниц](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) в [браузере](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80). Веб-браузеры получают HTML документ от сервера по протоколам [HTTP](https://ru.wikipedia.org/wiki/HTTP)/[HTTPS](https://ru.wikipedia.org/wiki/HTTPS) или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора. [Элементы HTML](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D1%8B_HTML) являются строительными блоками HTML страниц. С помощью HTML разные конструкции, изображения и другие объекты, такие как [интерактивная веб-форма](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0_(HTML)), могут быть встроены в отображаемую страницу. HTML предоставляет средства для создания заголовков, абзацев, списков, ссылок, цитат и других элементов. В HTML можно встроить программный код на языке программирования [JavaScript](https://ru.wikipedia.org/wiki/JavaScript), для управления поведением и содержанием веб-страниц. Также включение [CSS](https://ru.wikipedia.org/wiki/CSS) в HTML описывает внешний вид и макет страницы. Технология HTML состоит в том, что в обычный текстовый документ вставляются управляющие символы (тэги) и в результате мы получаем Web-страницу. Браузер при загрузке Web-страницы представляет её на экране в том виде, который задаётся тэгами.

*Основными достоинствами HTML-документов являются:* -малый информационный объём; -возможность просмотра на ПК, оснащенных различными операционными системами.

## 2. Основные понятия языка html

*Дескриптор (тэг)* – основной элемент кодирования, принятый в стандарте HTML. Они служат для привлечения внимания к определённым словам в документе, давая понять браузеру, что нужно выделить, где отобразить графику и тд. Тэг заключается в угловые скобки (<>). Могут быть одиночными или парными. Для парных обязательно наличие открывающего тега <tag>(чаще всего с параметром и его значением) и закрывающего тэга </tag>.

*Контейнер* – это дескрипторная пара, состоящая из открывающего и закрывающего дескрипторов (тэгов) <tag>…</tag>. Контейнеры предназначены для хранения некоторой информации, например текста или других HTML-дескрипторов. *Элемент HTML-документа* – это контейнер вместе с его содержимым. Тэги могут записываться в любом регистре.

*Создание структуры Web-страницы:*

Всё содержимое файла Интернет-страницы заключается в контейнер <html>…</html>, указывающий браузеру, что данный текст представляет собой HTML-документ и, возможно содержит в себе тэги, которые браузер должен выявить, распознать, интерпретировать. Интернет-страница состоит из двух частей: заголовка (HEAD) и тела (BODY). Эту базовую структуру в простейшем виде можно наглядно показать следующим образом:

<HTML> начало контейнера HTML-документа <HEAD> начало контейнера заголовка <TITLE> начало контейнера строки – названия страницы

… строка названия страницы </TITLE> конец контейнера строки – названия страницы </HEAD> конец контейнера заголовка <BODY> начало контейнера тела страницы … тело (всё содержимое) страницы </BODY> конец контейнера тела страницы </HTML> конец контейнера HTML-документа

<! – и -- > - такой полутэг используется для добавления комментариев на страничке, т. е. для полезных пояснений, которые не показываются в браузере.

**Формат тегов HTML:** <имя\_тега>Содержимое</имя\_тега>

**Например:**<a>Текст внутри тега</a>

Отметим ещё одну особенность HTML –он является регистронезависимым, имена тегов и атрибутов можно записывать и в нижнем, и в верхнем регистре.

### Виды тегов.

### Теги языка HTML можно условно разделить на несколько категорий:

**1. Теги для форматирования текста и указания ссылок:** h1, h2, h3, h4, h5, h6 – теги заголовков; b, em – теги для изменения начертания текста; ul, ol – теги для создания списков; a – тег для создания ссылок.

**2. Теги структуры документа:** nav – тег для создания навигации по сайту; aside – создаёт боковую панель на сайте, где размещается неосновной контент; header – тег для создания шапки сайта или раздела; section – тег для создания раздела веб-страницы; div – тег, который выделяется отдельный блок, с целью его последующего изменения; span – определяет строчные элементы.

**3. Функциональные теги:** html – включает в себя всё содержимое страницы; body – раздел html-файла, где содержится техническая инфомрация; script – предназначен для создания/подключения скриптов; link – предназначен для подключения внешних файлов (шрифтов или стилей); meta – содержит информацию, предназанченную для поисковых систем и браузеров.

**4. Теги встраиваемого контента:** img – изображение; audio ; iframe – отдельная область документа, где можно подгружать независимые документы.

**5. Теги форм (для взаимодействия с пользователем):** form, input – теги для создания формы. **6. Теги таблиц:** table, td, tr – теги для создания таблицы.

**16.HTML. Теги форматирования текста.**

**HTML** ( «язык [гипертекстовой](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82) разметки») — стандартизированный язык гипертекстовой разметки документов для просмотра [веб-страниц](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) в [браузере](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80). Веб-браузеры получают HTML документ от сервера по протоколам [HTTP](https://ru.wikipedia.org/wiki/HTTP)/[HTTPS](https://ru.wikipedia.org/wiki/HTTPS) или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора.

## Теги выделения фрагментов текста

Тег <strong> используется для выделения важных участков текста. Браузеры обычно отображают текст внутри тега <strong> жирным шрифтом. <strong> - тег логического выделения. Используя его, вы указываете на важность текста в выделенной области. <strong>Важный фрагмент текста</strong>

Тег <em> используется для акцентирования внимания, "подчеркивания" фрагмента текста. Браузеры обычно отображают текст внутри тега <em> курсивом. <em> - тег логического выделения.

Тег <mark> используется для подсветки фрагмента текста. <mark>Подсвеченный фрагмент</mark>

Тег <b> используется для написания текста жирным шрифтом. Тег физического форматирования, не придает выделенным участком логический вес. <b>Текст жирным шрифтом</b>

Тег <i> используется для написания текста курсивом. Используйте тег <i> только если характер выделения не подходит под другой тег форматирования текста по смыслу. <i>Текст курсивом</i>

Тег <s> используется для указания некорректности, неактуальности выделенного фрагмента. Текст внутри тега <s> выводится перечеркнутым. <s>Перечеркнутый текст</s>

Тег <small> используется для написания текста шрифтом меньшего размера. <small>Текст меньшего размера</small>

Тег <sub> используется для создания текста нижнего индекса. <sub>Текст нижнего индекса</sub> Тег <sup> используется для создания текста верхнего индекса. <sup>Текст верхнего индекса</sup>

Тег <pre> указывает, что текст внутри должен быть выведен в том формате, в котором он был написан в редакторе: с учетом пробелов, сохранением позиции строк. <pre>Предварительно отформатированный текст</pre>

Теги устанавливающие суть содержимого

Теги <h1> - <h6> указывают, что содержимое является заголовком (подзаголовком). <h1> - заголовок наиболее высокого уровня, <h6> - наименьшего. Заголовки определяют структуру текста HTML документа. Браузера обычно выделяют заголовки размером шрифта, в зависимости от уровня заголовка. <h1>Заголовок</h1>

Тег <dfn> указывает, что содержимое является термином. Браузеры обычно выводят содержимое тега <dfn> курсивом. <dfn>Термин</dfn>

Тег <q> используется для указания на короткий процитированный участок внутри строки текста. Браузеры обычно выводят содержимое тега в кавычках. <q>Короткая цитата</q>

Тег <blockquote> указывает, что содержимое является цитатой. Содержимое тега выводится в отдельном блоке. <blockquote>Цитата</blockquote>

Тег <code> указывает, что содержимое является кодом компьютерной программы. Браузеры обычно отображают текст внутри тега <code> моноширинным шрифтом. <code>Код компьютерной программы</code>

Тег <samp> указывает, что содержимое является результатом вывода компьютерной программы. Браузеры обычно отображают текст внутри тега <samp> моноширинным шрифтом. <samp>Результат вывода компьютерной программы</samp>

Тег <kbd> указывает, что содержимое является клавиатурным вводом. Браузеры обычно отображают текст внутри тега <kbd> моноширинным шрифтом. <kbd>Клавиатурный ввод</kbd>

Тег <var> указывает, что содержимое является переменной компьютерной программы. Браузеры обычно отображают текст внутри тега <var> курсивом. <var>Переменная компьютерной программы</var>

Тег <time> указывает, что содержимое является датой, временем, периодом времени или событием, относящемся к указаному в атрибуте datatime времени. <time>Дата и/или время</time>

**14.HTML5. Особенности, примеры тегов, назначение. Понятие семантических элементов.**

HTML — это язык для структурирования и представления содержимого, HTML5 (HyperText Markup Language, version 5) — это пятая версия стандарта, которая ещё находится в разработке, но уже является вполне рабочей. Цель разработки HTML5 — улучшение уровня поддержки мультимедиа-технологий при сохранении удобочитаемости кода для человека и простоты анализа для парсеров. HTML5 был создан, как единый язык разметки, который расширяет, улучшает и рационализирует разметку документов, а также добавляет единое API для сложных веб-приложений. HTML5 обеспечивает удобную читаемость программ для человека и обработки для браузеров. Имеет поддержку части тэгов HTML4 и языка JavaScript.

Какие же основные преимущества  HTML5🡪применения более простого вида кода, например, div заменены более совершенными элементами; дизайнерские решение, которые позволяют сделать индивидуальное решения для сайта. Также улучшен пользовательский интерфейс; можно использовать новые поля ввода для разных целей, например, поиска.; новая более совершенная семантика HTML5 дает возможность быстрее и проще различать футер, заголовки, панель навигации. С HTML5 можно использовать новые элементы, в том числе <canvas>, <audio>, <video>.

Также среди нововведений HTML5 стоит выделить следующие элементы: Figure – рисунок; Header – заголовок страницы; Section – крупный блок страницы; Footer – нижняя часть страницы; meta charset = «UTF-8» — обновление кодировки страницы; Nav – навигация по сайту; Aside – дополнительный контент в виде боковой колонки; Article – статья, основная часть контента. HTML5 позволяет машинам «читать» изображение и анимацию. Этого не было раньше.

Семантическая вёрстка — подход к разметке, который опирается не на содержание сайта, а на смысловое предназначение каждого блока и логическую структуру документа. В качестве примера не семантических элементов можно привести теги <div> и <span>. Они ничего не говорят о характере их контента. Примеры семантических элементов: <form>, <table> и <article>. Они четко описывают, какого характера контент они содержат.

Семантические элементы HTML5 поддерживаются всеми современными браузерами. Основные семантические теги HTML

### <article>🡪Значение: независимая, отделяемая смысловая единица, например комментарий, статья, виджет. <article> определяет независимый, самодостаточный контент. Контент, помещенный в этот него, должен иметь смысл сам по себе, т.е. должен быть понятен в отрыве от остальных частей веб-сайта. Желателен заголовок внутри. 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. Может иметь свой заголовок, встречаться несколько раз на странице. 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. Типовые ошибки: многие считают, что в <nav> может быть только список навигационных ссылок, но[согласно спецификации](https://html.spec.whatwg.org/multipage/sections.html#the-nav-element) 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. 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 тд. Чаще всего повторяется на всех страницах сайта. Этих элементов может быть несколько на странице. Тег <footer> не обязан находиться в конце раздела. Типовые ошибки: использовать только как подвал сайта.

## Как разметить страницу с точки зрения семантики

Процесс разметки можно разделить на несколько шагов с разной степенью детализации. 1)Крупные смысловые блоки на каждой странице сайта. Теги: <header>, <main>, <footer>. 2)Крупные смысловые разделы в блоках. Теги: <nav>, <section>, <article>, <aside>. 3)Заголовок всего документа и заголовки смысловых разделов. Теги: <h1>-<h6>. 4)Мелкие элементы в смысловых разделах. Списки, таблицы, демо-материалы, параграфы и переносы, формы, цитаты, контактная информация и прогресс. 5)Фразовые элементы. Изображения, ссылки, кнопки, видео, время и мелкие текстовые элементы.

**15.HTML. Теги заголовка веб-страницы.**

HTML (HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора.

К этим тегам относятся элементы, которые располагаются в контейнере <head>. Все эти теги напрямую не отображаются в окне браузера, за исключением тега <title>, который определяет название веб-страницы.

Элемент <head> располагается перед элементом <body>. Он содержит информацию о веб-странице. Это и есть заголовок документа HTML. Информация, расположенная в элементе <head>, не отображается в окне браузера.

<title>Используется для отображения строки текста в левом верхнем углу окна браузера, а также на вкладке. Такая строка сообщает пользователю название сайта и другую информацию, которую добавляет разработчик.

<meta>Метатеги используются для хранения информации, предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных. Хотя тег <meta> всего один, он имеет несколько атрибутов, поэтому к нему и применяется множественное число.

Так, для краткого описания содержимого веб-страницы используется значение description атрибута name, как показано в примере 5.2.

<html>

<head>

<title>HTML</title>

<meta name="description" content="Сайт об HTML и создании сайтов">

<meta http-equiv="content-type" content="text/html; charset=utf-8">

</head>

<body>

<p>...</p>

</body>

</html>

Описание сайта, заданное с помощью тега <meta> и значения description, обычно отображается в поисковых системах или каталогах при выводе результатов поиска. Значение keywords также предназначено в первую очередь для повышения рейтинга сайта в поисковых системах, в нем перечисляются ключевые слова, встречаемые на веб-странице (пример 5.3).

<html>

<head>

<title>HTML</title>

<meta name="keywords" content="HTML, META, метатег, тег, поисковая система">

<meta http-equiv="content-type" content="text/html; charset=utf-8">

</head>

<body>

<p>...</p>

</body>

</html>

Ключевые слова можно перечислять через пробел или запятую. Поисковые системы сами приведут запись к виду, который они используют.

**17.HTML. Встроенные и блочные элементы.**

**HTML** (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *HyperText Markup Language* — «язык [гипертекстовой](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82) разметки») — стандартизированный язык гипертекстовой разметки документов для просмотра [веб-страниц](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) в [браузере](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80). Веб-браузеры получают HTML документ от сервера по протоколам [HTTP](https://ru.wikipedia.org/wiki/HTTP)/[HTTPS](https://ru.wikipedia.org/wiki/HTTPS) или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора.

Строчные элементы: Строчные элементы – это элементы, ширина которых по горизонтали определяется шириной их содержимого. Например, элементы <strong> и <em>. В отличие от блочных элементов строчные элементы не имеют собственных строк по горизонтали. То есть строчные элементы будут располагаться в одной строке подряд, если не добавить разрыв с помощью элемента разрыва строки <br>. Это стандартное размещение обычно удобно использовать, так как это позволяет пометить отдельные слова в абзаце с помощью встроенного элемента, типа <strong> или <em>, без переноса соседнего текста на следующую строку.

## Блочные элементы: Блочные элементы ведут себя иначе, чем строчные элементы, поскольку они занимают всю строку горизонтального пространства на веб-странице. Они автоматически начинаются с новой строки, а последующие элементы автоматически переносятся на новую строку. Например, элементы заголовка <h1> - <h6> – это блочные элементы, которые автоматически помещают свое содержимое в новую строку и перемещают любое содержимое, которое идет за ним, на следующую строку. Блочные элементы всегда будут переносить строчные элементы на следующую строку, даже если записать эти элементы HTML в одной строке документа HTML. <p>Этот абзац — блочный элемент</p>

[<address>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/address)Контактная информация. [<blockquote>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/blockquote) Длинная («блочная») цитата.[<div>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/div) Фрагмент документа. [<fieldset>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/fieldset) Группирование элементов формы. [<form>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/form) Форма ввода. [<hr>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/hr) Горизонтальная разделительная линия. [<li>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/li) Пункт списка[<ol>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/ol) Упорядоченный список.

[HTML5](https://developer.mozilla.org/ru/docs/HTML/HTML5): [<article>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/article)Содержание статьи. [<aside>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/aside) Побочное содержание. [<details>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/details) Раскрывающийся блок с подробностями. [<figure>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/figure) Группирование медиа-контента с подписью. [<header>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/header) Верхняя часть («шапка») раздела или страницы..[<main>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/main) Содержит основной контент, уникальный для страницы. [<nav>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/nav) Содержит навигационные ссылки.

Элемент ***<div>*** — это общий **блочный** элемент, который применяется для группировки одного или нескольких блочных элементов. Он указывает, что сгруппированные вместе элементы образуют некую смысловую единицу и должны рассматриваться, например таблицей стилей CSS, как одна единица. Используя элемент ***<div>*** вместе с таблицами стилей CSS, вы сможете придать одинаковое стилевое оформление для всех элементов, помещенных в контейнер ***<div>***.

Элемент ***<span>*** — это общий **встроенный** элемент, который используется для встроенных элементов, которые не вводят переводов строк. Элемент ***<span>*** может содержать только текст и другие встроенные элементы (вы не можете поместить туда блочные элементы: заголовки, списки, элементы группировки контента и т. д.).

Встроенные элементы являются частью потока документа, и размер обычно не изменяться вручную. Только встроенные элементы могут находиться во встроенных элементах. И что еще более запутанно, встроенные элементы - это единственные элементы, которые могут содержаться в элементе абзаца, даже если <p> -элемент является блочным элементом. <p></p>; элемент является блочным элементом, тогда как <b> - встроенный . (b обозначает жирный и меняет внешний вид текста)

**18.HTML. Теги форматирования списков.**

HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора.

Списком называется взаимосвязанный набор отдельных фраз или предложений, которые начинаются с маркера или цифры. Списки предоставляют возможность упорядочить и систематизировать разные данные и представить их в наглядном и удобном для пользователя виде.

В HTML для создания списков используются **теги группы lists**. К ним относятся:

* [<ul>](https://guruweba.com/html/teg-ul-html-markirovannyy-spisok/) - тег маркированного списка; устанавливает маркированный список, каждый элемент которого начинается с небольшого символа — маркера.
* [<ol>](https://guruweba.com/html/teg-ol-html-numerovannyy-spisok/) - тег нумерованного списка; Тег <ol> устанавливает нумерованный список, т.е. каждый элемент списка начинается с числа или буквы и увеличивается по нарастающей.
* [<li>](https://guruweba.com/html/teg-li-html-element-spiska/) - тег элемента списка; Тег <li> определяет отдельный элемент списка. Внешний тег <ul> или <ol> устанавливает тип списка — маркированный или нумерованный.
* [<dl>](https://guruweba.com/html/teg-dl-html-spisok-opredeleniy/) - тег списка определений;
* [<dt>](https://guruweba.com/html/teg-dt-html-termin-spiska-opredeleniy/) - тег термина в списке определений;
* [<dd>](https://guruweba.com/html/teg-dd-html-opisanie-v-spiske-opredeleniy/) - тег описания термина в списке определений.

Тройка элементов предназначена для создания списка определений. Каждый такой список начинается с контейнера **<dl>**, куда входит тег **<dt>** создающий термин и тег **<dd>** задающий определение этого термина. Закрывающий тег **</dd>** не обязателен, поскольку следующий тег сообщает о завершении предыдущего элемента. Тем не менее, хорошим стилем является закрывать все теги.

В HTML есть три типа списков: маркированный, нумерованный и список определений (терминов).

Маркированный список. Для создания маркированного списка используются теги <ul> и <li>. Тег ul - это контейнер маркированного списка. Тег li - элемент списка.

<ul>||| <li>Яблоки</li>||| <li>Абрикосы</li>||| <li>Бананы</li>||| <li>Сливы</li>|||</ul>

Типы маркеров: для маркированного списка доступны 3 типа маркеров по умолчанию: disc, square и circle. Задать тип маркера можно при помощи CSS свойства list-style-type. Тип маркера может быть задан, как для списка в целом (свойство применяется к <ul>), так и для конкретного элемента (свойство применяется к <li>).

<ul>||| <li style="list-style-type: disc;">Disc - закрашенный круг, точка.</li>||| <li style="list-style-type: circle;">Circle - окружность, пустая внутри.</li>||| <li style="list-style-type: square;">Square - квадрат.</li>|||</ul>

Положение маркера в списке: HTML поддерживает 2 типа положений маркеров списка: внутри или снаружи. Положение маркера регулируется CSS свойством list-style-position. Положение может быть задано, как для списка в целом, так и для отдельного элемента.

<ul>||| <li style="list-style-position: inside;">Маркер внутри (inside)</li>||| <li style="list-style-position: outside;">Маркер снаружи (outside)</li>|||</ul>

Свой маркер в HTML списке 🡪 Использовать в качестве маркера HTML списка можно и свою картинку. Для этого используйте CSS свойство list-style-image.

<ol style="list-style-image: url('/images/image.png');"> ||| <li>Дизайн</li>||| <li>Верстка</li>||| <li>Интеграция</li>|||</ol>

Нумерованный список 🡪 Чтобы создать нумерованный список используйте теги <ol> и <li>. Тег ol - это контейнер нумерованного списка. Тег li - элемент списка. <ol>||| <li>Выучить HTML</li>||| <li>Добавить CSS</li>||| <li>Освоить JavaScript</li>|||</ol>

## Список определений🡪 Для создания списка определений используются теги <dl>, <dt> и <dd>. Тег dl - это контейнер списка определений, dt - термин, dd - описание термина. <dl>||| <dt>Спектр</dt>||| <dd>Распределение значений физической величины, например, массы, частоты или энергии.</dd>||| <dt>Люминесценция</dt>||| <dd>Эффект нетеплового свечение вещества, происходящий после поглощения им энергии.</dd>|||</dl>

**19.HTML. Таблицы.**

HTML ( «язык гипертекстовой разметки») — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора.

HTML-таблицы упорядочивают и выводят на экран данные с помощью строк или столбцов. Таблицы состоят из ячеек, образующихся при пересечении строк и столбцов.

Ячейки таблиц могут содержать любые HTML-элементы. Каждой таблице можно добавить связанный с ней заголовок, расположив его перед таблицей или после. Таблицы редко используются для вёрстки веб-страниц и компоновки отдельных элементов, так как это не обеспечивает гибкость структуры и адаптивность сайта, существенно увеличивая HTML-разметку.

Создание таблиц в HTML

Таблица создаётся при помощи элемента <table></table>, который является контейнером для элементов таблицы и все элементы должны находиться внутри него. С помощью данной разметки можно создать таблицу, состоящую из двух столбцов и двух строк: <table><tr><th>текст заголовка</th><th>текст заголовка</th></tr> <!--ряд с ячейками заголовков-->

<tr><td>данные</td><td>данные</td></tr> <!--ряд с ячейками тела таблицы--></table>

По умолчанию таблица и ячейки не имеют видимых границ. Границы задаются с помощью свойства border: /\* внешние границы таблицы серого цвета толщиной 1px \*/ table { border: 1px solid grey;}

/\* границы ячеек первого ряда таблицы \*/th { border: 1px solid grey;}

/\* границы ячеек тела таблицы \*/ td { border: 1px solid grey;}

Промежутки между ячейками таблицы убираются с помощью свойства table {border-collapse: collapse;}.

Ширина таблицы по умолчанию равна ширине её внутреннего содержимого. Чтобы установить ширину, нужно задать значение для свойства width: width: 100%; width: 600px;

Если для ячеек заданы внутренние отступы и границы, то ширина таблицы будет включать в себя следующие значения: padding-left и padding-right, ширина border-left плюс ширина border-right последней ячейки в ряду. Если заданы ширина и границы ячеек, то ширина таблицы будет складываться из ширины ячеек плюс ширина border-left и ширина border-right последней ячейки в ряду.

Строки или ряды таблицы создаются с помощью элемента <tr>. Количество горизонтальных строк таблицы определяется количеством элементов <tr></tr>. Элемент <th> создаёт заголовок столбца — специальную ячейку, текст в которой выделяется полужирным. Количество ячеек заголовка определяется количеством <th></th>. Для элемента доступны colspan, rowspan, headers.

Элемент <td> создаёт ячейки таблицы, внутрь которых помещаются данные таблицы. Элементы <td></td>, расположенные в одном ряду, определяют количество ячеек в строке таблицы.

Количество пар ячеек <td> должно быть равно количеству пар ячеек <th>. Для элемента доступны атрибуты colspan, rowspan, headers.

Элемент <caption> создает подпись таблицы. Добавляется непосредственно после тега <table>, вне строки или ячейки.

Элемент <colgroup> создает структурную группу столбцов, выделяя логически однородные ячейки. Группирует один или более столбцов для единого форматирования, позволяя применить стили к столбцам вместо того, чтобы повторять стили для каждой ячейки и для каждой строки. Добавляется непосредственно после тегов <table> и/или <caption>.

Элемент <col> формирует группы столбцов, которые делят таблицу на разделы, не относящиеся к общей структуре, т.е. не содержащие информацию одного типа. Позволяет задавать свойства столбцов для каждого столбца в пределах элемента <colgroup>. С помощью атрибута style можно изменить основной цвет фона ячеек. Для элемента <col> доступен атрибут span, задающий количество столбцов для объединения. <table> <colgroup>

<col span="2" style="background:Khaki"><!-- С помощью этой конструкции задаем цвет фона для первых двух столбцов таблицы-->

<col style="background-color:LightCyan"><!-- Задаем цвет фона для следующего (одного) столбца таблицы--> </colgroup>

<tr> <th>№ п/п</th> <th>Наименование</th> <th>Цена, руб.</th> </tr></table>

Элемент <thead> создает группу заголовков для строк таблицы с целью задания единого оформления. Используется в сочетании с <tbody> и <tfoot> для указания каждой части таблицы.

Элемент должен быть использован в следующем порядке: как дочерний элемент <table>, после <caption> и <colgroup>, и перед <tbody>, <tfoot> и <tr> элементами. В пределах одной таблицы можно использовать один раз.

Элемент <tbody> группирует основное содержимое таблицы. Используется в сочетании с элементами <thead> и <tfoot>. Элемент <tfoot> создает группу строк для представления информации о суммах или итогах, расположенную в нижней части таблицы. Используется в таблице один раз. Располагается после элемента <thead>, перед элементами <tbody> и <tr>.

Атрибуты colspan и rowspan объединяют ячейки таблицы. Атрибут colspan задает количество ячеек, объединенных по горизонтали, а rowspan — по вертикали. <td colspan="5" style="text-align:right">ИТОГО:</td><td>1168,80</td><!-- Задаем количество ячеек по горизонтали для объединения-->

**20.HTML. Гиперссылки и якоря.**

HTML — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора.

Для создания ссылок предназначен тег <a>. Атрибуты тега <a>:

href🡪Указывает адрес назначения. Это может быть абсолютный или относительный URL-адрес или имя якоря. Hreflang🡪Указывает язык ресурса по ссылке. Используйте языковые значения из BCP47 для HTML5 и RFC1766 для HTML 4. Используйте этот атрибут, только если задан href.

rel🡪Определяет отношения между текущим документом и документом, на который ведет ссылка, заданная атрибутом href.

target🡪Определяет где нужно открыть ссылку, в новой вкладке или окне.

title🡪Указывает дополнительную информацию о ссылке. Эта информация чаще всего отображается в виде всплывающей подсказки, когда курсор перемещается по ссылке. Этот атрибут может использоваться почти во всех HTML-тегах.

Теги <a> обычно используются для связывания отдельных веб-страниц, но их также можно использовать для связи между различными местами в одном документе, в оглавлении и даже при запуске внешних приложений.

Это основной способ использования [элемента гиперссылки <a>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/A): <**a** href="http://example.com/"> example.com</**a**>

<**a** href="ftp://example.com/">Ссылка на FTP</**a>**

## [Ссылка на якорь](https://yocton.ru/html5/giperssylki-i-yakorya#ssylka-na-yakor)

Якоря (закладки) используются для перехода к определенным элементам на странице HTML. Тег <a> может указывать на любой элемент с атрибутом id. Якоря в основном используются для перехода к подразделу страницы и используются в сочетании с заголовками. В HTML5 вместо устаревшего атрибута name для определения якоря требуется указывать атрибут id.

<h2 id="Topic1">Первая тема</h2> <p>Содержимое первой темы</p>

<h2 id="Topic2">Вторая тема</h2> <p>Содержимое второй темы</p>

Теперь вы можете использовать якорь в своем оглавлении: <h1>Содержание</h1>

<a href='#Topic1'>Щелкните для перехода к первой теме</a>

<a href='#Topic2'>Щелкните для перехода ко второй теме</a>

Эти якоря также привязаны к веб-странице, на которой они находятся (page1.html). Таким образом, можно связывать одну страницу с якорем на другой странице. <**a** href="page1.html# Topic1"></**a**>.

## [Ссылка на страницу на том же сайте](https://yocton.ru/html5/giperssylki-i-yakorya#ssylka-na-stranicu-na-tom-zhe...)

Вы можете использовать [относительный путь](https://ru.wikipedia.org/wiki/%D0%9F%D1%83%D1%82%D1%8C_%D0%BA_%D1%84%D0%B0%D0%B9%D0%BB%D1%83) для ссылки на страницы в пределах одного сайта. <a href="/page">ссылка</a> \\\ Вышеприведенный пример ссылается на файл page в корневом каталоге (/) сервера.

<a href="/page">ссылка</a> |<a href="http://example.com/page">ссылка</a>. Обе ссылки указывают на файл page в корневом каталоге example.com.

## [Открыть ссылку в новой вкладке или окне](https://yocton.ru/html5/giperssylki-i-yakorya#otkryt-ssylku-v-novoy-vkladke...)

Атрибут target указывает где нужно открыть ссылку. Установив его значение равно \_blank, вы сообщаете браузеру открыть ее в новой вкладке или окне. <a href="example.com" target="\_blank">Ссылка</a>

**21.HTML. Универсальные атрибуты тегов.**

HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора. Многие теги в HTML имеют атрибуты. Часто они являются опциональными, а некоторые универсальны.

### accesskey🡪Позволяет получить доступ к элементу с помощью заданного сочетания клавиш. <a href="images/primer.jpg" accesskey="x">Клавиша X для открытия ссылки</a>

Разные браузеры используют различные комбинации клавиш. Допустим, мы используем клавишу **X**, тогда команды будут иметь следующий вид: Google Chrome: **Alt+X,** Mozilla Firefox: **Shift+Alt+X**. Атрибут accesskey применим к тегам <a>, <area>, <button>, <input>, <label>, <legend> и <textarea>.

class 🡪 Задаёт стилевой класс, который позволяет связать определённый тег со стилевым оформлением (CSS). Допускается указывать сразу несколько классов, разделяя их пробелами. Имена классов могут содержать в себе латинские буквы, цифры, символы дефиса (-) и подчеркивания (\_). <style> p.red { font-family: arial, helvetica, sans-serif; color: red;}</style>

<body> <p class="red">Это текст красного цвета</p> </body>

### dir 🡪 Задаёт направление и отображение текста — слева направо или справа налево. Использовать данный атрибут каждый раз не представляет нужды по той причине, что современные веб-браузеры способны самостоятельно «понимать» нужное направление текста, если используется [кодировка](https://webistore.ru/raznoe/chto-takoe-kodirovka/) Unicode, но его можно изменить вручную. Атрибут dir может принимать одно из двух значений: **ltr** (текст отображается слева направо) или **rtl** (текст отображается справа налево). <p dir="rtl">текст для отображения справа налево</p>

### hidden 🡪 Скрывает содержимое элемента от просмотра. Этот элемент не будет отображаться на странице, но будет доступен через скрипты. В качестве значения можно указать hidden (hidden=»hidden») или оставить атрибут пустым (hidden=»» или hidden). <title>hidden</title>

### id🡪 Задаёт стилевой идентификатор, который представляет из себя уникальное имя элемента. Указанное имя используется для изменения стиля элемента и обращения к нему через скрипты. Идентификатор должен встречаться в коде документа только один раз. <div id="help">Нужна помощь?</div>

### lang🡪 Указывает язык, который используется внутри текущего элемента. В качестве значения используются общеупотребительные коды языков. q:lang(de) { quotes: "\201E" "\201C"; /\* Вид кавычек для немецкого языка \*/ }

<p>Цитата на немецком: <q lang="de">Der Mensch, versuche die Gotter nicht</q>.</p>

### style🡪 Применяется для определения стиля элемента с помощью правил CSS. В качестве значений указываются стилевые правила: сначала следует имя стилевого свойства, затем через двоеточие его значение. Стилевые свойства разделяются между собой точкой с запятой. <p style="color: red; font-size: 2em">Пример применения стиля.</p>

### tabindex🡪 Устанавливает порядок получения фокуса при переходе между элементами при помощи клавиши **Tab**. Переход происходит от меньшего значения к большему. Tabindex применяется к тегам <a>, <area>, <button>, <input>, <object>, <select> и <textarea>. <p><button tabindex="5">Пятый элемент</button></p> <p><button tabindex="1">Первый элемент</button></p>

title🡪 Описывает содержимое элемента в виде всплывающей подсказки, которая появляется при наведении курсора на элемент. Вид подсказки зависит от используемого браузера и операционной системы. Значением атрибута может быть любая текстовая строка, заключенная в кавычки. <p title="Это всплывающий текст.">Наведите мышью на этот текст.</p>

**22.HTML. Изображения, рисунки и мультимедиа.**

HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора.

## Добавление изображений🡪Чтобы добавить изображения на страницу мы используем строчный элемент <img>. Он относится к самостоятельным или к пустым элементам; это означает, что не содержит какого-либо контента и существует в виде единственного тега. Для работы <img> должен быть включен атрибут src со значением, указывающим исходник изображения. Значение атрибута src это URL. Наряду с атрибутом src должен быть применён атрибут alt (альтернативный текст), который описывает содержимое изображения. Альтернативный текст будет отображаться вместо изображения, если по какой-то причине изображение не доступно. <img src="dog.jpg" alt="Чёрно-бело-коричневая собака с надетым на неё платком">

### Поддерживаемые форматы изображений🡪Изображения поступают в разных форматах файлов и каждый браузер может поддерживать (или не поддерживать) различные форматы. По большому счёту, наиболее типовые поддерживаемые форматы изображений в Интернете — это gif, jpg и png.

### Размеры изображений🡪Одним из способов установки размеров является использование атрибутов width и height непосредственно в теге <img> в HTML. Кроме того, для указания размеров изображения могут быть использованы свойства width и height в CSS. Когда одновременно применяются свойства CSS и атрибуты HTML, то свойства CSS будут иметь приоритет над атрибутами HTML. img { height: 200px; width: 200px;}

### Позиционирование изображений🡪По умолчанию изображения позиционируются как строчно-блочные элементы, однако их положение может быть изменено с помощью CSS, в частности, float, display и свойств блочной модели, включая padding, border и margin. Добавив свойство display к изображению и установив его значение как block мы заставим изображение быть блочным элементом. Это отображает изображение на отдельной строке, что позволяет окружающему содержимому располагаться выше и ниже изображения. img { display: block;}/ Для обеспечения пространства вокруг изображения используется свойство margin. Дополнительно можно применить свойства padding, border и background чтобы создать рамку вокруг изображения по желанию.

## Добавление аудио🡪HTML5 предлагает быстрый и простой способ добавить [аудиофайлы](https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_HTML5_audio_and_video) на сайт через [элемент](https://webref.ru/html/audio) <audio>. Как и элемент <img>, элемент <audio> принимает URL исходника, указанного в атрибуте src. При этом, в отличие от элемента <img> элемент <audio> требует открывающий и закрывающий теги, которые мы обсудим в ближайшее время. <audio src="jazz.ogg"></audio>

### Атрибуты <audio>🡪Несколько других атрибутов могут сопровождать атрибут src для элемента <audio>, наиболее популярные это autoplay, controls, loop и preload. Атрибуты autoplay, controls и loop — логические атрибуты и не требуют наличия значения. Вместо этого, когда каждый присутствует в элементе <audio>, значение атрибута будет установлено истинным и элемент <audio> будет вести себя соответственно. По умолчанию, элемент не отображается на странице. Для отображения элемента <audio> на странице необходим логический атрибут controls. Когда он применяется к элементу <audio> браузер покажет элементы управления по умолчанию, включая воспроизведение, паузу, поиск и регулировку громкости. <audio src="jazz.ogg" controls></audio> При наличии логического атрибута loop для элемента <audio> аудиофайл будет повторяться постоянно, с начала и до конца.

### Альтернатива аудио и несколько источников🡪В настоящее время разные браузеры поддерживают различные форматы аудиофайлов, тремя наиболее популярными из которых являются ogg, mp3 и wav. Используя элемент <source> и атрибут src для каждого формата файла, мы можем перечислить один аудиофайл за другим. Используем атрибут type, который поможет браузеру быстро определить, какие типы аудио доступны. <audio controls> <source src="jazz.ogg" type="audio/ogg"> <source src="jazz.mp3" type="audio/mp3"> Пожалуйста, <a href="jazz.mp3" download>скачайте</a> аудиофайл.</audio>

## Добавление видео🡪Добавление [видео в HTML5](http://dev.opera.com/articles/introduction-html5-video/) очень похоже на добавление аудио. Мы используем элемент <video> на месте элемента <audio>. Все те же атрибуты (src, autoplay, controls, loop и preload) и альтернативы применимы и здесь. <video src="earth.ogv" controls></video>

**23.HTML. Формы и поля ввода.**

HTML — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора.

HTML формы используются для передачи данных на сервер. Форма может содержать элементы ввода: текстовые поля, флажки, переключатели, кнопки подтверждения и другие. Форма также может содержать следующие элементы: списки выбора, текстовую область, группу полей, легенду, и метки. Тег <form> используется для создания HTML формы: <form> элементы ввода </form>

## HTML Формы - Элемент Ввода

Наиболее важный элемент формы - элемент ввода. Элемент ввода используется для ввода пользовательской информации. Элемент ввода может меняться различными способами, в зависимости от атрибута type. Элемент ввода может иметь тип: текстовое поле, флажок, пароль, переключатель, кнопка подтверждения, и др.

Основной элемент форм в HTML — поле для ввода. Оно реализуется с помощью тега <input> и позволяет задать различные поля для ввода данных: текст, пароли, чекбоксы, радио кнопки, кнопки отправки, загрузка файла, ввод даты и так далее. Самый простой способ использования <input> — создание текстового поля для ввода. Именно <input> присутствует во всех полях поиска. Чтобы создать простое текстовое поле для ввода данных, необходимо внутрь формы добавить непарный тег <input> и указать атрибут type со значением text: <form> <input type="text"> </form>

В стандарте HTML5 появилось ещё несколько интересных типов для тега <input>. Например: email, number, search, tel, url. Их назначение в том, что значения автоматически проверяются браузером на корректность. <input type="password" /> определяет поле ввода пароля:

## Переключатели 🡪<input type="radio" /> определяет радио кнопку. Радио кнопки позволяют пользователю выбирать ТОЛЬКО ОДИН из ограниченного числа вариантов:

## <form> <input type="radio" name="sex" value="male" /> Мужской<br /> <input type="radio" name="sex" value="female" /> Женский </form>

## Флажки🡪<input type="checkbox" /> определяет флажок. Флажки позволяют пользователю выбрать ОДНУ или БОЛЕЕ опций из ограниченного числа вариантов. <form> <input type="checkbox" name="vehicle" value="bike" /> У меня есть мотоцикл <br /> <input type="checkbox" name="vehicle" value="car" /> У меня есть автомобиль </form>

## Кнопка Подтверждения/Отправки🡪<input type="submit" /> определяет кнопку подтверждения (отправки). Кнопка подтверждения используется для отправки данных на сервер. Данные отправляются на страницу указанную в атрибуте формы action. Файл указанный в атрибуте action обычно делает что либо с полученными данными: <form name="input" action="http://Uroki-HTML.ru/html\_form\_action.php" method="get"> Имя пользователя: <input type="text" name="user" /> <input type="submit" value="Отправить" /> </form>

Если вы напечатаете некоторые символы в текстовом поле сверху и кликните на кнопке "Отправить", браузер пошлет введенные вами данные на страницу "html\_form\_action.php". Эта страница покажет вам то, что вы ввели.

<form>🡪Определяет форму для ввода пользовательских данных |||<input>🡪Определяет элемент ввода /// <textarea>🡪Определяет многострочную текстовую область ввода |||<label>🡪Определяет метку (подпись) для элемента ввода \\\ <fieldset>🡪Определяет границу вокруг элементов ввода формы |||<legend>🡪Определяет заголовок для элемента группировки fieldset \\\ <select>🡪Определяет список выбора (раскрывающийся список) ||| <optgroup>🡪Определяет группу связанных опций в списке выбора

<option>🡪Определяет опцию в списке выбора ||| <button>🡪Определяет кнопку

**24.HTML. DOM. Дерево элементов. Основные понятия.**

HTML — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора. Браузер, когда запрашивает страницу и получает в ответе от сервера её исходный HTML-код, должен сначала его разобрать. В процессе анализа и разбора HTML-кода браузер строит на основе него **DOM-дерево**. После выполнения этого действия и ряда других браузер приступает к отрисовке страницы. В этом процессе он, конечно, уже использует **DOM-дерево**, а не исходный HTML-код. **DOM – это объектная модель документа, которую браузер создаёт в памяти компьютера на основании HTML-кода, полученного им от сервера.**

HTML-код – это текст страницы, а DOM – это набор связанных объектов, созданных браузером при парсинге её текста. Связи между объектами в модели формируются на основании того, как **HTML-элементы расположены в коде относительно друг друга**. При этом DOM документа после его формирования можно **изменять**. При изменении DOM браузер практически мгновенно перерисовывает изображение страницы. В результате **отрисовка страницы всегда соответствует DOM**. Для чтения и изменения DOM программно браузер предоставляет нам **DOM API** или, другими словами, программный интерфейс. По-простому **DOM API** – это набор огромного количества различных объектов, их свойств и методов, которые мы можем использовать для **чтения и изменения DOM**. Для работы с DOM используется JavaScript, т.к. на сегодняшний день это единственный язык программирования, скрипты на котором могут выполняться в браузере. **DOM API** нужен для того, чтобы мы могли с помощью JavaScript изменять страницу на «лету», т.е. делать её динамической и интерактивной.

Исходный код веб-страницы состоит из тегов, атрибутов, комментариев и текста. Теги — это базовая синтаксическая конструкция HTML. Одна такая пара тегов образует HTML-элемент. HTML-элементы могут иметь дополнительные параметры – атрибуты. В документе для создания определённой разметки одни элементы находятся внутри других. В результате HTML-документ можно представить как множество вложенных друг в друга HTML-элементов.

## Как строится DOM-дерево документа

Как уже было описано выше браузер строит дерево на основе HTML-элементов и других сущностей исходного кода страницы. При выполнении этого процесса он учитывает вложенность элементов друг в друга. В результате браузер полученное DOM-дерево использует не только в своей работе, но также предоставляет нам API для удобной работы с ним через JavaScript. При строительстве DOM браузер создаёт из HTML-элементов, текста, комментариев и других сущностей этого языка объекты (узлы DOM-дерева). В большинстве случаев веб-разработчиков интересуют только объекты (узлы), образованные из HTML-элементов. При этом браузер не просто создаёт объекты из HTML-элементов, а также связывает их между собой определёнными связями в зависимости от того, как каждый из них относится к другому в коде. Элементы, которые находятся непосредственно в некотором элементе являются по отношению к нему детьми. А он для каждого из них является родителем. При этом в HTML любой элемент всегда имеет одного родителя (HTML-элемент, в котором он непосредственно расположен). В HTML у элемента не может быть несколько родителей. Исключение составляет только элемент html. У него нет родителя. Чтобы получить DOM-дерево так как его строит браузер, необходимо просто «выстроить» все элементы в зависимости от их отношения друг к другу.

Создание DOM-дерева выполняется сверху вниз. При этом корнем DOM-дерева всегда является сам документ (узел document). Далее дерево строится в зависимости от структуры HTML кода.

|  |  |  |
| --- | --- | --- |
| Интерфейс | Описание | Дети |
|  | Документ | Element, comment, documentType |
|  | Фрагмент документа | Element, comment, text, entityReference |
| DocumentType | Тип документа | нет |
| entityReference | Ссылка на раздел | Element, progressingInstruction, text, entityReference |
| Element | Элемент | Element, progressingInstruction, text, entityReference |
| Attr | Атрибут | text, entityReference |
| text | Текст | Нет |
| entity | Раздел | Element, progressingInstruction, text, entityReference |
| Notation | Нотация | Нет |

**25.CSS. Понятие таблицы стилей, назначение, общая характеристика языка.**

Каскадные таблицы стилей - CSS (Cascade Style Sheets) — представляет собой набор параметров для форматирования, которые используются для управления видом и положением элементов web – страницы сайта. Термин *каскадные* указывает на возможность слияния различных видов стилей и на наследование стилей внутренними тегами от внешних. CSS – это язык, содержащий набор свойств для определения внешнего вида документа. Cпецификация CSS определяет свойства и описательный язык для установления связи с HTML-элементами. CSS – абстракция, в которой внешний вид Web-документа определяется отдельно от его содержания.

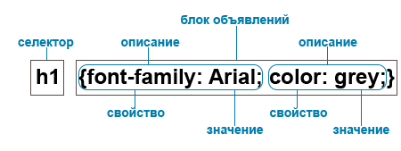
**Гибкое размещение информации о стиле:** Помещение таблиц стилей в отдельные файлы упрощает их повторное использование. Иногда полезно включать инструкции по представлению в документ, к которому они применяются, в начало документа или в атрибуты элементов в теле документа.

**Независимость от языков таблиц стилей:** Данная спецификация не привязывает HTML к конкретному языку таблиц стилей. Это позволяет использовать широкий диапазон таких языков, например, простые языки для большинства пользователей и более сложные для более специализированных случаев.

**Зависимость от устройств:** Таблицы стилей применяются к конкретным устройствам или группам устройств. Таблица стилей, предназначенная для экрана, может применяться при печати, но бесполезна для речевых браузеров.

**CSS (Cascading Style Sheets)** — язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, документам HTML и приложениям XML). Отделяя стиль представления документов от содержимого документов, CSS упрощает создание веб-страниц и обслуживание сайтов.

Объявление стиля состоит из двух частей: **селектора** и **объявления**. В HTML имена элементов нечувствительны к регистру. Объявление состоит из двух частей: имя свойства (например, color) и значение свойства (grey). Селектор сообщает браузеру, какой именно элемент форматировать, в блоке объявления (код в {}) перечисляются свойства и их значения.



Внешняя таблица стилей подключается к веб-странице с помощью элемента <link>, расположенного внутри раздела <head></head>. Такие стили работают для всех страниц сайта. <head> <link rel="stylesheet" href="css/assets.css" media="all"> </head>

**Внутренние стили** встраиваются в раздел <head></head> HTML-документа и определяются внутри элемента <style></style>. Внутренние стили имеют приоритет над внешними, но уступают встроенным стилям (заданным через атрибут style). <head> <style>

h1, h2 { color: red; font-family: "Times New Roman", Georgia, Serif; line-height: 1.3em; }

</style> </head> <body> ... </body>

Когда мы пишем **встроенные стили**, мы пишем CSS-код в HTML-файл, непосредственно внутри элемента с помощью атрибута style: <p style="font-weight: bold; color: red;">

**Селекторы** представляют структуру веб-страницы. С их помощью создаются правила для форматирования элементов веб-страницы. Селекторами могут быть элементы, их классы и идентификаторы, а также псевдоклассы и псевдоэлементы.

Соответствует любому HTML-элементу. Например, \* {margin: 0;} обнулит внешние отступы для всех элементов сайта. Также селектор может использоваться в комбинации с псевдоклассом или псевдоэлементом: \*:after {CSS-стили}, \*:checked {CSS-стили}.

Селекторы элементов позволяют форматировать все элементы данного типа на всех страницах сайта. Например, h1 {font-family: Lobster, cursive;} задаст общий стиль форматирования всех h1.

Селекторы класса позволяют задавать стили для одного и более элементов с одинаковым именем класса, размещенных в разных местах страницы или на разных страницах сайта. Селектор идентификатора позволяет форматировать **один** конкретный элемент. Значение id должно быть уникальным, на одной странице может встречаться только один раз и должно содержать хотя бы один символ. Нет никаких других ограничений на то, какую форму может принимать id. <div id="sidebar"></div> #sidebar { width: 300px; float: left;}

При определении стиля можно использовать любую комбинацию селекторов — селектор элемента, псевдокласса элемента, класса или идентификатора элемента.

<div id="wrap" class="box clear"></div>

div {border: 1px solid #eee;} #wrap {width: 500px;} .box {float: left;} .clear {clear: both;}

**26.Способы подключения стилей к веб-страницы. Как выбрать правильный?**

Для подключения стилей на веб-страницу существует несколько способов, которые различаются своими возможностями и назначением.

При использовании связанных стилей описание селекторов и их значений располагается в отдельном файле, как правило, с расширением css, а для связывания документа с этим файлом применяется тег **<link>**. Данный тег помещается в контейнер **<head**>. Пример: **<link** rel="stylesheet" href="mysite.css"**>.** Значение атрибута тега **<link>** — rel остаётся неизменным независимо от кода. Значение href задаёт путь к CSS-файлу, он может быть задан как относительно, так и абсолютно. Таким образом можно подключать таблицу стилей, которая находится на другом сайте. Файл со стилем не хранит никаких данных, кроме синтаксиса CSS. В свою очередь и HTML-документ содержит только ссылку на файл со стилем, т. е. таким способом в полной мере реализуется принцип разделения кода и оформления сайта. Поэтому использование связанных стилей является наиболее универсальным и удобным методом добавления стиля на сайт. Ведь стили хранятся в одном файле, а в HTML-документах указывается только ссылка на него.

При использовании глобальных стилей свойства CSS описываются в самом документе и располагаются в заголовке веб-страницы. По своей гибкости и возможностям этот способ добавления стиля уступает предыдущему, но также позволяет хранить стили в одном месте, в данном случае прямо на той же странице с помощью контейнера **<style>**. Пример: **<style>**  **H1** { font-size: 120%; font-family: Verdana, Arial, Helvetica, sans-serif; color: #333366; }

**</style>**

Внутренний или встроенный стиль является по существу расширением для одиночного тега, используемого на текущей веб-странице. Для определения стиля используется атрибут style, а его значением выступает набор стилевых правил. Пример: **<body> <p** style="font-size: 120%; font-family: monospace; color: #cd66cc"**>**Пример текста**</p> </body>**

Внутренние стили рекомендуется применять на сайте ограниченно или вообще отказаться от их использования. Дело в том, что добавление таких стилей увеличивает общий объём файлов, что ведет к повышению времени их загрузки в браузере, и усложняет редактирование документов для разработчиков.

Все описанные методы использования CSS могут применяться как самостоятельно, так и в сочетании друг с другом. В этом случае необходимо помнить об их иерархии. Первым имеет приоритет внутренний стиль, затем глобальный стиль и в последнюю очередь связанный стиль.

**27.CSS. Селекторы. Виды селекторов.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

**Селекторы — это один из фундаментальных механизмов CSS.** Селекторы — это часть правила CSS, которые позволяют определить к какому элементу веб страницы следует применять правила. **Именно они определяют то, к каким элементам будут применены стили, указанные в фигурных скобках. Синтаксис:** селектор { свойство: значение;

свойство: значение; ...}

Пример: p { text-align: center; font-size: 20px;}

В CSS очень много различных типов селекторов. Используя один из них или комбинацию из нескольких, можно очень точно применить стили к нужным элементам.

К базовым селекторам можно отнести селектор по классу, тегу, идентификатору, атрибуту и универсальный селектор. Селектор по элементу предназначен для выбора элементов по имени тега. Синтаксис: имяТега

Селектор по классу предназначен для выбора элементов по классу (значению атрибута class). Синтаксис: .имяКласса

Селектор по идентификатору предназначен для выбора элемента по идентификатору (значению атрибута id). Синтаксис: #имяИдентификатора

Универсальный селектор (селектор звёздочка) предназначен для выбора всех элементов. Синтаксис: \*

Селекторы по атрибуту предназначены для выбора элементов по имени атрибута и (или) его значению.

Типы селекторов по атрибуту:

[attr] – по имени атрибута;

[attr=value] – по имени и значению атрибута;

[attr^=value] – по имени и значению, с которого оно должно начинаться;

[attr|=value] – по имени атрибута и его значению, которое равно value или начинается со value-;

[attr$=value] – по имени атрибута и значению, на которое оно должно заканчиваться;

[attr\*=value] – по указанному атрибуту и значению, которое должно содержать value;

[attr~=value] – по имени атрибута и значению, которое содержит value отделённое от других с помощью пробела.

**28.CSS. Задание цвета элементов. Способы задания цветов.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

Все способы задания цветов CSS базируются на одном принципе: в файле стилей для селектора указывается свойство color и его значение. А вот сами значения могут задаваться в различном формате.

Один из вариантов задания цветов CSS — шестнадцатиричный формат. Формат этот содержит три шестнадцатиричных числа, которые задают уровень красного, зеленого и синего цветов соответственно.

.main-menu{ color:#ffd76a;}

Следующий формат, при помощи которого задаются цвета CSS — rgb (red, green, blue). Значения чисел в этом формате представляют собой процентное соотношение от 0 до 100%, но наиболее широкое распространение получило представление значения чисел в диапазоне от 0 до 255.

.title{ background-color:rgb(255,176,0); }

Наиболее простым способом задания цветов CSS является определение его через классическое название. Существует 17 определений цветов. Большинство браузеров поддерживают также 117 так называемых svg-цветов, или x11. Их перечень вы можете посмотреть здесь. Задаются такие цвета CSS очень легко. .feature-name{ color:aqwa;}

Еще один формат задания цветов CSS — HSL (hue — оттенок, saturate — насыщенность, lightness — светлота). Оттенок определяется на цветовом круге, задается в градусах от 0 до 359, а насыщенность и светлота задаются в процентах от 0 до 100%.

В RGB и HSL можно добавить также альфа-канал, который будет определять прозрачность цвета, задается он числом в диапазоне от 0 до 1.

.about-title {border-color: rgba (180, 255, 125, 0.6); }

Самым распространенным и известным определением размеров является задание их в пикселях. Базовым размером шрифта, который по умолчанию применяется большинством браузеров, является 16px.

**29.CSS. Задание параметров шрифтов.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

**Шрифт** – это важная составляющая веб-дизайна, придающая сайту узнаваемость и выразительность.

CSS -свойство **font-family** задаёт список приоритетных шрифтов, которые используются для отображения страницы или определённого элемента. В том случае, если на компьютере, с которого производится доступ к веб-сайту, не установлен первый шрифт списка, ищется следующий шрифт и так до тех пор, пока подходящий не будет найден.

Для категоризации используют два типа имён: гарнитура шрифта **family name** и общее семейство **generic family**.

Первый тип — название шрифта (например, *Calibri, Times New Roman* и т. д.), второй — группа шрифтов с характерными общими чертами (например, *sans-serif*).

Указывать шрифты для сайта нужно, начиная с предпочтительного и заканчивая альтернативными вариантами. В конце списка рекомендуется указать родовое имя, поскольку в случае отсутствия определённых шрифтов на компьютере пользователя страница будет отображаться родственными.

Для задания параметров шрифтов в CSS используется свойство **font-family** элемента **font**. Это свойство задает гарнитуру шрифта.  
Все шрифты можно условно разделить на несколько групп:

* **Serif** - шрифты с засечками, например, Times New Roman;
* **Sans-serif** - шрифты рубленые, без засечек, например, Arial;
* **Monospace** - моноширинные шрифты, например, Courier New;
* **Cursive** - курсивные шрифты, например, Calisto MT;
* **Fantasy** - декоративные шрифты, например, Torhok.

**30.CSS. Задание параметров границ элементов.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

Универсальное свойство border позволяет одновременно установить толщину, стиль и цвет границы вокруг элемента. Значения могут идти в любом порядке, разделяясь пробелом, браузер сам определит, какое из них соответствует нужному свойству. Для установки границы только на определенных сторонах элемента, воспользуйтесь свойствами border-top, border-bottom, border-left, border-right.

### Синтаксис: border: [[border-width](http://htmlbook.ru/css/border-width) || [border-style](http://htmlbook.ru/css/border-style) || [border-color](http://htmlbook.ru/css/border-color)] | inherit

Значение border-width определяет толщину границы. Для управления ее видом предоставляется несколько значений border-style. border-color устанавливает цвет границы, значение может быть в любом допустимом для CSS формате. inherit наследует значение родителя.

Для установки границ элементов CSS предоставляет целый набор свойств, позволяющих задать стиль, толщину и цвет границ как по отдельности для каждой из сторон элемента, так и для всех сторон одновременно.

Как и во всех универсальных свойствах, любое персональное значение, которое не указанно, устанавливается в начальное значение. Обратите внимание, border не может быть использован для указания пользовательского значения [border-image (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/border-image" \o "Currently only available in English (US)), но вместо этого устанавливает его в начальное значение, т.е. none.

Пример: border: 1px;

border: 2px dotted;

border: medium dashed green;

**31.CSS. Задание размеров блочных элементов.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

Свойства **[width](https://basicweb.ru/css/css_pr_width.php" \o "CSS свойство width)** и **[height](https://basicweb.ru/css/css_pr_height.php" \o "CSS свойство height)** устанавливают ширину и высоту внутренней области элемента (*content area*), которая может содержать текст, изображения и прочие элементы.

В блочной модели CSS существует разница между значениями ширины и высоты, которые вы задаете для элемента и размером пространства, которое браузер резервирует для его отображения. Общая ширина и высота элементов представляет собой область окна браузера, которая состоит из ширины и высоты внутренних отступов, границ и указанных для них пользовательских значений.

Блочный элемент – этой такой элемент, который занимает всю ширину строки по умолчанию. Блочный элемент может содержать другие блочные или строчные элементы HTML. Визуально блочный элемент можно представить в виде прямоугольника, у которого есть внешние отступы (margin), границы (border) и внутренние отступы (padding), для содержимого, которое в него помещено. Внешние отступы (margin) не влияют на итоговый размер блочного элемента, зато они оказывают влияния на соседние элементы.

**Общая ширина элемента вычисляется по формуле:** [**width**](https://basicweb.ru/css/css_pr_width.php) (ширина) + **[padding-left](https://basicweb.ru/css/css_pr_padding-left.php" \o "CSS свойство padding-left)** (левый внутренний отступ) + **[padding-right](https://basicweb.ru/css/css_pr_padding-right.php" \o "CSS свойство padding-right)** (правый внутренний отступ) + [**border-left**](https://basicweb.ru/css/css_pr_border-left.php) (левая граница) + **[border-right](https://basicweb.ru/css/css_pr_border-right.php" \o "CSS свойство border-right)** (правая граница).

**Общая высота элемента вычисляется по формуле:** [**height**](https://basicweb.ru/css/css_pr_height.php) (высота) + **[padding-top](https://basicweb.ru/css/css_pr_padding-top.php" \o "CSS свойство padding-top)** (верхний внутренний отступ) + **[padding-bottom](https://basicweb.ru/css/css_pr_padding-bottom.php" \o "CSS свойство padding-bottom)** (нижний внутренний отступ) + [**border-top**](https://basicweb.ru/css/css_pr_border-top.php) (верхняя граница) + **[border-bottom](https://basicweb.ru/css/css_pr_border-bottom.php" \o "CSS свойство border-bottom)** (нижняя граница).

Пример: div { width: 150px; /\* устанавливаем ширину элемента \*/

height: 150px; /\* устанавливаем высоту элемента \*/

padding: 10px; /\* устанавливаем внутренние отступы элемента \*/

border: 5px; /\* устанавливаем границы элемента \*/ }

**32.CSS. Задание внутренних и внешних отступов блочных элементов.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

Каждый элемент в CSS заключён в блок, и понимание поведения этих блоков — это ключ к умению задавать раскладку с помощью CSS, то есть выстраивать одни элементы относительно других элементов.

Внешний отступ — это невидимое пространство вокруг элемента. Оно отталкивает другие элементы от него. Внешний отступ может быть как положительным, так и отрицательным. Негативное значение может привести к перекрытию некоторых элементов страницы. Независимо от того, используете ли вы стандартную или альтернативную блочную модель, внешний отступ всегда добавляется после расчёта размера видимого блока.

Мы можем контролировать все поля элемента сразу, используя свойство [margin](https://developer.mozilla.org/ru/docs/Web/CSS/margin), или каждую сторону индивидуально, используя эквивалентные полные свойства: [margin-top](https://developer.mozilla.org/ru/docs/Web/CSS/margin-top); [margin-right](https://developer.mozilla.org/ru/docs/Web/CSS/margin-right); [margin-bottom](https://developer.mozilla.org/ru/docs/Web/CSS/margin-bottom); [margin-left](https://developer.mozilla.org/ru/docs/Web/CSS/margin-left)

Пример: .box {

margin-top: -40px;

margin-right: 30px;

margin-bottom: 40px;

margin-left: 4em;}

Ключевой момент, который нужно понимать в отношении внешних отступов (margin), это концепция схлопывания. Если у вас есть два элемента, внешние отступы которых соприкасаются, и оба значения margin положительные, то эти значения будут объединены в одно, равное большему из двух значений. А если одно или оба значения отрицательны, то сумма отрицательных значений будет вычтена из общей суммы.

Внутренний отступ (padding) расположен между рамкой и областью контента блока. В отличии от внешних отступов (margin) вы не можете использовать отрицательные значения для padding: они должны быть положительными или равными 0. Любой применённый к вашим элементам фон будет отображаться под областью padding, поэтому внутренний отступ обычно используется, чтобы отодвинуть контент от рамок.

Вы можете контролировать значение padding для всех сторон элемента, используя свойство [padding](https://developer.mozilla.org/ru/docs/Web/CSS/padding), или для каждой стороны индивидуально, используя следующие полные свойства: [padding-top](https://developer.mozilla.org/en-US/docs/Web/CSS/padding-top); [padding-right](https://developer.mozilla.org/ru/docs/Web/CSS/padding-right); padding-bottom; [padding-left](https://developer.mozilla.org/ru/docs/Web/CSS/padding-left)

Пример: .box { padding-top: 0;

padding-right: 30px;

padding-bottom: 40px;

padding-left: 4em; }

**34.CSS. Выравнивание и позиционирование блочных элементов.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

CSS рассматривает макет html-документа как дерево элементов. Уникальный элемент, у которого нет родительского элемента, называется **корневым** элементом. Модуль CSS-позиционирование описывает, как любой из элементов может быть размещен независимо от порядка документа (т.е. извлечен из «потока»).

В CSS блок элемента может быть расположен в соответствии с тремя схемами позиционирования:

#### Нормальный поток

Нормальный поток включает блочный контекст форматирования (элементы с display block, list-item или table), строчный (встроенный) контекст форматирования (элементы с display inline, inline-block или inline-table), и относительное и «липкое» позиционирование элементов уровня блока и строки.

#### Обтекание

В обтекающей модели блок удаляется из нормального потока и позиционируется влево или вправо. Содержимое обтекает правую сторону элемента с float: left и левую сторону элемента с float: right.

#### Абсолютное позиционирование

В модели абсолютного позиционирования блок полностью удаляется из нормального потока и ему присваивается позиция относительно содержащего блока. Абсолютное позиционирование реализуется с помощью значений position: absolute; и position: fixed;.

Элементом «вне потока» может быть плавающий, абсолютно позиционированный или корневой элемент.

Свойство position определяет, какой из алгоритмов позиционирования используется для вычисления положения блока. Элемент считается позиционированным, если свойство position имеет значение, отличное от static. Свойство top задает расстояние, на которое верхний край абсолютно позиционированного блока (с учетом его margin) смещается ниже верхнего края содержащего блока. Для относительно позиционированных блоков определяет смещение относительно верхнего края самого блока (то есть блоку задается позиция в нормальном потоке, а затем смещение от этой позиции в соответствии с этим свойством). Свойство right указывает расстояние, на которое правый край абсолютно позиционированного блока (с учетом его margin) смещен влево от правого края содержащего блока. Для относительно позиционированных блоков определяет смещение относительно правого края самого блока. Свойство bottom указывает расстояние, на которое нижний край блока смещен вверх относительно нижнего края содержащего блока. Для относительно позиционированных блоков определяет смещение относительно нижнего края самого блока. Свойство left указывает расстояние, на которое левый край смещен вправо от левого края содержащего блока. Для относительно позиционированных блоков определяет смещение относительно левого края самого блока.

**33.CSS. Единицы измерения размеров. Преимущества и недостатки.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

Многие свойства CSS принимают в качестве своих значений числа, которые задают размеры соответствующих html-элементов. Например, довольно часто возникает необходимость указания ширины или высоты блочных элементов, величины внутренних и внешних отступов, межстрочного интервала и расстояния между соседними буквами, размера шрифта, ширины полей форм или ячеек таблицы, толщины границ и т.д. И все это в свою очередь приводит к необходимости использования соответствующих единиц измерения этих величин. Выделяют два вида единиц измерения: абсолютные, которые являются фиксированными и не зависят от устройства вывода, а также относительные, величина которых зависит от какого-нибудь исходного размера, например размера шрифта родительского элемента.

Пиксель px – это самая базовая, абсолютная и окончательная единица измерения. Количество пикселей задаётся в настройках [разрешения экрана](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0)), один px – это как раз один такой пиксель на экране. Все значения браузер пересчитает в пиксели. Пиксели могут быть дробными, например размер можно задать в 16.5px. **Достоинства:** достоинство пикселя – чёткость и понятность

**Недостатки:** Другие единицы измерения – в некотором смысле «мощнее», они являются относительными и позволяют устанавливать соотношения между различными размерами

При создании адаптивного сайта намного удобнее использовать относительные единицы измерения типа em, а не пиксели для установки размеров текста и отступов внутри и вокруг элементов. Единицы измерения em вычисляются в зависимости от размера шрифта родительского элемента, что позволяет тексту и отступам пропорционально увеличиваться при изменении свойства font-size на родительском элементе. Такие единицы измерения позволяют создавать систему пропорций, в которой изменение значения свойства font-size на одном элементе каскадно влияет на все дочерние элементы. Система пропорций, конечно, хорошо, но в единицах измерения em есть и недостатки.

## Преимущества EM: Очень удобно и быстро меняются размеры всего. Если мы меняем значение элемента то каскадно поменяются все дочерние элементы. Если нужно, то дочерние элементы можно переопределить как обычно, в пикселях. Это очень удобно для изменения размера программно. Например, пользователь может изменить их в настройках. Очень удобно, и сохраняется качество шрифта при масштабировании размеров страницы. Случается так, что пользователи переопределят размер шрифты через браузер. Использование em. позволяет посетителю видеть и использовать масштабируемый текст при любом разрешении и на любом устройстве.

## Недостатки EM: Браузеры округлят не целое значение размера , что не сказывается при больших размерах, но на маленьких деталях можно увидеть неточность. В случае наследование значения em тоже наследуется, это может принести не мало хлопот. Конфликтует с препроцессорами(после компиляции не получиться динамически поменять значениеem ). При использовании transform: scale() элементы перекрывают друг друга.

Проценты %, как и em – относительные единицы. Процент будет от значения свойства родителя с тем же названием, но не всегда. Это очень важная особенность процентов, про которую, увы, часто забывают. Преимущества: Относительность процентов позволяет на порядок уменьшить кол-во действий при адаптивной верстке. Не ограничивает пользователя в настройках браузера. Отлично подходит для построения относительных конструкций и позиционирования объектов. Недостатки: Требуют хорошего знания теории и четкого понимания - процент от чего вычисляется в тот или иной момент.

## W, VH, VMIN, VMAX - относительно окна (вьюпорта) браузера.

Все они так или иначе работают относительно окна браузера, так называемого вьюпорта. Вьюпорт, простыми словами, это вот эта рамка, через которую мы просматриваем сайт. Итак, VW работает относительно ширины вьюпорта, VH относительно высоты вьюпорта. VMIN - относительно наименьшего значения из (VW, VH). VMAX - относительно наибольшего значения из (VW, VH). 100vw равно ширине окна браузера (вьюпорта), 100vh равно высоте окна браузера (вьюпорта). 100wmin равно наименьшему из (vw, vh), 100vmax равно наибольшему из (vw, vh). Преимущества: Относительность VW, VH, VMIN, VMAX позволяет на порядок уменьшить кол-во действий при адаптивной верстке. Не ограничивает пользователя в настройках браузера. Можно привязаться к размерам вьюпорта, что позволяет строить отзывчивые свойства. Недостатки: Сложность в использовании. Нужно обладать хорошими знаниями адаптивной верстки и возможностей CSS. На мобильных устройствах, при работе на всю высоту вьюпорта (100vh), требуются дополнительные действия на JS.

**35.CSS. Способы позиционирования блочных элементов.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

HTML-документ состоит из большого количества элементов, вложенных друг в друга. Чтобы из этих элементов и CSS **построить изображение страницы**, их необходимо как-то в ней расположить. По умолчанию размещение всех элементов на странице осуществляется в **нормальном или базовом потоке**.

Во-первых, вывод элементов на страницу браузер осуществляет **в том порядке, в котором они следуют в HTML коде**. Во-вторых, в коде элементы вложены друг в друга, и чтобы это учитывать при выводе используют так называемые **воображаемые слои для отображения элементов**. При этом слой элемента тем выше (ближе к нам), чем данный элемент является более вложенным в коде, т.е. глубже расположен в нём. В-третьих, положение элемента в потоке зависит от значения свойства display.

Например, элементы, имеющее **блочное отображение** (display: block) отображаются в потоке как прямоугольные области, каждый из них на новой линии друг под другом сверху вниз.

Ширина элементов с блочным отображением по умолчанию равна доступной ширине родительского элемента, т.е. элемента, в который каждый из них непосредственно вложен. Высота их по умолчанию равна такой величине, которой будет достаточно, чтобы отобразить весь контент, который находится в каждом из них.

CSS свойство position — это одно из свойств с помощью которого можно изменить базовое поведение элементов в потоке. Другими словами, данное свойство позволяет «выдернуть» любой элемент из потока документа и разместить его в другом месте относительно окна браузера или других элементов на веб-странице. Свойство position имеет 5 значений: static (статичное позиционирование); relative (относительное); absolute (абсолютное); fixed (фиксированное); sticky (липкое).

Свойство position со значением static элементам назначается **по умолчанию**. Это значение означает что элемент является **не позиционированным**, т.е. отображается как обычно (в потоке). Явная установка элементу CSS-свойства position: static может понадобиться только в том случае, когда нужно переопределить другое значение position установленное элементу. Установка CSS свойств для задания положения элемента left, top, right и bottom никакого влияния на него не оказывают, т.к. его **местонахождение определяется потоком документа**.

Установка относительного позиционирования элементу осуществляется посредством задания ему CSS свойства position: relative. Относительно позиционированный элемент ведёт себя как элемент в потоке за исключением того, что его текущее положение можно при помощи определённых CSS свойств сместить. К этим CSS свойствам относятся left, top, right и bottom. Установка абсолютного позиционирования элементу осуществляется посредством задания ему position: absolute. Этот тип позиционирования позволяет разместить элемент именно там, где вы хотите. Позиционирование выполняется относительно ближайшего позиционированного предка.

Задание элементу фиксированного позиционирования осуществляется посредством установки ему position: fixed. Фиксированное позиционирование похоже на абсолютное, но в отличии от него оно **всегда привязывается к краям окна браузера** (viewport), и **остаётся в таком положении даже при скроллинге страницы**. Фиксированное позиционирование применяется для закрепления на странице навигационных меню, кнопки «вверх», панелей с социальными кнопками и многого другого. Относительное позиционирование очень часто используется вместе с абсолютным позиционированием.

**36.CSS. Псевдоклассы и псевдоэлементы.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

Обычно стили CSS применяются к тем элементам веб-страницы, которые видны в ее исходном коде. Но существуют случаи, когда необходимо создать стиль для определенного состояния элемента (например, внешний вид посещенной ссылки либо вид ссылки при наведенном на нее курсоре), а также для элемента, который четко не обозначен в структуре страницы. Примером такого элемента может быть первый символ в абзаце или первая строка.

С помощью псевдоклассов в CSS можно устанавливать стиль для уже существующих элементов веб-страницы, который будет применяться в случае каких-то действий пользователя. Псевдоэлементы же отличаются тем, что могут определять стиль несуществующего содержимого, а также четко не обозначенных элементов.

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-name

Пример: article p:first-child { font-size: 120%; font-weight: bold; }

Они нацелены на какой-то фрагмент вашего документа, находящийся в определённом состоянии, и ведут себя так, как если бы вы добавили класс в свой HTML.

Вот список некоторых псевдоклассов:

:link – этот псевдокласс задает стиль ссылкам, по которым пользователь еще не перешел; :visited – этот же, наоборот, применяет стиль к уже посещенным ссылкам; :hover – определяет стиль элемента, когда на него наведен курсор (может применяться не только к ссылкам); :active – задает стиль активной ссылке (то есть, в момент клика по ней); :focus – применяет стиль к элементу при фокусировке на нем (например, при установке курсора в строку поиска);

:not() – этот полезный псевдокласс позволяет выбрать и стилизовать только те элементы, которые не содержат селектор, указанный в скобках.

Псевдоэлементы ведут себя сходным образом, однако они действуют так, как если бы вы добавили в разметку целый новый HTML-элемент, а не применили класс к существующим элементам. Псевдоэлементы начинаются с двойного двоеточия ::.

::pseudo-element-name

Например, если вы хотите выбрать первую строку абзаца, вы могли бы обернуть её в <span> и использовать селектор элемента; однако это может не сработать, если количество слов, которые вы обернули, будет больше или меньше ширины родительского элемента. Поскольку мы, как правило, не знаем, сколько слов поместится в строке — т.к. их количество меняется, если меняется ширина экрана или размер шрифта — то надёжного решения при помощи HTML нет.

Селектор псевдоэлемента ::first-line сделает это наверняка — если количество слов увеличивается или уменьшается, он всё равно будет выбирать только первую строку. Список некоторых псевдоэлементов:

::after – используется вместе со свойством content и позволяет вывести необходимые данные после содержимого элемента; ::before – выполняет похожую функцию, что и предыдущий, только выводит данные перед содержимым элемента; ::selection – этот псевдоэлемент распознается браузерами только при использовании двух двоеточий и позволяет установить цвет и фон для текста, который выделен пользователем; ::first-letter – используется для изменения стиля первого символа в тексте элемента; ::first-line – используется для изменения стиля первой строки текста элемента.

**37.CSS. Каскадность, наследование, приоритеты стилей.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

Каскадность CSS – это механизм, благодаря которому к элементу HTML-документа может применяться более чем одно правило CSS. Правила могут исходить из различных источников: из внешней и внутренней таблицы стилей, от механизма наследования, от родительских элементов, от классов и ID, от селектора тега, от атрибута style и т. д. Поскольку в этих случаях часто происходит конфликт стилей, была создана система приоритетов: в конечном итоге применяется тот стиль, который исходит от источника с более высоким приоритетом.

Каскадирование стилей – это встроенная особенность CSS, которая заключается в том, что если к элементу применяется сразу несколько стилей, то результирующий стиль будет сформирован из всех присутствующих в этих стилях видов свойств, при чем в случае наличия двух или более одинаковых свойств к элементу будет применено значение свойства с наибольшим в данной ситуации приоритетом.

Все свойства CSS делятся на те, которые наследуются элементами-потомками от своих элементов-предков, и те, которые не наследуются. Так, например, если для параграфа задать границу (сформировать рамку), то все вложенные в него элементы (элементы-потомки) не унаследуют это свойство **CSS**, поскольку в данном случае **наследование** не имеет положительного эффекта. Согласитесь, не совсем разумно было бы переопределять для каждого элемента-потомка унаследованные от абзаца-родителя границы. Но, если мы зададим для абзаца, например, синий цвет шрифта, то он будет унаследован всеми элементами потомками до тех пор, пока мы не переопределим его в случае необходимости для конкретного элемента, а это эффективнее, чем определять шрифт отдельно для каждого элемента-потомка.

В случае равенства специфичности селекторов сравниваемых правил **приоритет** будет определяться по виду стиля, расположению таблиц стилей или же, если они окажутся в одной и той же таблице стилей, сравнению их расположения в коде этой таблицы. Перечислим основные **виды стилей** в порядке возрастания их приоритета.

Стиль браузера обладает наиболее низким приоритетом и применяется по умолчанию к элементам html-кода, к которым не применяется другое форматирование, говоря проще – к «голому» html-коду.

Стиль пользователя устанавливается в настройках браузера пользователем и, по сути, представляет собой измененный пользователем стиль браузера по умолчанию. Так что, если пользователь изменил стиль браузера, то по умолчанию станет применяться стиль пользователя.

Стиль автора устанавливается автором страницы (программистом) и обладает еще большим приоритетом. Именно он нас и будет больше всего интересовать.

Авторский стиль представляет собой результирующий стиль, который формируется из различных видов составляющих его отдельных стилей, опять же, на основе каскадирования. Перечислим их в порядке возрастания приоритета в случае применения к одному и тому же элементу при условии равенства специфичности селекторов.

* Стили, унаследованные элементами-потомками от своих родителей, обладают самым низким приоритетом.
* Стили, во внешних таблицах, обладают большим приоритетом. Правила, которые идут в коде внешней таблицы стилей ниже имеют больший приоритет перед правилами, которые встречаются в коде этой таблицы выше.Из нескольких внешних таблиц стилей большим приоритетом обладают таблицы, подключенные к документу при помощи элемента [«link»](https://okmysite.com/html/html_spravochnik/teg_link/teg_link.html) ниже в коде этого документа.
* **Стили внутренних таблиц** (внутри контейнеров [«style»](https://okmysite.com/html/html_spravochnik/teg_style/teg_style.html)) имеют приоритет перед стилями внешних таблиц, но только если они расположены в коде ниже, чем элемент [«link»](https://okmysite.com/html/html_spravochnik/teg_link/teg_link.html), при помощи которого была подключена внешняя таблица стилей. Если в документе используется несколько внутренних таблиц стилей, т.е. имеется несколько элементов [«style»](https://okmysite.com/html/html_spravochnik/teg_style/teg_style.html), то большим приоритетом обладают стили тех таблиц, которые расположены в коде ниже.
* Еще большим приоритетом обладают **встроенные стили**, которые преобладают над правилами с любой специфичностью селекторов. Действительно, куда уже конкретнее, если стиль расположен в открывающем теге элемента.

**38.CSS. Адаптивная верстка.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

Ранее в веб-разработке актуально было корректное отображение сайта в разных браузерах, включая пресловутый Internet Explorer, знаменитый своими фирменными глюками. Ситуация изменилась: сегодня поисковые системы отдают предпочтение страницам, которые корректно отображаются на любом устройстве независимо от размера дисплея. Поэтому, сайт нужно адаптировать не только под десктопные устройства, но и под мобильные.

Адаптивной версткой сайта называется использование комплекса инструментов, благодаря которому страницы правильно отображаются на экранах любых разрешений. Контент подстраивается под разрешение, ориентацию дисплея устройства и действия пользователя, и гармонично выглядит при любых обстоятельствах.

К преимуществам такого подхода следует отнести: 1) Экономия ресурсов и средств на содержание сайта. Вместо нескольких версий достаточно обеспечить бесперебойную работу одной универсальной. 2) Каждая из версий страницы имеет один и тот же URL, что упрощает SEO продвижение. 3) Параллельно с этим упрощается и реклама в социальных сетях. 4) Улучшается статистика в аналитических отчетах Яндекса и Google — благодаря синхронизации запросов с компьютеров, планшетов и смартфонов она учитывается для единого сайта. 5) Не требуется сложных серверных компонентов. Действующий сайт достаточно один раз модифицировать с помощью CSS стилей, чтобы содержимое страниц адаптировалось под размер дисплея посетившего сайт гаджета. 6) Исключено дублирование контента, что часто наблюдается при параллельном запуске мобильной версии сайта.

Это решение имеет и некоторые недостатки: 1) Сложность адаптации работающего проекта. Проще запустить новый сайт, чем переписать код. 2) Объем страниц у адаптивного сайта больше, чем у обычного: CSS стили и JavaScript дают дополнительные килобайты. Это замедляет загрузку отдельных страниц, что может не понравится, как посетителям, так и поисковым роботам. 3) Сложно размещать качественные иллюстрации с высоким разрешением. Приходится искать обходные пути, используя инструменты наподобие плагина Adaptive Images.

При создании «универсального» дизайна разработчики придерживаются следующих принципов: 1) Соответствие. Более компактная версия сайта оформлением не должна отличаться от объемной. Допускается скрыть лишние элементы с учетом потребностей пользователей различных устройств. Полная версия под разрешение монитора Full HD должна отображать полный функционал. 2) Относительность системы координат. Важно помнить, что изображение размером 480 пикселей выглядит по-разному на экране смартфона и дисплее ноутбука. Размер следует устанавливать по тем объектам, которые есть на экране каждого устройства. Сегодня это верхняя граница экрана, по которой равняются разработчики. 3) Контрольные точки. Размер адаптивных страниц привязывается к контрольным точкам. Сайт будет отображаться в том же виде, пока пользователь не зайдет с другого устройства. 4) Вложение объектов. Если размер одного блока зависит от размера другого, их допускается поместить в общий контейнер для взаимной связи. Прием эффективен в случае с логотипом, кнопками и прочими элементами навигации. 5) Подходящие шрифты. Допускается перевод их в веб формат, чтобы они подстраивались под размер экрана. 6) Адекватные иллюстрации. Если изображение имеет много деталей, его лучше делать растровым. В противном случае можно сделать векторным. Для более быстрой загрузки, картинки рекомендуется сжимать. 7) Юзабилити. Размер элементов, с которыми взаимодействует пользователь, нужно адаптировать для удобного использования. Например, палец по отношению сенсорного экрана имеет большие размеры, чем курсор мыши по сравнению с монитором. Пользователю может быть неудобно нажимать мелкие кнопки или переходить по ссылкам, которые расположены близко друг к другу. Разработчики ориентируются не на конкретные размеры страницы при адаптивной верстке, а на брейкпоинты — контрольные точки, при достижении которых меняется дизайн и соотношение элементов. «Каноническими» считаются следующие размеры дисплея для разных устройств: 1) Смартфоны — 320 пикселей, 480 и выше; 2) Планшеты — 768 пикселей и выше; 3) Нетбуки — 1024 пикселей и выше; 4) Мониторы — 1280 пикселей, 1600, 1920 и выше.

В случае с неадаптивной версткой на разных девайсах содержимое страницы может смещаться или уползать. Брейкпоинты жестко привязывают элементы к ширине экрана: они увеличиваются или уменьшаются, перемещаются в разные места интерфейса, могут появиться новые.

**41.CSS. Препроцессоры: основные понятия, назначение, примеры кода на SASS или LESS.**

**CSS (каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

**CSS препроцессор** (от англ. CSS preprocessor) — это надстройка над CSS, которая добавляет ранее недоступные возможности для CSS, с помощью новых синтаксических конструкций.

Основная задача препроцессора — это предоставление удобных синтаксических конструкций для разработчика, чтобы упростить, и тем самым, ускорить разработку и поддержу стилей в проектах.

CSS препроцессоры преобразуют код, написанный с использованием препроцессорного языка, в чистый и валидный CSS-код.

Основные плюсы — это читабельность кода, структурирование и повышение производительности.

#### CSS — это сложно🡪Стандартный CSS — это сложно. Синтаксис без использования вложенности, которую предлагают CSS-препроцессоры, сложен для зрительного восприятия. Кроме того, нужно помнить имя родителя при вложенности. Отсутствие нормальных переменных и «функций» делает CSS-код грязным и узконаправленным.

#### Доступная документация🡪Прошли те времена, когда документация от препроцессоров была доступна только людям «в теме». Сейчас любой желающий может в кратчайшие сроки освоить любой из препроцессоров, причём с минимальными затратами сил.

#### Простота использования🡪Использовать препроцессоры стало проще, чем раньше, причём намного проще. Для этого нужно лишь установить программу, которая будет следить за файлами, предназначенными для препроцессора, и при их изменении будет компилировать содержимое этих файлов в чистый CSS-код.

#### Структура и логичность кода🡪Самым популярным предлагаемым функционалом любого CSS-препроцессора является возможность вкладывать селекторы друг в друга.

Less🡪Основан в 2009 году Алексис Сельер и написан на JavaScript (изначально был написан на Ruby). Имеет все базовые возможности препроцессоров и даже больше, не имеет условных конструкций и циклов в привычном для нас понимании. Основным плюсом является его простота, практически стандартный для CSS синтаксис и возможность расширения функционала за счёт системы плагинов.

Sass (SCSS)🡪Самый мощный из CSS-препроцессоров. Основан в 2007 году как модуль для HAML и написан на Ruby (есть порт на C++). Имеет больший ассортимент возможностей в сравнении с Less. Возможности самого препроцессора расширяются за счёт многофункциональной библиотеки Compass, которая позволяет выйти за рамки CSS и работать, например, со спрайтами в автоматическом режиме. Имеет два синтаксиса: 1) Sass (Syntactically Awesome Style Sheets) — упрощённый синтаксис CSS, который основан на идентации. Считается устаревшим. 2) CSS (Sassy CSS) — основан на стандартном для CSS синтаксисе.

Stylus🡪 Основан в 2010 году небезызвестной в наших кругах личностью TJ Holowaychuk. удобный и расширяемый препроцессор, он гибче Sass. Написан на JavaScript. Поддерживает много вариантов синтаксиса от подобного CSS до упрощённого (отсутствуют :, ;, {} и некоторые скобки).

Переменные Sass добавляются с символом $, а значение и имя разделяются точкой с запятой, как и свойство CSS.

$mainColor: #0982c1; $siteWidth: 1024px; $borderStyle: dotted;

body { color: $mainColor; border: 1px $borderStyle $mainColor; max-width: $siteWidth; }

LESS переменные точно такие же, как переменные Sass, за исключением того, что имена переменных добавляются символом @

@mainColor: #0982c1; @siteWidth: 1024px; @borderStyle: dotted;

body { color: @mainColor; border: 1px @borderStyle @mainColor; max-width: @siteWidth; }

**39.CSS. Flexbox. Основные понятия, структура, примеры свойств.**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в современном интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке. **CSS flexbox** — модуль макета гибкого контейнера — представляет собой способ компоновки элементов, в основе лежит идея оси. Flexbox состоит из **гибкого контейнера (flex container)** и **гибких элементов (flex items)**. Гибкие элементы могут выстраиваться в строку или столбик, а оставшееся свободное пространство распределяется между ними различными способами.

Модуль flexbox позволяет решать следующие задачи: 1)Располагать элементы в разных направлениях: слева направо, сверху вниз, снизу вверх. 2) Переопределять порядок отображения элементов. 3) определять размеры элементов. 4)Решать проблему с центрированием. 5)Переносить элементы внутри контейнера. 6)Создавать колонки одинаковой высоты.

**Главная ось** — ось, вдоль которой выкладываются flex-элементы. Она простирается в основном измерении. **Main start и main end** — линии, которые определяют начальную и конечную стороны flex-контейнера, относительно которых выкладываются flex-элементы. **Основной размер** — ширина или высота flex-контейнера или flex-элементов, в зависимости от того, что из них находится в основном измерении, определяют основной размер flex-контейнера или flex-элемента. **Поперечная ось** — ось, перпендикулярная главной оси. Она простирается в поперечном измерении.

**Cross start и cross end** — линии, которые определяют начальную и конечную стороны поперечной оси, относительно которых выкладываются flex-элементы. **Поперечный размер** — ширина или высота flex-контейнера или flex-элементов, в зависимости от того, что находится в поперечном измерении, являются их поперечным размером.

Flex-контейнер устанавливает новый гибкий контекст форматирования для его содержимого. Flex-контейнер не является блочным контейнером, поэтому для дочерних элементов не работают CSS-свойства. Также, на flex-контейнер не оказывают влияние свойства column-\*, создающие колонки в тексте. Для управления элементами с помощью этой модели нужно установить свойство display следующим образом: .flex-container { /\*генерирует flex-контейнер уровня блока\*/ display: -webkit-flex; display: flex; }

.flex-container {/\*генерирует flex-контейнер уровня строки\*/

display: -webkit-inline-flex; display: inline-flex; }

После установки данных значений свойства каждый дочерний элемент автоматически становится flex-элементом, выстраиваясь в один ряд.

Flex-элементы — блоки, представляющие содержимое flex-контейнера в потоке. Flex-контейнер устанавливает новый контекст форматирования для своего содержимого, который обуславливает следующие особенности: 1) Для flex-элементов блокируется их значение свойства display. Значение display: inline-block; и display: table-cell; вычисляется в display: block;. 2) Абсолютно позиционированный flex-элемент не участвует в компоновке гибкого макета. 3) Поля margin соседних flex-элементов не схлопываются. 4) Процентные значения margin и padding вычисляются от внутреннего размера содержащего их блока. 5) margin: auto; расширяются, поглощая дополнительное пространство в соответствующем измерении. Их можно использовать для выравнивания или раздвигания смежных flex-элементов.

*Направление главной оси: flex-direction🡪* Управляет направлением главной оси, вдоль которой укладываются flex-элементы, в соответствии с текущим режимом записи. Синтаксис flex-direction: row; flex-direction: row-reverse; flex-direction: column; flex-direction: column-reverse; flex-direction: inherit;

*Управление многострочностью flex-контейнера: flex-wrap🡪*Свойство определяет, будет ли flex-контейнер однострочным или многострочным, а также задает направление поперечной оси, определяющее направление укладки новых линий flex-контейнера. Синтаксис flex-wrap: nowrap; // Значение по умолчанию flex-wrap: wrap; // Flex-элементы переносятся, располагаясь в несколько горизонтальных рядов flex-wrap: wrap-reverse; // переносятся на новые линии, располагаясь в обратном порядке слева-направо

*Краткая запись направления и многострочности: flex-flow*

Свойство позволяет определить направления главной и поперечной осей, а также возможность переноса flex-элементов при необходимости на несколько строк. Представляет собой сокращённую запись свойств flex-direction и flex-wrap. Значение по умолчанию flex-flow: row nowrap;. Свойство не наследуется.

*Порядок отображения flex-элементов: order*

Свойство определяет порядок, в котором flex-элементы отображаются и располагаются внутри flex-контейнера. Применяется к flex-элементам. Первоначально все flex-элементы имеют order: 0;. При указании значения от -1 для элемента он перемещается в начало сроки, значение 1 — в конец.

Flex-элемент будет полностью «негибок», если его значения flex-grow и flex-shrink равны нулю, и «гибкий» в противном случае. Свойство является сокращённой записью свойств flex-grow, flex-shrink и flex-basis. Значение по умолчанию: flex: 0 1 auto;.

*Коэффициент роста: flex-grow🡪* определяет коэффициент роста одного flex-элемента относительно других flex-элементов в flex-контейнере при распределении положительного свободного пространства.

*Базовый размер: flex-basis🡪* устанавливает начальный основной размер flex-элемента до распределения свободного пространства в соответствии с коэффициентами гибкости. Для всех значений, кроме auto и content, базовый гибкий размер определяется так же, как width в горизонтальных режимах записи.

**42.Понятие CSS-фреймворков. Примеры, сравнение, назначение.**

**CSS Framework** — [фреймворк](https://dic.academic.ru/dic.nsf/ruwiki/1171069), созданный для упрощения работы [верстальщика](https://dic.academic.ru/dic.nsf/ruwiki/93583), быстроты [разработки](https://dic.academic.ru/dic.nsf/ruwiki/1115246) и исключения максимально возможного числа ошибок вёрстки (проблемы совместимости различных версий браузеров и т. д.). Как и библиотеки скриптовых [языков программирования](https://dic.academic.ru/dic.nsf/ruwiki/1465), CSS-библиотеки, обычно имеющие вид внешнего css-файла, «подключаются» к проекту (добавляются в заголовок веб-страницы).

Преимущества: Позволяет не искушенному в тонкостях вёрстки программисту или дизайнеру правильно создать HTML-макет. Вёрстка на базе слоёв, а не таблиц; Более быстрая разработка; [Кроссбраузерность](https://dic.academic.ru/dic.nsf/ruwiki/1533739); Возможность использования генераторов кода[[1]](https://dic.academic.ru/dic.nsf/ruwiki/1424052#cite_note-css_framework_code_gen1-1)[[2]](https://dic.academic.ru/dic.nsf/ruwiki/1424052#cite_note-css_framework_code_gen2-2) и [визуальных редакторов](https://dic.academic.ru/dic.nsf/ruwiki/840382); Единообразие кода при работе в команде позволяет снизить число разногласий при разработке

Недостатки: библиотеки имеют неудобные для человека имена селекторов, например, span-1, g-3, grid\_12. Для изучения css-библиотеки требуется некоторое время и усилия. Вместе с библиотекой можно получить ошибки, сделанные ранее её разработчиком. Библиотеки бывают сильно "раздуты" - много лишнего кода, который не будете использовать в своем проекте.

**Фреймворк отличается от библиотеки тем**, что библиотека может быть использована в программном продукте просто как набор подпрограмм близкой функциональности, не влияя на архитектуру программного продукта и не накладывая на неё никаких ограничений. Каркас диктует правила построения архитектуры приложения, задавая на начальном этапе разработки поведение по умолчанию, каркас, который нужно будет расширять и изменять согласно указанным требованиям. Вывод, библиотека является фреймворком, если: задает правила построения архитектуры программного кода; диктует принципы структурирования файлов; внутри себя имеет несколько библиотек.

Для того, чтобы избежать большого количества работы и ускорить процесс верстки веб-страниц, были созданы специальные библиотеки или CSS-фреймворки. По сути, CSS-фреймворк — это просто файл CSS, который вы стандартным образом подключаете к вашей веб-странице. Этот CSS файл просто содержит уже написанный за вас набор стилей, которые вы сможете применять к элементам на вашем веб-сайте, после того как эта библиотека будет подключена к вашей веб-странице. После того, как этот файл фреймворка будет подключен к веб-странице, вы можете добавлять к вашим элементам ни какие-то конкретные стили CSS, а просто добавлять классы. Например, <div class="box"></div>

Добавив класс вы говорите каким образом CSS фреймворк должен оформить тот или иной веб-элемент.  Например, добавив класс box вы оформите какой-то блок div как "коробку с рамкой".

Используя CSS фреймворки, вы мыслите уже не каким-то конкретным кодом, а классами, которые можно добавлять к HTML-элементам. Наиболее популярные CSS фреймворки: Bootstrap и Bulma. И главное, для чего стоит использовать фреймворки - это экономия времени. Добавить для какого-то элемента определенный класс намного проще, чем если вы напишете десятки строк кода. Второе преимущество - стандартизация вашего кода. Если вы хотите сделать кнопку, вы уже знаете какой класс вам для нее нужно присвоить. Это не зависит от того с каким сайтов вы работаете. Везде все работает одинаково.

**40.CSS. Grid. Основные понятия, структура, примеры свойств.**

**CSS (каскадные таблицы стилей)** — язык описания внешнего вида HTML-документа. Это одна из базовых технологий в интернете. Практически ни один сайт не обходится без CSS, поэтому HTML и CSS действуют в единой связке.

[Грид-раскладка (CSS Grid Layout)](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout) представляет собой двумерную систему сеток в CSS. Гриды подойдут и для верстки основных областей страницы, и небольших элементов пользовательского интерфейса. Представляет собой пересекающийся набор горизонтальных и вертикальных линий, образующих колонки и строки. Элементы могут быть помещены в грид в пределах линий этих колонок и строк. Грид имеет следующие особенности: Создание грида с фиксированными размерами полоc, например используя пиксели. Это установит грид на определенный пиксель, соответствующим желаемому макету. Сожно создать грид с гибкими размерами, используя проценты или новую единицу измерения — «fr».

Размещение элементов в заданном месте на гриде используя номера строк, имена или путём привязки к области грида. Можно определить явную сетку с помощью грид-раскладки. Cпецификация грид-раскладки достаточно гибкая, чтобы добавить при необходимости дополнительные строки и колонки. В нее включены такие возможности как, например, добавление «стольких колонок, сколько будет помещено в контейнер». Грид содержит механизм выравнивания, таким образом мы можем контролировать, как элементы выравниваются после размещения в области сетки и как выравнивается вся сетка. В ячейку или область грида может быть помещено несколько элементов; эти элементы могут частично перекрывать друг друга. Такое наложение можно контролировать с помощью свойства [z-index](https://developer.mozilla.org/ru/docs/Web/CSS/z-index).

Все начинается с создания сетки в вашем **грид-контейнере**. Мы создаём grid контейнер, объявляя на элементе display: grid или display: inline-grid. Как сделаем, все прямые потомки этого элемента станут элементами сетки.

В этом примере у меня есть контейнер div с классом-обёрткой и пятью дочерними элементами внутри. <div class="wrapper"> <div>One</div> </div>

Здесь .wrapper является грид-контейнером. .wrapper { display: grid;}

Все прямые потомки теперь являются грид-элементами.

Мы определяем ряды и колонки в нашей сетке при помощи свойств [grid-template-columns](https://developer.mozilla.org/ru/docs/Web/CSS/grid-template-columns) и [grid-template-rows](https://developer.mozilla.org/ru/docs/Web/CSS/grid-template-rows). Грид-трек – это промежуток между любыми двумя линиями грида. грид с тремя колонками шириной по 200px. Каждый дочерний элемент будет располагаться в отдельной ячейке грида.

.wrapper { display: grid; grid-template-columns: 200px 200px 200px;}

Размер треков может быть задан с помощью любой единицы длины. Спецификация также вводит дополнительную единицу длины, позволяющую создавать гибкие (flexible) грид-треки. Новая единица длины «fr» представляет собой долю доступного пространства в грид-контейнере. три одинаковых по ширине трека, расширяющихся и сужающихся в соответствии с доступным пространством. grid-template-columns: 1fr 1fr 1fr;

При задании размеров явного грида или при определении размеров автоматически создаваемых колонок или строк нам может понадобиться задать трекам минимальный размер, но при этом быть уверенными, что они треки растянутся, чтобы вместить весь добавленный в них контент.

В гриде есть решение этой задачи – функция [minmax()](https://developer.mozilla.org/ru/docs/Web/CSS/minmax). В следующем примере я использую minmax() в качестве [grid-auto-rows](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-auto-rows" \o "Currently only available in English (US)). То есть автоматически создаваемые строки будут как минимум 100 пикселей в высоту, а как максимум – примут значение auto. Использование auto означает следующее: высота строки «растягивается» до размера ячейки с самым высоким элементом контента.

Обычные значения, использованные выше, могут быть умещены в одну строку: для колонок – с использованием [grid-column (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-column" \o "Currently only available in English (US)), для строк – с использованием [grid-row (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-row" \o "Currently only available in English (US)). Следующий пример сделает такое же расположение, как и предыдущий, но с менее громоздким кодом CSS. значение до слэша («/») – это первая линия, значение после – последняя линия.

## *Грид-ячейка* – наименьшая часть на гриде. Концептуально она похожа на ячейку таблицы. Как мы видели в предыдущих примерах, едва грид определён для родительского элемента, дочерние элементы автоматически размещаются в каждой ячейке заданного грида. На рисунке ниже я выделил первую ячейку грида. [Грид-области](https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout#%D0%B3%D1%80%D0%B8%D0%B4-%D0%BE%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D0%B8)🡪Элементы могут занимать одну или несколько ячеек внутри строки или колонки, таким образом, создаётся *грид-область*. Грид-области должны быть перпендикулярными – невозможно создать область, например, в форме буквы «L». Выделенная грид-область на рисунке ниже занимает два строчных трека и два колоночных.

**43.Bootstrap. назначение, общая характеристика, примеры работы.**

Bootstrap — это открытый и бесплатный [HTML-](https://blog.skillfactory.ru/glossary/html/), [CSS-](https://blog.skillfactory.ru/glossary/css/) и [JS-фреймворк](https://blog.skillfactory.ru/glossary/javascript/), который используют веб-разработчики для быстрой верстки адаптивных дизайнов сайтов и веб-приложений. Включает в себя CSS- и HTML-шаблоны оформления для веб-форм, меток, типографики, кнопок, блоков навигации и других компонентов веб-интерфейса. Фреймворк используют [frontend-](https://skillfactory.ru/frontend-razrabotchik?utm_source=blog&utm_medium=referral&utm_campaign=bootstrap&utm_content=coding_fr&utm_term=text" \t "_blank), [fullstack-разработчики](https://skillfactory.ru/python-fullstack-web-developer?utm_source=blog&utm_medium=referral&utm_campaign=bootstrap&utm_content=coding_fpw&utm_term=text). Для работы с инструментом нужны минимальные знания верстки, поэтому он подходит новичкам.

Bootstrap используется, когда: у сайта много страниц; страницы собраны из простых базовых элементов — кнопок или таблиц; не будет глобального редизайна; шаблонность страниц окупается скоростью внедрения.

**Основные отличия Bootstrap 4 от Bootstrap 3:** сетка и другие компоненты построены на [флексах](https://itchief.ru/html-and-css/flexbox), а не на float; размеры, указываются в rem и em, не в px; размер шрифта увеличен с 14px до 16px; обновлены почти все компоненты;

Bootstrap требует использования doctype «HTML5». Без него у вас возникнут некоторые проблемы со стилями.

<!DOCTYPE html> ||| <html lang="en"> ||| ... |||</html>

Bootstrap разрабатывался как mobile first, т.е. его настройки прежде всего оптимизированы под мобильные устройства, а уж потом с помощью медиа-запросов мы подгоняем масштаб компонентов как нам необходимо на прочих устройствах. Для большей простоты масштабирования в CSS мы изменяем глобальное значение box-sizing с content-box на border-box.

Система сеток Bootstrap 4 использует контейнеры, ряды и колонки, чтобы удобно располагать содержимое. Бутстрап реализован с помощью [флексбокса](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Using_CSS_flexible_boxes) и полностью адаптивен. Ниже приведен пример и глубокий взгляд на объединение сетки.

<div class="container"> <div class="row"> <div class="col-sm"> Одна из трёх колонок </div> <div class="col-sm"> Одна из трёх колонок </div> <div class="col-sm"> Одна из трёх колонок </div> </div> </div>

Бутстрап использует em и rem для задания большинства размеров, а пиксели px – для «брейкпойнтов» сетки и ширин контейнеров. Это происходит потому, что ширина зоны видимости на каждом устройстве измеряется в пикселях и не изменяется с [размером шрифта](https://drafts.csswg.org/mediaqueries-3/#units).

## Автоматическое расположение с помощью колонок: классы колонок со специальными контрольными точками (например, .col-sm-6) для легкого расположения колонок без использования явных номеров классов.

### Установка ширины одной колонки: Авто-расположение колонок в сетке флексбокса также дает возможность установить ширину одной колонки, при этом остальные «родственные» колонки автоматически изменят свой размер вокруг нее. <div class="container"> <div class="row"> <div class="col"> 1 из 3 </div> <div class="col-6"> 2 из 3 (широкая) </div>

<div class="col"> 3 из 3 </div> </div> </div>

### Содержимое адаптивной ширины: Используйте классы col-{breakpoint}-auto для создания колонок, изменяющих свою ширину по ширине содержимого.

### Мультиряд одинаковой ширины: Создавайте колонки одной ширины, которые охватывают множественные ряды, добавлением .w-100 туда, где необходимо сместить вниз на новую строку. Делайте их адаптивными, применяя .w-100 вместе с некоторыми [адаптивными утилитами отображения](https://bootstrap-4.ru/docs/4.0/utilities/display/).

<div class="row"> <div class="col">col</div> <div class="w-100"></div></div>

### Разрывы колонок: Чтобы сместить колонки на новую строку, нужно: добавьте элемент с width: 100% туда, где вы хотите обернуть ваши колонки новой строкой. В норме это достигается с помощью множественных .row, но не каждый исполнительный метод может это поддержать.

<div class="row"> <div class="col-6 col-sm-3">.col-6 .col-sm-3</div> <div class="col-6 col-sm-3">.col-6 .col-sm-3</div>

<!-- Заставит следующие столбцы переходить на новую строку --> <div class="w-100"></div> </div>

Вложенность: Чтобы вложить ваш контент в сетку по умолчанию, добавьте новый класс .row и набор колонок .col-sm-\* внутри существующей колонки .col-sm-\*. Вложенные ряды должны включать набор колонок, которые добавляется к 12-ти или просто нескольким (не обязательно, чтобы вы использовали все 12 возможных колонок).

<div class="row"> <div class="col-sm-9"> <div class="row"> <div class="col-8 col-sm-6"> </div> <div class="col-4 col-sm-6"> </div> </div> </div> </div>

**44.Общая характеристика языка программирования JavaScript.**

JavaScript — это кроссплатформенный объектно-ориентированный язык сценариев, используемый для создания интерактивных веб-страниц (например, использование сложной анимации, создание нажимаемых кнопок, всплывающего меню и т. п.). Программы на этом языке называются скриптами. Они могут быть написаны прямо в HTML веб-страницы и запускаться автоматически при ее загрузке.

Язык программирования [JavaScript](https://progkids.com/blog/javascript) пользуется довольно высокой популярностью. Связано это с тем, что его можно использовать как для создания динамичности статических веб страниц, так и для написания полноценных приложений SPA и даже компьютерных игр. Первые предпосылки для появления этого языка появились еще в 1992 году, когда была начата разработка скриптового встраиваемого языка Cmm. В 1995 году Брендан Эйх получил задачу внедрить язык программирования в браузер Netscape. Последняя на сегодняшний день версия языка ES6 вышла в 2015 году. Язык придерживается принципа сокращения кода при большей функциональности.

К основным особенностям [JavaScript](https://progkids.com/blog/javascript) языка программирования относятся: 1) Динамическая типизация. Тип данных будет определяться только тогда, когда переменной или const будет присваиваться ее значение. 2) Гибкая работа с функциями. В JS функции можно не только выполнять, но еще и возвращать функции из функций, передавать функции в качестве параметров другим функциями и присваивать функции в качестве значения переменных. 3) JavaScript поддерживается всеми современным браузерами. 4) Объектно-ориентированное программирование. То есть это такая методология программирования, в которой вся программа представляется в виде совокупности объектов

Помимо этого, важной особенностью [JavaScript](https://progkids.com/blog/javascript) является его развитая инфраструктура. На сегодняшний день разработчики могут работать с большим количеством библиотек и фреймворков, несколькими сборщиками, вспомогательными библиотеками и генераторами статических сайтов. Что касается сфер применения язык [JavaScript](https://progkids.com/blog/javascript) широко используется в веб-разработке. Причем работает он в сочетании с HTML и CSS.

Помимо этого, на языке JavaScript можно создавать следующие программы и приложения: Разработка программного обеспечения для мобильных устройств (с помощью React Native). Разработка серверных решений с помощью Node.js.

Преимущества [JavaScript](https://progkids.com/blog/javascript) :Незаменимость при разработке веб сайтов и приложений. Язык легко интегрируется с версткой и сервером. Высокая скорость работы и производительность. Большое количество инструментов и богатая инфраструктура. Относительная простота. Написание программ занимает обычно меньше времени. При этом объем кода также обычно меньше, в сравнении со многими другими языками. Относительная легкость изучения.

Недостатки [JavaScript](https://progkids.com/blog/javascript) : Отсутствует возможность загрузки и чтения файлов. Нестрогая типизация. Уровень безопасности. В такой язык программирования как JS достаточно просто внедрить вредоносный код.

JavaScript является слабо типизированным или динамическим языком. Это значит, что вам не нужно определять тип переменной заранее. Тип определится автоматически во время выполнения программы.

Стандарт ECMAScript определяет 8 типов:

6 типов данных являющихся примитивами: 1) [Undefined](https://developer.mozilla.org/ru/docs/Glossary/undefined) (Неопределённый тип)  : typeof instance === "undefined" 2) [Boolean](https://developer.mozilla.org/ru/docs/Glossary/Boolean) (Булев, Логический тип) : typeof instance === "boolean" 3) [Number](https://developer.mozilla.org/ru/docs/Glossary/Number) (Число) : typeof instance === "number" 4) [String](https://developer.mozilla.org/ru/docs/Glossary/String) (Строка) : typeof instance === "string" 5) [BigInt](https://developer.mozilla.org/ru/docs/Glossary/BigInt)  : typeof instance === "bigint" 6) [Symbol](https://developer.mozilla.org/ru/docs/Glossary/Symbol) (в ECMAScript 6)  : typeof instance === "symbol"

[Null](https://developer.mozilla.org/ru/docs/Glossary/Null) (Null тип ) : typeof instance === "object". Специальный примитив, используемый не только для данных но и в качестве указателя на финальную точку в [Цепочке Прототипов](https://developer.mozilla.org/ru/docs/Web/JavaScript/Inheritance_and_the_prototype_chain);

[Object](https://developer.mozilla.org/ru/docs/Glossary/Object) (Объект) : typeof instance === "object". Простая структура, используемая не только для хранения данных, но и для создания других структур, где любая структура создаётся с использованием ключевого слова [new](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/new): new [Object](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object), new [Array](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array), new [Map (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map" \o "Currently only available in English (US)), new [Set](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Set), new [WeakMap](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/WeakMap), new [WeakSet](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/WeakSet), new [Date](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Date) и множество других структур;

По стандарту Java считается одним из самых простых языков программирования для изучения на сегодня. Язык JavaScript тоже попадает в эту категорию, поскольку он основан на синтаксисе Java. Считается, что **JavaScript**, как более современный, характеризуется более широким применением.

**45.Клиентский JavaScript. Схема работы, назначение, примеры, способы связывания.**

**JavaScript** – это интерпретируемый язык программирования, разработанный для взаимодействия с веб-страницами. JavaScript представляет собой реализацию ECMAScript. **ECMAScript** – это просто описание языка, который определён в стандарте ECMA-262.

В браузерах по умолчанию встроено специальное ПО, называемое интерпретатором JavaScript, чтобы браузер мог выполнять код JavaScript. Как правило, JavaScript называют клиентским языком, подчеркивая, что сценарий исполняется на клиентском компьютере в браузере, не на веб-сервере.

Список того, что позволяет JavaScript: Добавлять различную анимацию; Реагировать на события мыши и клавиатуры; Осуществлять проверку ввода данных в поля формы до отправки на сервер, что снимает доп нагрузку с сервера; извлекать данные о компьютере посетителя; Определять браузер; Изменять содержимое HTML-элементов, добавлять новые теги, изменять стили. Этим конечно же список не ограничивается, так как помимо перечисленного JavaScript позволяет делать и многое другое.

Существуют так же и некоторые ограничения, распространяемые на данный язык: 1) JavaScript не может закрывать окна и вкладки, которые не были открыты с его помощью 2) Не может защитить исходный код страницы и запретить копирование текста или изображений со страницы 3) Не может осуществлять кроссдоменные запросы, получать доступ к веб-страницам, расположенным на другом домене. Даже когда страницы из разных доменов отображаются в одно и тоже время в разных вкладках браузера, код JavaScript принадлежащий одному домену не будет иметь доступа к информации о веб-странице из другого домена. Гарантия безопасности частной информации, которая может быть известна владельцу домена, страница которого открыта в соседней вкладке 4) Нет доступа к файлам, расположенным на компьютере пользователя, и доступа за пределы самой веб-страницы, единственным исключением являются файлы cookie, это небольшие текстовые файлы, которые JavaScript может записывать и считывать

При написании клиентского JavaScript для сайтов или приложений вы не обойдётесь без использования API - то есть интерфейсов для работы с различными функциями браузера или операционной системы, на которой работает сайт, или даже для работы с данными, получаемыми с других сайтов или сервисов. В этом модуле мы изучим, какие это API и как использовать некоторые наиболее распространённые API, которые вам будут часто попадаться во время разработки.

Web-браузеры, такие как Navigator, могут интерпретировать операторы клиентского JavaScript, внедрённые в HTML-страницу. Если браузер (или клиент) запрашивает такую страницу, сервер высылает полное содержимое документа, включая HTML и операторы JavaScript, клиенту по сети. Браузер читает страницу сверху вниз, отображая результирующий HTML и выполняя операторы JavaScript по мере из обнаружения. Операторы клиентского JavaScript, внедрённые в HTML-страницу, могут реагировать на пользовательские события, такие как щелчок мыши, ввод данных в форму и навигация по странице. Например, Вы можете написать функцию JavaScript для проверки правильности, введённой пользователем в форму информации - номера телефона или zip-кода. Без передачи по сети, JavaScript, внедрённый на HTML-страницу, может проверить введённые данные и вывести диалоговое окно, если пользователь ввёл неправильные данные.

Типичным представителем клиентских языков является JavaScript. Все действия и команды, которые мы задаем, скажем, на языке JavaScript, выполняются браузером. Это означает, что один и тот же код, написанный нами, обрабатывается в одном случае браузером Internet Explorer, в другом - Firefox, в третьем - Opera, в четвертом - Google Chrome, т.е. тем обозревателем, который использует каждый конкретный человек для просмотра нашей страницы.

*Клиентский JavaScript* расширяет ядро языка за счёт объектов, управляющих браузером (Navigator или другой подобный web-браузер) и его Document Object Model (DOM). Например, клиентские расширения позволяют приложению размещать элементы на HTML-форме и отвечать на пользовательские события, такие как щелчок мышью, ввод данных в форму и навигация по страницам. Пример: <script> function Hello() { document.getElementById("demo").innerHTML = "Привет, мир!"; } </script>

**46.JavaScript. Инструкции, комментарии, объявление переменных.**

JavaScript — это кроссплатформенный объектно-ориентированный язык сценариев, используемый для создания интерактивных веб-страниц (например, использование сложной анимации, создание нажимаемых кнопок, всплывающего меню и т. п.). Программы на этом языке называются скриптами. Они могут быть написаны прямо в HTML веб-страницы и запускаться автоматически при ее загрузке.

Приложения на JavaScript состоят из инструкций с соответствующим синтаксисом. Одна инструкция может состоять из нескольких строк. На одной строке может находиться несколько инструкций, если они разделены точкой с запятой. Они являются не ключевыми словами, а группами ключевых слов.

Инструкция — это команда для компьютера выполнить что-то. Код на JavaScript — это набор инструкций, которые, как правило, отделяются друг от друга символом ;. Вот пример кода с двумя инструкциями: console.log('Mother of Dragons.'); console.log('Dracarys!');

Кроме кода, в файлах с исходным кодом могут находиться комментарии. Это текст, который не является частью программы и нужен программистам для пометок. С их помощью добавляют пояснения, как работает код, какие здесь ошибки нужно поправить или не забыть что-то добавить позже.

// Удалить строку ниже после реализации задачи по регистрации console.log(10);

Комментарии в JavaScript бывают двух видов: однострочные и многострочные.

Однострочные комментарии начинаются с //. После этих двух символов может следовать любой текст, вся строчка не будет анализироваться и исполняться. Комментарий может занимать всю строчку. Если одной строчки мало, то создаются несколько комментариев: // For Winterfell! // For Lanisters!

Комментарий может находиться на строчке после какого-нибудь кода: console.log('I am the King'); // For Lannisters!

Многострочные комментарии начинаются с /\* и заканчиваются на \*/.

/\* The night is dark and

full of terrors.\*/

console.log('I am the King');

Такие комментарии, обычно, используют для документирования кода, например функций.

Переменная – это просто символьное имя для значения. Переменная даёт возможность обратиться к значению по имени, это означает, что, когда в программе указывается имя переменной, вместо неё подставляется значение.

Прежде чем использовать переменную, её необходимо объявить. Переменные объявляются с помощью ключевого слова var или let, за которым следует имя переменной: var num; let num2;

Один раз использовав ключевое слово var или let, можно объявить несколько переменных, перечислив их через запятую: var num, num2; let num3, num4;

**48.JavaScript. Базовые математические и логические операторы.**

JavaScript — это кроссплатформенный объектно-ориентированный язык сценариев, используемый для создания интерактивных веб-страниц (например, использование сложной анимации, создание нажимаемых кнопок, всплывающего меню и т. п.). Программы на этом языке называются скриптами. Они могут быть написаны прямо в HTML веб-страницы и запускаться автоматически при ее загрузке.

Оператор – это просто внутренняя функция JavaScript. При использовании того или иного оператора мы, по сути, просто запускаем ту или иную встроенную функцию, которая выполняет какие-то определённые действия и возвращает результат.

В JavaScript выделяют следующие **математические операторы**: унарный плюс + и минус; сложение +; вычитание -; умножение \*; деление /; остаток от деления %; возведение в степень \*\*; увеличение значения переменной на 1 ++; уменьшение значения переменной на 1 --.

Примеры: console.log(8 + 4); // 12

console.log(8 % 3); // 2 /// console.log(2 \*\* 3); // 8

let num = 7; console.log(++num); // 8

console.log(--num); // 7

Логические операторы помогают составлять сложные логические условия, которые часто нужны при ветвлении с помощью if...else.

Оператор ! (логическое НЕ) преобразует (если необходимо) значение своего операнда в булево значение, затем инвертирует это значение в противоположное и возвращает полученное в результате инвертирования булево значение: var a = true, b = false; alert(!a); // false alert(!b); // true

Если оператор ! дважды применить к операнду, то значение будет преобразовано в его логический эквивалент:

var a = 1, b = 0; alert(!!a); // true. Тоже самое, что и Boolean(a);

alert(!!b); // false. Тоже самое, что и Boolean(b);

Оператор && (логическое И) сначала вычисляет значение левого операнда. Если левый операнд имеет значение false или значение, которое может быть преобразовано в false, то возвращается значение левого операнда, при этом значение правого операнда вычисляться не будет:

var y = 1; var x = 0 && ++y; // ++y не будет вычисляться

alert("x: " + x); // 0 alert("y: " + y); // 1

Если левый операнд имеет значение true или значение, которое может быть преобразовано в true, оператор переходит к вычислению значения правого операнда, и возвращает значение правого операнда: var x = 1 && 2; var y = 1 && 0; alert("x: " + x); // 2 alert("y: " + y); // 0

Оператор || (логическое ИЛИ) сначала вычисляет значение левого операнда. Если левый операнд имеет значение true или значение, которое может быть преобразовано в true, то возвращается значение левого операнда, при этом значение правого операнда вычисляться не будет:

var y = 1; var x = 1 || ++y; // ++y не будет вычисляться alert("x: " + x); // 1 alert("y: " + y); // 1

Если левый операнд имеет значение false или значение, которое может быть преобразовано в false, оператор переходит к вычислению значения правого операнда, и возвращает значение правого операнда: var x = 0 || 2; var y = null || 0; alert("x: " + x); // 2 alert("y: " + y); // 0

**49.JavaScript. Операции со строками.**

JavaScript — это кроссплатформенный объектно-ориентированный язык сценариев, используемый для создания интерактивных веб-страниц (например, использование сложной анимации, создание нажимаемых кнопок, всплывающего меню и т. п.). Программы на этом языке называются скриптами. Они могут быть написаны прямо в HTML веб-страницы и запускаться автоматически при ее загрузке.

Операции со строками в JavaScript могут быть следующими: объединение строк, обрезка, подсчёт символов, выбор определённого символа. Их вполне достаточно для создания сайтов.

С помощью знака "+" возможно объединять строки: var a = 'Тише, мыши, '; var b = 'кот на крыше'; console.log(a + b); // результат: 'Тише, мыши, кот на крыше'

Объединять можно не только строковые переменные. Можно объединить строку и число.

Функция X.substr(Y, Z) обрезает строку X начиная с символа под номером Y и длиной в Z символов. Если Y положительный, то считается с начала строки, если отрицательный, то с конца. Если Z положительное, то результат будет содержать количество символов равной Z, иначе будет содержать пустую строку. Примеры: var text = 'Вкусный сыр'; var c = text.substr(2, 3); // вернёт "усн"

var d = text.substr(2, -1); // вернёт пустую строку var b = text.substr(-1); // вернёт "р"

var f = text.substr(-2); // вернёт "ыр"

Часто появляется задача определить есть ли в строке определённая подстрока. Для этого используется функция X.indexOf( Y ), которая ищет подстроку Y в строке X. Это подстрока найдена, то возвращается позиция, на которой находится подстрока или "-1", если нет вхождения.

var url = '/store/cheese/all/'; // адрес страницы

if( url.indexOf('/cheese/') != -1 ) { // находим подстроку

console.log('Попробуйте все виды сыра!'); // подстрока найдена }else{ console.log('Магазин'); }

Функция indexOf нашла подстроку '/cheese/' в значении переменной url и вернула положение. В нашем случае получилась цифра "6". А условие if( 6 != -1 ) является правдой, поэтому на экране браузера появится надпись:

Попробуйте все виды сыра!

Для получения определённого символа в строке, существует функция X.charAt( Y ), которая возвращает символ с позиции Y из строки X. Если поставить Y равным нулю, то вернётся первый символ строки. Так же к символам в строке можно обращаться через квадратные скобки, что сокращает запись:

var a = 'Вкусный сыр'; var b = a.charAt(0); // вернёт 'В' var c = a[0]; // вернёт 'В' var d = a[1]; // вернёт 'к' var e = a[5]; // вернёт 'н'

**47.JavaScript. Простые типы данных.**

JavaScript — это кроссплатформенный объектно-ориентированный язык сценариев, используемый для создания интерактивных веб-страниц (например, использование сложной анимации, создание нажимаемых кнопок, всплывающего меню и т. п.). Программы на этом языке называются скриптами. Они могут быть написаны прямо в HTML веб-страницы и запускаться автоматически при ее загрузке.

Типы данных в JavaScript можно разделить на две категории: простые типы и объекты. К категории простых типов в языке JavaScript относятся числа, текстовые строки и логические (или булевы) значения.

**Простые** (их также называют примитивными) типы:

логический (англ. Boolean) — может принимать два возможных значения, иногда называемых истиной (true) и ложью (false);

нулевой (англ. Null) – значение null представляет ссылку, которая указывает, обычно намеренно, на несуществующий или некорректный объект или адрес;

неопределённый (англ. Undefined) – обозначает предопределенную глобальную переменную, инициализированную неопределенным значением;

числовой (англ. Number) – числовой тип данных в формате 64-битного числа двойной точности с плавающей запятой;

строковый (англ. String) – представляет собой последовательность символов, используемых для представления текста;

символ (англ. Symbol​) — тип данных, экземпляры которого уникальны и неизменяемы. (новый в ECMAScript 6).

Логические, или булевы значения (по фамилии их изобретателя — Буля), могут иметь лишь одно из двух значений: true (истина) или false (ложь). Значения true или false обычно появляются в операциях сравнения или логических операциях. Приведенная ниже программа создает булеву переменную, а затем тестирует ее значение с использованием инструкции if/else: <script>  
  var b = true;   if  (b == true)  {     document.write ("Значение – true!");  }  else  {    document.write ("Значение – false!");   } </script>

В JavaScript, нет различия между целым числом и числом с плавающей точкой – по сути, JavaScript представляет все числа в качестве значения с плавающей точкой. Для представления чисел в JavaScript используется 64-битный формат, определяемый стандартом [IEEE-754](http://en.wikipedia.org/wiki/IEEE_754-1985). Число, находящееся непосредственно в коде программы, называется числовым литералом. Помимо десятичных целых литералов JavaScript распознает шестнадцатеричные значения.  
Числа в шестнадцатиричном формате могут включать любую последовательность цифр от 0 до 9 и буквы от a до f, которая обязательно начинается с последовательности символов «0x». var a = 255; var b = 0xFF;

Строковый тип (string) — это неизменяемая, упорядоченная последовательность 16-битных значений, каждое из которых представляет символ Unicode (буквы, цифры, знаки пунктуации, специальные символы и пробелы). Строки могут быть пустыми либо состоять из одного и более символов. Строки создаются при помощи двойных (") или одинарных (') кавычек. В строке, ограниченной парой одинарных кавычек, можно использовать двойные кавычки, и наоборот – одинарные кавычки можно использовать в строке, заключенной в пару двойных кавычек:

var firstName= "Max"; var lastName = 'White';

Нулевой тип (null) содержит единственное специальное значение – null. Ключевое слово null невозможно использовать в качестве имени функции или переменной. Значение null является ссылкой на «пустой» объект и имеет специальное назначение – обычно оно используется для инициализации переменной, которой впоследствии будет присвоено значение. Оператор typeof для значения null возвращает строку «object», что свидетельствует о том, что значение null является специальным «пустым» объектом.

<script>  var car = null;  alert(typeof car); // выведет "object " </script>

Неопределенный тип (undefined) образует свой собственный тип, который содержит единственное специальное значение – undefined. Такое значение имеет переменная, объявленная с помощью оператора var, но не инициализированная:<script>  var car;  alert(car); // выведет "undefined" </script>

Значение undefined возвращается при обращении к переменной, которой никогда не присваивалось значение, а также к несуществующему свойству объекта или элементу массива.

Символ (symbol) является нововведением JavaScript начиная с ECMAScript версии 6. Символ – это уникальное, неизменяемое, примитивное значение, которое служит для создания уникальных идентификаторов.

Чтобы создать символ нужно вызвать функцию Symbol: var mySymbol = Symbol();

**50.JavaScript. Массивы. Создание, использование, основные операции.**

JavaScript — это кроссплатформенный объектно-ориентированный язык сценариев, используемый для создания интерактивных веб-страниц (например, использование сложной анимации, создание нажимаемых кнопок, всплывающего меню и т. п.). Программы на этом языке называются скриптами. Они могут быть написаны прямо в HTML веб-страницы и запускаться автоматически при ее загрузке.

Массив — это упорядоченная коллекция значений. Значения в массиве называются элементами, и каждый элемент характеризуется числовой позицией в массиве, которая называется индексом. Массивы в языке JavaScript являются нетипизированными: элементы массива могут иметь любой тип, причем разные элементы одного и того же массива могут иметь разные типы. Элементы массива могут даже быть объектами или другими массивами, что позволяет создавать сложные структуры данных, такие как массивы объектов и массивы массивов. Массив (Array) в JavaScript является глобальным объектом, который используется для создания массивов; которые представляют собой высокоуровневые спископодобные объекты. Пример создания пустого массива:

// посредством литерала массива const arr = []; // с использованием конструктора Array()

const otherArr = new Array();

Так как индексы нумеруются с 0, то для получение первого, второго и третьего элемента нужно использовать индексы 0, 1 и 2.

const colors = ['black', 'white', 'grey']; console.log( colors[0] ); // 'black'

console.log( colors[1] ); // 'white' console.log( colors[2] ); // 'grey'

При попытке получить доступ к несуществующему элементу возвращается undefined:

console.log( colors[3] ); // undefined

Чтобы изменить элемент, ему нужно просто присвоить новое значение: colors[1] = 'yellow'; // ['black', 'yellow', 'grey']

сли при добавлении элемента случайно пропустить индекс, то в массиве появится неопределенный элемент: colors[5] = 'green'; // ['black', 'yellow', 'grey', 'red', , 'green'] console.log( colors[4] ); // undefined

Определить количество элементов в массиве можно с помощью свойства length: console.log( colors.length ); // 6

Зная количество, получить последний элемент можно так: const lastIndex = colors.length – 1;

console.log( colors[lastIndex] ); // "green"

Найти элемент в массиве можно с помощью метода indexOf(): const disks = ['500Gb', '1Tb', '2Tb'];

const index = disks.indexOf('1Tb'); // 1

В качестве результата он возвращает индекс первого найденного элемента.

Если элемент не найден, то indexOf() возвращает -1.

Метод lastIndexOf() выполняет то же самое что indexOf(), только осуществляет это с конца: const nums = ['One', 'Two', 'Three', 'One'];

nums.lastIndexOf('One'); // 3 nums.lastIndexOf('Two'); // 1

Начиная с версии ECMAScript 7 появился новый метод includes(). Он позволяет проверить содержит ли массив указанный элемент. н похож на indexOf(), но в отличие от него возвращает true или false. [1, 2, 3].includes(2); // true [1, 2, 3].includes(4); // false

Удаление элемента массива с помощью delete делает его неопределённым (пустым). Длина массива при этом не изменяется.

const nums = ['One', 'Two', 'Three', 'Four', 'Five']; delete nums[2]; // ['One', 'Two', , 'Four', 'Five']

Если нужно конкретно удалить элемент из массива, то можно воспользоваться методом splice.

В JavaScript для добавления и удаления элементов имеются следующие методы: push() – для добавления одного или нескольких элементов в конец массива; unshift() – для добавления одного или нескольких элементов в начало массива; pop() – для удаления последнего элемента; shift() – для удаления первого элемента.

**51.JavaScript. Условные операторы.**

Условные операторы - это операторы языка JavaScript (ECMAScript), которые в зависимости от некоторого условия позволяют выполнить одно или несколько определённых инструкций. Формы условных операторов в JavaScript: условный оператор if (с одной ветвью); условный оператор if...else (с двумя ветвями); условный оператор else if... (с несколькими ветвями); тернарный оператор (?:); оператор выбора switch.

Если выражение в условии оператора if не равно true или false, то JavaScript приведёт его к одному из этих значений. Данное действие он выполняет с помощью так называемого "правила лжи". Смысл правила: любое выражение является true, кроме следующих значений: false (ложь); "" или '' (пустая строка); NaN (специальный числовой тип данных «не число»); 0 (число «ноль»); null («пустое» значение); undefined («неопределённое» значение).

Синтаксис оператора if: if (условие) { инструкция}

Условный оператор if состоит из: ключевого слова if; условия (выражения в ()), которое должно равняться true или false (или быть приведено к одному из этих значений); инструкции, которые нужно выполнить, если условие является true или приведено к нему (если инструкция одна, {} необязательны).

Оператор if...else используется, когда необходимо при истинности условия выполнить одни инструкции, а при ложности - другие. Синтаксис: if (условие) { одно или несколько инструкций (будут выполняться, когда условие равно true или приведено к true) } else { одно или несколько инструкций (будут выполняться, когда условие равно false или приведено к false) }

Оператор else if... (несколько условий)

Синтаксис: if (условие1) { инструкции 1 } else if (условие2) { инструкции 2

} else if (условие3) { инструкции 3

//...} else if (условиеN) { инструкции N

} else { инструкции, которые будут выполнены, если ни одно из условий не равно true или не приведёно к этом значению}

Тернарный оператор – оператор JavaScript, который можно использовать, когда необходимо в зависимости от условия выполнить одно из двух заданных выражений. Синтаксис: условие ? выражение1 : выражение2

Тернарный оператор состоит из трех операндов, которые разделяются с помощью символов ? и :. Условие тернарного оператора задаётся в первом операнде. Его также можно заключить в скобки. Если условие равно true или будет приведено к этому значению будет выполнено выражение1, иначе - выражение 2. В JavaScript допустимы множественные тернарные операторы (?:): (dayNumber === 0) ? 'Воскресенье' : (dayNumber === 1) ? 'Понедельник' : (dayNumber === 2) ? 'Вторник'

Оператор switch предназначен для выполнения одного варианта инструкций из нескольких в зависимости от значения выражения. Выбор того или иного варианта определяется посредством строгого равенства результата выражения значению случая (case).

Синтаксис оператора switch: switch (выражение) { case значение1:

// ... инструкции, которые будут выполнены, если результат вычисления выражения равен «значение1»

break; // необязательная инструкция (если её не использовать, то будет выполнена следующая команда оператора switch)

case значение2: // ... инструкции break; // ...

case значениеN: // ... инструкции break; default: // инструкции}

Ключевое слово default является необязательным. Оно используется, когда необходимо задать инструкции, которые нужно выполнить, если результат выражения будет не равен ни одному значению варианта (case).

Инструкция break является необязательной. Предназначена для прерывания выполнения switch и передачи управлению инструкции, идущей после него.

**52. JavaScript. Операторы циклов.**

Операторы цикла выполняют блок кода заданное число раз, либо до тех пор, пока не выполнится заданное условие. Циклы особенно удобны, когда нужно выполнять один и тот же код много раз, каждый раз с разным значением. Операторы предназначенные для организации циклов в JavaScript:•Цикл\_for; •Цикл\_do...while; •Цикл\_while; •for...in; •for...of

Цикл for повторяет действия, пока не произойдёт какое-либо специальное событие завершения цикла. Объявление оператора for выглядит следующим образом: for ([начало]; [условие]; [шаг]) {выражения}

При его выполнении происходит следующее: 1)Выполняется выражение начало, если оно указано. обычно инициализирует один или несколько счётчиков, синтаксис позволяет выражению быть любой сложности. Также используется для объявления переменных. 2) Выполняется условие. Если условие истинно, выполняются выражения. Если ложно, цикл for прерывается. Если же условие полностью пропущено, то оно считается истинным. 3) Выполняются выражения. Чтобы выполнить несколько выражений, используются блок-выражение { ... } для группировки выражений. 4) Обновляется шаг, если он есть, а затем управление возвращается к шагу 2.

Цикл do...while повторяется пока заданное условие истинно. Оператор do...while имеет вид: do { выражения } while (условие);

Выражения выполняются пока условие истинно. Если условие истинно, выражения выполнятся снова. В конце каждого прохода условие проверяется. Если условие ложно, выполнение приостанавливается и управление передаётся выражению после do...while. Цикл while выполняет выражения пока условие истинно. Выглядит он так: while (условие) { выражения}

Условие проверяется на истинность до того, как выполняются выражения в цикле. Если условие истинно, выполняются выражения, а затем условие проверяется снова. Если условие ложно, выполнение приостанавливается и управление переходит к выражению после while. Чтобы в циклах использовать несколько выражений, используйте блок выражение { ... }, чтобы сгруппировать их. Используйте оператор break, чтобы прерывать циклы. Оператор continue используется, чтобы шагнуть на шаг вперёд в циклах.

Оператор for...in проходит по всем перечислимым свойствам объекта. JavaScript выполнит указанные выражения для каждого отдельного свойства. Цикл for...in выглядит так: for (variable in object) { Выражения }

Оператор for...of создаёт цикл, проходящий по перечислимым(итерируемым) объектам (en-US) (включая String, Array, Map, Set, объект arguments (en-US), NodeList и так далее), вызывая на каждой итерации функцию с выражениями, которые надо выполнить для получения значения каждого отдельного свойства.

for (variable of object) { выражения}

Разница между циклами for...of и for...in в том, что тогда как for...in проходит по именам свойств, for...of проходит по значениям свойств.

**55.JavaScript. Литерация объектов. JSON.**

В литеральной нотации объект описывается внутри блока фигурных скобок ({}), как набор разделенных запятой свойств, представленных в виде пар ключ/значение. После имени(ключа) ставится двоеточие. Значение может быть представлено чем угодно. Если в качестве значения используется функция, то такое свойство называется методом.

JSON (JavaScript Object Notation) – это текстовый формат представления данных в нотации объекта JavaScript. Предназначен JSON для обмена данными.

Структура строки JSON практически ничем не отличается от записи JavaScript объекта. Она состоит из набора пар ключ-значения. В этой паре ключ отделяется от значения с помощью знака двоеточия (:), а одна пара от другой - с помощью запятой (,). При этом ключ в JSON, в отличие от объекта обязательно должен быть заключен в двойные кавычки. В объекте ключ (имя свойства) не обязательно следует заключать в двойные кавычки.

Значение в JSON можно задать только в одном из следующих форматов: string, number, object, array, boolean (логическим значением true или false) или null (специальным значением JavaScript). Например, значение ключа в JSON не может быть функцией или датой (объектом типа Date). При этом стоит обратить внимание на то, что JSON не допускает использование внутри своей структуры комментариев. Пример:// объект JavaScript

var person = { name: "Иван", birthDay: new Date(Date.UTC(1985, 03, 05)), getAge: function () { var ageDate = new Date(Date.now() - this.birthDay.getTime());

return Math.abs(ageDate.getUTCFullYear() - 1970); }};

// строка JSON

var jsonPerson = '{"name":"Иван","birthDay":"1985-04-05T00:00:00.000Z"}';

JSON может стать иерархичным и сложным, включая вложенные объекты и массивы. { "name": "Гомэр", "work": { "place": "Атомная станция", "location": "Спрингфилд" }, "children": [ "Барт", "Лиза", "Мэги" ] }

Работа с JSON в JavaScript обычно осуществляется в двух направлениях: перевод строки JSON в объект (данный процесс ещё называют парсингом); конвертирование объекта в строку JSON (другими словами действие обратное парсингу). В JavaScript для парсинга строки в JSON рекомендуется использовать метод JSON.parse: var person = JSON.parse(personData);

Перевод объекта JavaScript в строку JSON осуществляется с помощью метода JSON.stringify. Данный метод осуществляет действие обратное методу JSON.parse: var personString = JSON.strigify(person);

**53.JavaScript. Функции.**

Функция — это самостоятельный блок кода, который можно, один раз объявив, вызывать столько раз, сколько нужно. Функция может, но необязательно, принимать параметры. Функции возвращают единственное значение.

Функции в JavaScript являются объектами типа Function. Их ключевое отличие от обычных объектов, дающее им те исключительные возможности, которыми они обладают, заключается в том, что функции можно вызывать.

Кроме того, функции в JavaScript называют «функциями первого класса» так как их можно назначать переменным, их можно передавать другим функциям в качестве аргументов, их можно возвращать из других функций.

Вот как выглядит объявление функции (function declaration).

function doSomething(foo) { //сделать что-нибудь }

Функцию можно назначить переменной или константе. Такая конструкция называется функциональным выражением (function expression).

const doSomething = function(foo) { //сделать что-нибудь }

Функция назначена константе, но сама она имени не имеет. Такие функции называют анонимными. Подобным функциям можно назначать имена. const doSomething = function doSomFn(foo) { //сделать что-нибудь }

В стандарте ES6 появились стрелочные функции (arrow function), которые особенно удобно использовать в виде так называемых «встроенных функций» (inline function) — в роли аргументов, передаваемых другим функциям (коллбэков). const doSomething = foo => { //сделать что-нибудь }

Если тело стрелочной функции содержит лишь одну команду, результат которой возвращает эта функция, его можно записать без фигурных скобок и без ключевого слова return. const myFunction = (a,b) => a + b

Параметры представляют собой переменные, которые задаются на этапе объявления функции и будут содержать передаваемые ей значения (аргументы). Функции в JavaScript могут либо не иметь параметров, либо иметь один или несколько параметров.

Начиная со стандарта ES6 у функций могут быть так называемые «параметры по умолчанию» (default parameters). Они представляют собой стандартные значения, задаваемые параметрам функций в том случае, если при её вызове значения некоторых параметров не задаются. const doSomething = (foo = 1, bar = 'hey') => { //сделать что-нибудь }

Передаваемые функциям аргументы можно представлять в виде массивов. Для того чтобы разобрать эти аргументы можно воспользоваться оператором, который выглядит как три точки ( «оператор расширения»).

const doSomething = (foo = 1, bar = 'hey') => { //сделать что-нибудь }

const args = [2, 'ho!']

doSomething(...args)

Если функции нужно принимать много параметров, то запомнить порядок их следования может быть непросто. В таких случаях используются объекты с параметрами и возможности по деструктурированию объектов ES6.

const doSomething = ({ foo = 1, bar = 'hey' }) => { //сделать что-нибудь }

const args = { foo: 2, bar: 'ho!' }

doSomething(args)

Все функции возвращают некое значение. Если команда возврата явно не задана — функция возвратит undefined.

Выполнение функции завершается либо после того, как оказывается выполненным весь код, который она содержит, либо после того, как в коде встречается ключевое слово return. Если после ключевого слова return указать некое значение, то это значение возвращается в место вызова функции в качестве результата выполнения этой функции. const doSomething = () => { return 'test' }

const result = doSomething() // result === 'test'

Из функции можно возвращать лишь одно значение. Для того чтобы получить возможность возврата нескольких значений, возвращать их можно либо в виде объекта, используя объектный литерал, либо в виде массива, а при вызове функции применять конструкцию деструктурирующего присваивания. Имена параметров при этом сохраняются.

Однострочные стрелочные функции поддерживают неявный возврат результатов своей работы. const myFunction = () => ({value: 'test'})

const obj = myFunction()

console.log(obj.value) //test

Функции можно объявлять внутри других функций. Область видимости вложенной функции ограничена внешней по отношению к ней функцией, её нельзя вызвать извне. const doSomething = () => {const doSomethingElse = () => {}

doSomethingElse()

return 'test'}

**54.JavaScript. Обработка исключений.**

Исключение - это сигнал, указывающий на возникновение какой-либо исключительной ситуации или ошибки. Возбуждение исключения (throw) - это способ просигнализировать о такой ошибке или исключительной ситуации. Перехватить исключение (catch) - значит обработать его, т.е. предпринять действия, необходимые или подходящие для восстановления после исключения.

В JavaScript исключения возбуждаются в тех случаях, когда возникает ошибка во время выполнения кода и когда программа явно возбуждает его с помощью оператора throw. Исключения перехватываются с помощью операторов try/catch/finally, которые описываются позже.

Оператор throw имеет следующий синтаксис: throw выражение;

Результатом выражения может быть значение любого типа. Оператору throw можно передать число, представляющее код ошибки, или строку, содержащую текст сообщения об ошибке. Интерпретатор JavaScript возбуждает исключения, используя экземпляр класса Error одного из его подклассов, и вы также можете использовать подобный подход. Объект Error имеет свойство name, определяющее тип ошибки, и свойство message, содержащее строку, переданную функции-конструктору.

// Функция факториала числа

function factorial(number)

{// Если входной аргумент не является допустимым значением, возбуждается исключение!

if (number < 0)

throw new Error("Число не может быть отрицательным");

// В противном случае значение вычисляется и возвращается нормальным образом

for (var i = 1; number > 1; i \*= number, number--); /\* пустое тело цикла \*/

return i; }

Когда возбуждается исключение, интерпретатор JavaScript немедленно прерывает нормальное выполнение программы и переходит к ближайшему обработчику исключений. В обработчиках исключений используется оператор catch конструкции try/catch/finally. Если обработчик исключения так и не будет найден, исключение рассматривается как ошибка и о ней сообщается пользователю.

Конструкция try/catch/finally реализует механизм обработки исключений в JavaScript. Оператор try просто определяет блок кода, в котором обрабатываются исключения. За блоком try следует catch с блоком инструкций, вызываемых, если где-либо в блоке try возникает исключение. За оператором catch следует блок finally, содержащий программный код, выполняющий заключительные операции, который гарантированно выполняется независимо от того, что происходит в блоке try.

И блок catch, и блок finally не являются обязательными, однако после блока try должен обязательно присутствовать хотя бы один. Блоки try, catch и finally начинаются и заканчиваются фигурными скобками.

Следующий фрагмент иллюстрирует синтаксис конструкции try/catch/finally: try { // код }

catch (ex) { // код }

finally { // код }

Обратите внимание, что за ключевым словом catch следует идентификатор в скобках. Этот идентификатор похож на параметр функции. Когда будет перехвачено исключение, этому параметру будет присвоено исключение (например, объект Error). В отличие от обычной переменной идентификатор, ассоциированный с оператором catch, существует только в теле блока catch.

Пример: try { // Запросить число у пользователя

var n = Number(prompt("Введите положительное число", ""));

// Вычислить факториал числа, предполагая, что входные данные корректны

var f = factorial(n);

// Вывести результат

alert(n + "! = " + f); }

catch (ex) { // Если данные некорректны, управление будет передано сюда

alert(ex); // Сообщить пользователю об ошибке }

**56.JavaScript. Классы в ES6.**

Класс — это не полностью новая языковая сущность, как может показаться на первый взгляд. В JavaScript класс – это разновидность функции. Базовый синтаксис для классов:class MyClass [extends SuperClass] {prop = value;/свойство

constructor(...) { // конструктор /// super(…) // ... }

[static] method(...) {} // метод

get something(...) {} // геттер

set something(...) {} // сеттер // ...}

Затем используйте вызов new MyClass() для создания нового объекта со всеми перечисленными свойствами и методами. Одним из видов является статический метод — это метод, который вызывается в самом классе, а не в экземпляре класса. Объявляется он с помощью ключевого слова static перед именем метода.

Добавив хэш перед именем переменной, можно легко создать закрытую переменную. Она будет доступна только в классе. Ключевое слово extends используется в объявлениях классов и выражениях классов для создания класса, дочернего относительно другого класса. Если у потомка свой constructor, то, чтобы в нём вызвать конструктор родителя – используется синтаксис super() с аргументами для родителя. При наследовании вызов конструктора родителя осуществляется через super(...args), вызов родительских методов – через super.method(...args).

При вызове new MyClass() автоматически вызывается метод constructor(), в нём мы можем инициализировать объект. Например: class User { constructor(name) { this.name = name; } sayHi() { alert(this.name); }}

// Использование: let user = new User("Иван"); user.sayHi();

Когда вызывается new User("Иван"): •Создаётся новый объект. •constructor запускается с заданным аргументом и сохраняет его в this.name.

Затем можно вызывать на объекте методы, такие как user.sayHi().

Вот что на самом деле делает конструкция class User {...}: •Создаёт функцию с именем User, которая становится результатом объявления класса. Код функции берётся из метода constructor (она будет пустой, если такого метода нет). •Сохраняет все методы, такие как sayHi, в User.prototype.

При вызове метода объекта new User он будет взят из прототипа. Таким образом, объекты new User имеют доступ к методам класса. Важные отличия: •Во-первых, функция, созданная с помощью class, помечена специальным внутренним свойством [[IsClassConstructor]]: true. Поэтому это не совсем то же самое, что создавать её вручную. В отличие от обычных функций, конструктор класса не может быть вызван без new. •Методы класса являются неперечислимыми. Определение класса устанавливает флаг enumerable в false для всех методов в "prototype". •Классы всегда используют use strict. Весь код внутри класса автоматически находится в строгом режиме. Как и функции, классы можно определять внутри другого выражения, передавать, возвращать, присваивать и т.д. Как и в литеральных объектах, в классах можно объявлять вычисляемые свойства, геттеры/сеттеры и т.д.

**60.JavaScript. Стрелочные функции.**

Функция — это самостоятельный блок кода, который можно, один раз объявив, вызывать столько раз, сколько нужно. Функция может, но необязательно, принимать параметры. Функции возвращают единственное значение. Функции в JavaScript являются объектами типа Function. Их ключевое отличие от обычных объектов, дающее им те исключительные возможности, которыми они обладают, заключается в том, что функции можно вызывать. В стандарте ES6 появились стрелочные функции (arrow function), которые особенно удобно использовать в виде так называемых «встроенных функций» (inline function) — в роли аргументов, передаваемых другим функциям (коллбэков). const doSomething = foo => { //сделать что-нибудь }

Выражения стрелочных функций имеют более короткий синтаксис по сравнению с функциональными выражениями и лексически привязаны к значению this. Выражение стрелочных функций не позволяют задавать имя, поэтому стрелочные функции анонимны, если ни к чему не присвоить.

Если тело стрелочной функции содержит лишь одну команду, результат которой возвращает эта функция, его можно записать без фигурных скобок и без ключевого слова return. const myFunction = (a,b) => a + b

Параметры представляют собой переменные, которые задаются на этапе объявления функции и будут содержать передаваемые ей значения (аргументы). Функции в JavaScript могут либо не иметь параметров, либо иметь один или несколько параметров. Начиная со стандарта ES6 у функций могут быть так называемые «параметры по умолчанию» (default parameters). Они представляют собой стандартные значения, задаваемые параметрам функций в том случае, если при её вызове значения некоторых параметров не задаются. const doSomething = (foo = 1, bar = 'hey') => { //сделать что-нибудь }

Передаваемые функциям аргументы можно представлять в виде массивов. Для того чтобы разобрать эти аргументы можно воспользоваться оператором, который выглядит как три точки ( «оператор расширения»).

const doSomething = (foo = 1, bar = 'hey') => { //сделать что-нибудь }

const args = [2, 'ho!'] /// doSomething(...args)

Если функции нужно принимать много параметров, то запомнить порядок их следования может быть непросто. В таких случаях используются объекты с параметрами и возможности по деструктурированию объектов ES6.

const doSomething = ({ foo = 1, bar = 'hey' }) => { //сделать что-нибудь }

const args = { foo: 2, bar: 'ho!' }

doSomething(args)

Все функции возвращают некое значение. Если команда возврата явно не задана — функция возвратит undefined. Выполнение функции завершается либо после того, как оказывается выполненным весь код, который она содержит, либо после того, как в коде встречается ключевое слово return. Если после ключевого слова return указать некое значение, то это значение возвращается в место вызова функции в качестве результата выполнения этой функции. const doSomething = () => { return 'test' }

const result = doSomething() // result === 'test'

Из функции можно возвращать лишь одно значение. Для того чтобы получить возможность возврата нескольких значений, возвращать их можно либо в виде объекта, используя объектный литерал, либо в виде массива, а при вызове функции применять конструкцию деструктурирующего присваивания. Имена параметров при этом сохраняются. Однострочные стрелочные функции поддерживают неявный возврат результатов своей работы. const myFunction = () => ({value: 'test'})

const obj = myFunction() console.log(obj.value) //test

Функции можно объявлять внутри других функций. Область видимости вложенной функции ограничена внешней по отношению к ней функцией, её нельзя вызвать извне. const doSomething = () => {const doSomethingElse = () => {}

doSomethingElse() /// return 'test'}

**57.JavaScript. Функции высших порядков.**

Функции, оперирующие другими функциями — либо принимая их в качестве аргументов, либо возвращая их, называются функциями высшего порядка. Например, можно сделать функцию, создающую новые функции.

function greaterThan(n) { return function(m) { return m > n; }; }

var greaterThan10 = greaterThan(10); console.log(greaterThan10(11)); // → true

Часто используются функции обратного вызова (callback)- это функция, переданная в функцию высшего порядка в качестве аргумента и вызываемая в коде функции высшего порядка ( по завершению какого-либо действия).

function greeting(name) { alert('Hello ' + name);}

function processUserInput(callback) { var name = prompt('Please enter your name.'); callback(name);}

processUserInput(greeting); // greeting - функция обратного вызова

Некоторые функции высшего порядка встроены в язык, например, Array.prototype.map, Array.prototype.filter и Array.prototype.reduce. Метод map() создаёт новый массив, вызывая callback-функцию, указанную в качестве аргумента, для каждого элемента входного массива. Метод map() берёт каждое возвращённое значение от callback-функции и создаёт новый массив, используя эти значения. Callback-функция, отправленная в метод map(), принимает 3 аргумента: element, index, и array.

Метод filter() создаёт новый массив из элементов, которые прошли тест предусмотренный callback-функцией. Эта функция, отправленная методу filter(), принимает 3 аргумента element, index, и array.

Метод reduce() выполняет callback-функцию для каждого элемента вызываемого массива, что в результате приводит к одному выходному значению. Метод reduce принимает два параметра: 1) reducer-функцию (callback), 2) и опционально initialValue. Reducer-функция (callback) принимает 4 параметра: accumulator, currentValue, currentIndex, sourceArray.

Если параметр initialValue предусмотрен, тогда accumulator будет равен initialValue , а currentValue равен первому элементу в массиве. Если параметр initialValue не предусмотрен, тогда accumulator будет равен первому элементу массива, а currentValue – второму.

**58.JavaScript. Наследование.**

JavaScript – это язык, основанный на прототипах. Это значит, что свойства и методы объектов можно повторно использовать посредством общих объектов, которые можно клонировать и расширять. Это называется наследованием прототипов. Таким образом, JavaScript реализует ООП через прототипное наследование. В плане наследования JavaScript работает лишь с одной сущностью: объектами.

Каждый объект имеет внутреннее свойство [[Prototype]], которое представляет собой внутреннюю ссылку на другой объект, называемый его прототипом. У объекта-прототипа также есть свой собственный прототип и так далее до тех пор, пока цепочка не завершится объектом, у которого свойство prototype равно null. По определению, null не имеет прототипа и является завершающим звеном в цепочке прототипов.

Таким образом, когда мы пытаемся получить доступ к свойству или методу, то он сначала ищется в самом объекте. Если его в нём нет, то в прототипе. Когда его нет в прототипе, поиск выполняется в прототипе прототипа и так далее до тех пор, пока указанное имя не будет найдено или не будет достигнут конец цепочки (в этом случае возвращается undefined).

Один из способов создать объект в JavaScript – это использовать метод Object.create(). Прототип создаваемого объекта указывается в первом аргументе этого метода. Для получения и установки прототипа можно воспользоваться методами Object.getPrototypeOf() и Object.setPrototypeOf().

Класс в ES6 не представляет собой новую объектно-ориентированную модель наследования, это просто синтаксический сахар для существующего в JavaScript прототипного наследования. Появился метод constructor, который запускается при создании экземпляра класса. В ES6 теперь можно объявлять статические свойства класса. В классах, как и в обычных объектах, можно объявлять геттеры и сеттеры через get/set.

Для наследования используется директива extends или super(...), таким образом мы вызываем конструктор родителя и можем туда передать аргументы. Если при наследование у дочернего класса нет метода constructor, то автоматически присвоится конструктор родителя.

В ES6 очень удобно стало вызывать родительские методы, раньше необходимо было использовать конструкцию Parent.prototype.method (с использованием прототипа), теперь все стало проще, можно использовать директиву super.

**59.JavaScript. Деструктуризация массивов и объектов.**

Деструктуризация - это разложение сложной структуры (объекта, массива, функции) на простые части (свойства, элементы, составляющие) в целях получения к ним доступа более эффективным способом. Таким образом, деструктуризация (destructuring) позволяет извлечь из объекта или массива отдельные значения в переменные или константы.

Синтаксис деструктуризации без использования объемного (сложного) кода позволяет:•получить доступ к элементам сложных объектов;•управлять вложенными структурами объектов.

Деструктуризация массива

// деструктурирующее присваивание

// записывает firstName=arr[0], surname=arr[1]

let [firstName, surname] = arr;

Ненужные элементы массива могут быть отброшены через запятую: // второй элемент не нужен

let [firstName, , title] = ["Julius", "Caesar", "Consul", "of the Roman Republic"];

Если мы хотим не просто получить первые значения, но и собрать все остальные, то мы можем добавить ещё один параметр, который получает остальные значения, используя оператор «остаточные параметры» – троеточие ("..."):

let [name1, name2, ...rest] = ["Julius", "Caesar", "Consul", "of the Roman Republic"];

Если в массиве меньше значений, чем в присваивании, то ошибки не будет. Отсутствующие значения считаются неопределёнными(undefined).

Если нам необходимо указать значения по умолчанию, то мы можем использовать =: let [name = "Guest", surname = "Anonymous"] = ["Julius"];

Деструктуризация объекта

У нас есть существующий объект с правой стороны, который мы хотим разделить на переменные. Левая сторона содержит «шаблон» для соответствующих свойств. В простом случае это список названий переменных в {...}. Порядок не имеет значения. let options = { title: "Menu", width: 100, height: 200};

let {title, width, height} = options;

Если мы хотим присвоить свойство объекта переменной с другим названием, например, свойство options.width присвоить переменной w, то мы можем использовать двоеточие. Двоеточие показывает «что : куда идёт»:

// { sourceProperty: targetVariable }

let {width: w, height: h, title} = options;

Остаточные свойства и значения по умолчанию реализуются так же, как и в деструктуризации массива.

Если объект или массив содержит другие вложенные объекты или массивы, то мы можем использовать более сложные шаблоны с левой стороны, чтобы извлечь более глубокие свойства. Например:

const customer = { name: 'Sherlock', address: { streetNo: '221b Baker Street', city: 'London', }}

const { name, address: { streetNo, city } } = customer;

**61.JavaScript. Промисы.**

Интерфейс Promise (промис) представляет собой обёртку для значения, неизвестного на момент создания промиса. Он позволяет обрабатывать результаты асинхронных операций так, как если бы они были синхронными: вместо конечного результата асинхронного метода возвращается своего рода обещание (дословный перевод слова "промис") получить результат в некоторый момент в будущем.

Объект Promise создаётся при помощи ключевого слова new и своего конструктора. Конструктор Promise принимает в качестве аргумента функцию, называемую "исполнитель" (executor function). Эта функция должна принимать две функции-колбэка в качестве параметров. Первый из них (resolve) вызывается, когда асинхронная операция завершилась успешно и вернула результат своего исполнения в виде значения. Второй колбэк (reject) вызывается, когда операция не удалась, и возвращает значение, указывающее на причину неудачи, чаще всего объект ошибки.

const myFirstPromise = new Promise((resolve, reject) => { // выполняется асинхронная операция, которая в итоге вызовет: // resolve(someValue); // успешное завершение

// или // reject("failure reason"); // неудача});

Обычно функция executor описывает выполнение какой-то асинхронной работы, по завершении которой необходимо вызвать функцию resolve или reject. Обратите внимание, что возвращаемое значение функции executor игнорируется.

Promise может находиться в трёх состояниях:•ожидание (pending): начальное состояние, не исполнен и не отклонён (результат undefined). •исполнено (fulfilled): операция завершена успешно (после вызова resolve()). •отклонено (rejected): операция завершена с ошибкой (после вызова (reject)).

При создании промис находится в ожидании (pending), а затем может стать исполненным (fulfilled), вернув полученный результат (значение), или отклонённым (rejected), вернув причину отказа. В любом из этих случаев вызывается обработчик, прикреплённый к промису методом then. (Если в момент назначения обработчика промис уже исполнен или отклонён, обработчик всё равно будет вызван, асинхронное исполнение промиса и назначение обработчика не будет происходить в «состоянии гонки», как в случае с событиями в DOM.)

У каждого промиса есть определённый набор методов, которые мы можем использовать: •then – выполняется, когда промис завершился успешно (после вызова функции resolve()); •catch – вызывается, если промис завершается ошибкой (после вызова reject()); •finally – выполняется в любом случае после завершения промиса, вне зависимости от конечного состояния.

Синтаксис: promise

.then((value) => { // ... })

.catch((error) => { // ... })

.finally(() => { // ... });

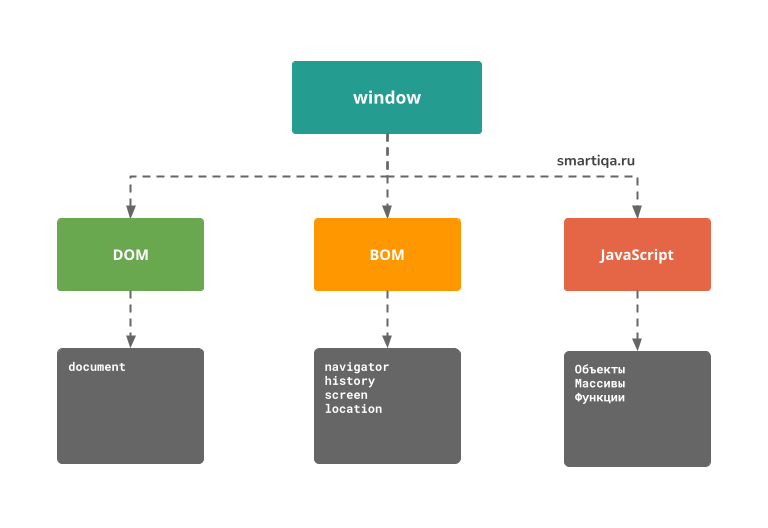
При этом then() позволяет обработать не только успех, но и ошибку. Для этого необходимо передать функцию в качестве второго аргумента (аргумент является необязательным).

promise.then( value => { // действия при успешном завершении промиса },

error => { // действия при завершении промиса с ошибкой } );

**62.DOM. Браузерное окружение в JavaScript.**

Современный JavaScript используется не только в браузерах. Среда, в которой он запускается, будь то браузер, сервер или что-то ещё, называется окружением. У разных окружений разные возможности и функциональность. Окружение предоставляет языку дополнительные возможности и функции. Браузерное окружение даёт возможность работать со страницами сайтов. Представление этих дополнительных возможностей и функциональности в виде объектов, к которым у языка есть доступ — это объектная модель. На картинке ниже представлены объекты браузерного окружения, доступные языку JavaScript.

Помимо возможностей самого языка, мы получаем доступ к ряду объектов: •Объект window (корневой элемент, представляющий собой окно браузера и включающий методы управления им); Позволяет обращаться к переменным и функциям в любом месте программы. Ключевое слово window можно опускать. •Объект DOM (Document object model, объектная модель документа) – совокупность всех элементов веб-страницы в виде дерева; •Объект BOM (Browser object model, объектная модель браузера) – дополнительные инструменты для непосредственной работы с браузером. Можно получить данные о просмотрщике, операционной системе, текущем адресе ресурса, истории посещения страниц и пр.

Спецификация DOM описывает структуру HTML-документа в виде дерева, в котором каждый узел– это объект. DOM нужен для создания и изменения элементов на странице. Все теги, комментарии и текст web-ресурса становятся объектами, с которыми можно работать.

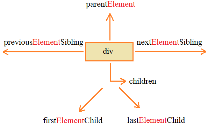
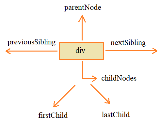
Формально дерево DOM содержит 12 различных типов узлов. В основном, используются такие: •document – объект, представляющий дерево DOM (точка входа в DOM); •узлы-элементы – теги HTML-документа; •текстовые узлы – элементы, содержащие текстовую информацию; •комментарии – элементы с комментариями (иногда они используются JS для записи и чтения информации, скрытой от пользователя).

**63.DOM. Навигация. Способы нахождения элементов. Поиск по дереву.**

DOM позволяет делать что угодно с элементами и их содержимым, но для начала нужно получить соответствующий DOM-объект. Все операции с DOM начинаются с объекта document. Это главная «точка входа» в DOM. Из него мы можем получить доступ к любому узлу. Чтобы браузер успел загрузить весь документ и создать полное DOM-дерево, тег <script> прописывается в конце документа. Если его подключить, например, в разделе head, то переменная body будет равна null (что означает, что объект не существует).

Для навигации по DOM-дереву следует иметь представление о следующих понятиях: •Корнем дерева является узел Document, который представляет документ целиком. •Узел, расположенный непосредственно над данным узлом, называется родительским по отношению к данному узлу. •Узлы, расположенные на один уровень ниже другого узла, являются дочерними по отношению к данному узлу. •Узлы, находящиеся на том же уровне и имеющие того же родителя, называются сестринскими. •Узлы, расположенные на любое число уровней ниже другого узла, являются его потомками. •Родительские, прародительские и любые другие узлы, расположенные выше данного узла, являются его предками.

Относительно любого узла DOM-дерева можно осуществлять следующие переходы(рис 1). Если же нам нужны только узлы-элементы (как правило, это теги HTML-документа), то используется похожий набор свойств(рис 2).



Методы childNodes и children возвращают коллекции. Коллекция – особый перебираемый объект-псевдомассив. Они отражают текущее состояние DOM. Для перебора коллекции мы можем использовать конструкцию for..of: for (let node of nodes) {…}

Свойства навигации по DOM хороши, когда элементы расположены рядом. А что, если нет? Для этого в DOM есть дополнительные методы поиска. •Если у элемента есть атрибут id, то мы можем получить его вызовом document.getElementById(id), где бы он ни находился. •Самый универсальный метод поиска – это elem.querySelectorAll(css), он возвращает все элементы внутри elem, удовлетворяющие данному CSS-селектору. •Метод elem.querySelector(css) возвращает первый элемент, соответствующий данному CSS-селектору. •elem.getElementsByTagName(tag) ищет элементы с данным тегом и возвращает их коллекцию. Передав "\*" вместо тега, можно получить всех потомков. •elem.getElementsByClassName(className) возвращает элементы, которые имеют данный CSS-класс. •document.getElementsByName(name) возвращает элементы с заданным атрибутом name.

Кроме того: •Метод elem.matches(css) проверяет, удовлетворяет ли элемент CSS-селектору. •Метод elem.closest(css) ищет ближайшего по иерархии предка, соответствующему данному CSS-селектору. Сам элемент также включён в поиск. •Метод, который проверяет наличие отношений между предком и потомком: elemA.contains(elemB) вернёт true, если elemB находится внутри elemA (elemB потомок elemA) или когда elemA==elemB.

Все методы "getElementsBy\*" возвращают живую коллекцию. Такие коллекции всегда отражают текущее состояние документа и автоматически обновляются при его изменении.

**64.DOM. Свойства узлов. Изменение свойств элементов.**

Объект DOM (Document object model, объектная модель документа) – совокупность всех элементов веб-страницы в виде дерева. Получив DOM-узел, мы можем перейти к его ближайшим соседям используя следующие свойства: •Для всех узлов: parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling. •Только для узлов-элементов: parentElement, children, firstElementChild, lastElementChild, previousElementSibling, nextElementSibling.

Каждый DOM-узел принадлежит определённому классу. Классы формируют иерархию. Весь набор свойств и методов является результатом наследования.

Главные свойства DOM-узла: •nodeType – Свойство nodeType позволяет узнать тип DOM-узла. Его значение – числовое: 1 для элементов,3 для текстовых узлов, и т.д. Только для чтения. •nodeName/tagName – Для элементов эти свойства возвращают название тега. Для узлов-неэлементов используется nodeName – возвращает название узла. Только для чтения. •innerHTML – Внутреннее HTML-содержимое узла-элемента. Можно изменять. •outerHTML – Полный HTML узла-элемента. Запись в elem.outerHTML не меняет elem. Вместо этого она заменяет его во внешнем контексте. •nodeValue/data – Содержимое узла-неэлемента (текст, комментарий). Эти свойства практически одинаковые. Можно изменять. •textContent – предоставляет доступ к тексту внутри элемента за вычетом всех <тегов>. Запись в него(присваивание) помещает текст в элемент, при этом все специальные символы и теги интерпретируются как текст. Можно использовать для защиты от вставки произвольного HTML кода. •hidden – Когда значение установлено в true, делает то же самое, что и CSS display:none.

Те свойства, которые можно изменять, меняются простым присваиванием.

В зависимости от своего класса DOM-узлы имеют и другие свойства. Например у элементов <input> (HTMLInputElement) есть свойства value, type, у элементов <a> (HTMLAnchorElement) есть href и т.д. Большинство стандартных HTML-атрибутов имеют соответствующие свойства DOM. HTML-атрибуты и свойства DOM не всегда одинаковы.

h1.innerHTML = "Измененный <u>заголовок</u>";

h1.innerHTML += " с добавленным текстом"; (В этом случае сформируется новая строка: "Измененный <u>заголовок</u> с добавленным текстом")

h1.outerHTML = "<h2>Измененный <u>заголовок</u> из h1 в h2";

let msg = prompt("Ваш комментарий:", "");

comm.textContent = msg; comm.hidden = true;

**65.DOM. Браузерные событ Обработчики событий.**

Объект DOM (Document object model, объектная модель документа) – совокупность всех элементов веб-страницы в виде дерева. Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM).

Вот список самых часто используемых DOM-событий:

События мыши: •click – происходит, когда кликнули на элемент левой кнопкой мыши (на устройствах с сенсорными экранами оно происходит при касании). •contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши. •mouseover / mouseout – когда мышь наводится на / покидает элемент. •mousedown / mouseup – когда нажали / отжали кнопку мыши на элементе. •mousemove – при движении мыши.

События на элементах управления: •submit – пользователь отправил форму •focus – пользователь фокусируется на элементе

Клавиатурные события: •keydown и keyup – когда пользователь нажимает / отпускает клавишу.

События документа: •DOMContentLoaded – когда HTML загружен и обработан, DOM документа полностью построен и доступен.

CSS events: •transitionend – когда CSS-анимация завершена.

Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло. Именно благодаря обработчикам JavaScript-код может реагировать на действия пользователя. Есть несколько способов назначить событию обработчик.

Обработчик может быть назначен прямо в разметке, в атрибуте, который называется on<событие>. Например, чтобы назначить обработчик события click на элементе input, можно использовать атрибут onclick. При клике мышкой на кнопке выполнится код, указанный в атрибуте onclick. Удобнее всего указывать функцию, прописанную ранее. Пример: <button onclick=" myFunc()">Нажми меня</button>

Можно назначать обработчик, используя свойство DOM-элемента on<событие>. Тк у элемента DOM может быть только одно свойство с именем onclick, назначить более одного обработчика нельзя. Пример:elem.onclick = myFunc; Убрать обработчик можно назначением elem.onclick = null.

Фундаментальный недостаток описанных выше способов назначения обработчика –- невозможность повесить несколько обработчиков на одно событие. Функция addEventListener() позволяет присваивать элементам более одного обработчика событий, а также работает не только с тегами, но и любыми DOM-объектами. Для удаления используется removeEventListener().

Синтаксис: element.addEventListener(event, handler[, options]); / element.removeEventListener(event, handler[, options]);

event – Имя события, например "click".

handler – Ссылка на функцию-обработчик.

options – Дополнительный объект со свойствами: обычно указывается только значение false, чтобы обработчик автоматически не удалялся после выполнения.

Пример: main = document.documentElement; main.addEventListener('click', one); main.addEventListener('click', two);

Мы можем назначить обработчиком не только функцию, но и объект при помощи addEventListener. В этом случае, когда происходит событие, вызывается метод объекта handleEvent().

Чтобы хорошо обработать событие, могут понадобиться детали того, что произошло. Когда происходит событие, браузер создаёт объект события, записывает в него детали и передаёт его в качестве аргумента функции-обработчику. Некоторые свойства объекта event: event.type – тип события; event.currentTarget – элемент, на котором сработал обработчик; event.clientX / event.clientY – координаты курсора в момент клика относительно окна, для событий мыши.

**66.DOM. Методы обхода дерева элементов.**

Объект DOM (Document object model, объектная модель документа) – совокупность всех элементов веб-страницы в виде дерева.

Для навигации по DOM-дереву следует иметь представление о следующих понятиях: • Корнем дерева является узел Document, который представляет документ целиком. • Узел, расположенный непосредственно над данным узлом, называется родительским по отношению к данному узлу. • Узлы, расположенные на один уровень ниже другого узла, являются дочерними по отношению к данному узлу. • Узлы, находящиеся на том же уровне и имеющие того же родителя, называются сестринскими. • Узлы, расположенные на любое число уровней ниже другого узла, являются его потомками. • Родительские, прародительские и любые другие узлы, расположенные выше данного узла, являются его предками.

Каждый узел документа представлен объектом Node. Есть три различных типа узлов. Корнем дерева является узел Document, который представляет документ целиком. Узлы, представляющие HTML-элементы, являются узлами типа Element, а узлы, представляющие текст, - узлами типа Text. Document, Element и Text - это подклассы класса Node. Document и Element являются двумя самыми важными классами в модели DOM.

Объект Document можно представить как дерево объектов Node. Тип Node определяет свойства, позволяющие перемещаться по такому дереву. Существует еще один прикладной интерфейс навигации по документу, как дерева объектов Element.

Объект Document, его объекты Element и объекты Text, представляющие текстовые фрагменты в документе - все они являются объектами Node. Класс Node определяет следующие важные свойства: • parentNode - Родительский узел данного узла или null для узлов, не имеющих родителя, таких как Document. • childNodes - Доступный для чтения объект, подобный массиву (NodeList), обеспечивающий представление дочерних узлов. • firstChild, lastChild - Первый и последний дочерние узлы или null, если данный узел не имеет дочерних узлов. • nextSibling, previousSibling - Следующий и предыдущий братские узлы. Братскими называются два узла, имеющие одного и того же родителя. Порядок их следования соответствует порядку следования в документе. Эти свойства связывают узлы в двусвязный список. • nodeType - Тип данного узла. Узлы типа Document имеют значение 9 в этом свойстве. Узлы типа Element - значение 1. Текстовые узлы типа Text - значение 3. Узлы типа Comments - значение8 и узлы типа DocumentFragment - значение11. • nodeValue - Текстовое содержимое узлов Text и Comment. • nodeName - Имя тега элемента Element, в котором все символы преобразованы в верхний регистр.

Cвойства объекта Element, аналогичные свойствам доступа к дочерним и братским узлам объекта Node: • firstElementChild, lastElementChild - Похожи на свойства firstChild и lastChild, но возвращают дочерние элементы. • nextElementSibling, previousElementSibling - Похожи на свойства nextSibling и previousSibling, но возвращают братские элементы. • childElementCount - Количество дочерних элементов. Возвращает то же значение, что и свойство children.length.

**87.Express.js. Возврат JSON.**

Express — это простейшая, гибкая и оптимизированная платформа Node.js, упрощающая разработку веб-приложения, которое может обрабатывать различные типы запросов, например GET, PUT, POST и DELETE.

Когда вы слушаете соединения на маршруте в Express, функция обратного вызова будет вызвана при каждом сетевом вызове с экземпляром объекта Request и экземпляром объекта Response.

Пример: app.get('/', (req, res) => res.send('Hello World!'))

Здесь мы использовали метод Response.send(), который принимает любую строку. Вы можете отправить JSON клиенту, используя Response.json(), полезный метод. Он принимает объект или массив и преобразует его в JSON перед отправкой: res.json({ username: 'Flavio' })

Метод json () правильно устанавливает все заголовки для вас и возвращает ответ в формате JSON. JSON (JavaScript Object Notation) – это общий формат для представления значений и объектов. Его описание задокументировано в стандарте RFC 4627. Первоначально он был создан для JavaScript, но многие другие языки также имеют библиотеки, которые могут работать с ним. Таким образом, JSON легко использовать для обмена данными, когда клиент использует JavaScript, а сервер написан на Ruby/PHP/Java или другом языке.

JavaScript предоставляет методы: JSON.stringify для преобразования объектов в JSON. JSON.parse для преобразования JSON обратно в объект.

// Get all members

app.get('/api/members', (req, res) => { res.json(some\_json);});

Следует отметить, что в принципе мы можем отправить объект и с помощью стандартного метода response.send(request.body). В реальности метод response.json() устанавливает для заголовка "Content-Type" значение "application/json", серилизует данные в json с помощью функции JSON.stringify() и затем отправляет данные с помощью response.send().

Res.json заменяет нам:

res.setHeader('Content-Type', 'application/json');

res.send(JSON.stringify(some\_json));

**67.DOM. Программное создание нового элемента и добавление его в дерево.**

Объект DOM (Document object model, объектная модель документа) – совокупность всех элементов веб-страницы в виде дерева. При помощи JavaScript мы можем не только находить элементы в web-документе или менять их свойства, но и вносить изменения в структуру DOM-дерева: создавать новые теги, удалять имеющиеся, заменять их на другие.

Для этого применяются следующие инструменты: •document.createElement(tag) – создаёт новый элемент с заданным тегом •document.createTextNode(text) – создаёт новый текстовый узел с заданным текстом •node.append(...nodes or strings) – добавляет узлы или строки в конец node, •node.prepend(...nodes or strings) – вставляет узлы или строки в начало node, •node.before(...nodes or strings) –- вставляет узлы или строки до node, •node.after(...nodes or strings)–- вставляет узлы или строки после node, •node.replaceWith(...nodes or strings) –- заменяет node заданными узлами или строками. •node.remove() – удаление узла •node.cloneNode(flDeep) – создние копии узла. Если flDeep равен true, то создается «глубокий» клон объекта со всеми его свойствами и дочерними элементами. При значении false получаем клон без дочерних элементов.

Для вставки нескольких элементов данные методы могут принимать сразу более одного аргумента. const paragraph = document.createElement('p');

paragraph.textContent = "I'm a paragraph!"; или paragraph.innerHTML = "I'm a <strong>bold</strong> paragraph!";

const text = document.createTextNode("I'm a text node!"); document.body.prepend(text);

const pModified = p.cloneNode(true); h1.replaceWith(h2)

Приведенные выше способы добавляют текст только безопасным способом (все специальные символы преобразуются. Например, знак > переведется в &gt;). Чтобы получить возможность внедрять в документ строки с тегами, применяется метод insertAdjacentHTML(). Вообще, в JavaScript имеется набор методов insertAdjacent, которые позволяют вставить один или несколько узлов в указанную позицию position относительно node.

Всего существует 3 таких метода: •node.insertAdjacentElement(position, element) – для вставки элемента (element); •node.insertAdjacentHTML(position, htmlString) – для вставки строки (htmlString) как HTML; •node.insertAdjacentText(position, string) – для вставки строки (string);

Значение position, может быть, одним из следующих: •'beforebegin' – непосредственно перед node; •'afterbegin' – перед первым дочерним узлом node (в начало); •'beforeend' – после последнего дочернего узла node (в конец); •'afterend' – непосредственно после node;

p.insertAdjacentHTML('beforebegin',

'<a href="https://www.google.com/"><strong>Google</strong></a>');

**68.Понятие фронтенд фреймворков. Назначение, примеры, использование.**

Фронтенд-фреймворки отвечают за внешний вид веб-приложения. В отличие от серверных, они никак не связаны с логикой работы. Этот тип фреймворков работает в браузере. С их помощью можно улучшать и внедрять новые пользовательские интерфейсы, создавать разные анимации и одностраничные приложения. Вот некоторые из них: •Angular; •Vue.js; •Svelte; •React •Bootstrap

“Bootstrap”🡪Bootstrap ставит в приоритет адаптивный веб-дизайн, упрощая его с помощью сеточной системы и использования классов. В дополнение к этому, данный front end framework предлагает предварительно стилизованные компоненты, состоящие из HTML, CSS и в некоторых случаях JavaScript.

“React”🡪Она предназначена для создания интерактивных интерфейсов пользователя. Её можно использовать для создания посадочных страниц и мобильных приложений с высокой производительностью, простотой использования и масштабируемостью. Одна из особенностей заключается в быстрой работе, что позволяет программистам экономить время разрабатывая при этом серьёзное ПО. Другие преимущества “React” заключаются в:•Доступности повторного использования его компонентов в различных частях web-приложения; •Наличии одностороннего потока информации; •Достаточно крупном сообществе, которое обеспечивает стабильность библиотеки; •Универсальности и возможности применения с другими фреймворками.

“Vue”🡪Vue (произносится /vjuː/, примерно как view) — прогрессивный фреймворк для создания пользовательских интерфейсов. В отличие от фреймворков-монолитов, Vue создавался пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления (view), упрощая интеграцию с другими библиотеками и существующими проектами. С другой стороны, Vue полностью подходит и для разработки сложных одностраничных приложений (SPA, Single-Page Applications), если использовать его в комбинации с современными инструментами и дополнительными библиотеками

**89.REST API. Основные понят, назнач, использован.**

Representational State Transfer (REST) в переводе — это передача состояния представления. Технология позволяет получать и модифицировать данные и состояния удаленных приложений, передавая HTTP-вызовы через интернет или любую другую сеть.

Если проще, то REST API — это когда серверное приложение дает доступ к своим данным клиентскому приложению по определенному URL.

Принципы REST API определены в диссертации его создателя Роя Филдинга. Основные из них: •единый интерфейс; •разграничение клиента и сервера; •нет сохранения состояния; •кэширование всегда разрешено; •многоуровневая система; •код предоставляется по запросу.

Методы HTTP: основа работы REST API🡪Чтобы ресурс, который вы запрашиваете, выполнял нужные действия, используют разные способы обращения к нему. Например, если вы работаете со счетами с помощью ресурса /invoices, который мы придумали выше, то можете их просматривать, редактировать или удалять.

В API-системе четыре классических метода: 1.GET — метод чтения информации. GET-запросы всегда только возвращают данные с сервера, и никогда их не меняют и не удаляют. В бухгалтерском приложении GET /invoices вы открываете список всех счетов. 2.POST — создание новых записей. В нашем приложении POST /invoices используется, когда вы создаете новый счет на оплату. 3.PUT — редактирование записей. Например, PUT /invoices вы исправляете номер счета, сумму или корректируете реквизиты. 4.DELETE — удаление записей. В нашем приложении DELETE /invoices удаляет старые счета, которые контрагенты уже оплатили.

Для чего используют REST API: Архитектура REST API — самое популярное решение для организации взаимодействия между различными программами. Так произошло, поскольку HTTP-протокол реализован во всех языках программирования и всех операционных системах, в отличие от проприетарных протоколов. Чаще всего REST API применяют: 1.Для связи мобильных приложений с серверными. 2.Для построения микросервисных серверных приложений. Это архитектурный подход, при котором большие приложения разбиваются на много маленьких частей. 3.Для предоставления доступа к программам сторонних разработчиков. Например, Stripe API позволяет программистам встраивать обработку платежей в свои приложения.

**74.React.js. Передача параметров как свойств.**

ReactJS - это открытая JavaScript библиотека для создания интерфейсов, которая призвана решать проблемы частичного обновления содержания веб-страницы, с которыми сталкиваются в разработке одностраничных приложений. Компоненты принимают произвольные входные данные, называемые «props» или свойствами. const element = <Welcome name="Сара" />;

Когда React видит элемент, представляющий пользовательский компонент, он передаёт JSX-атрибуты этому компоненту в виде единственного объекта «props». То есть значение из атрибута name перейдет в свойство props.name. Далее в компоненте мы можем работать с этими атрибутами(свойствами).

function Welcome(props) { return <h1>Привет, {props.name}</h1>; }

Данная функция — корректный компонент React, потому что он принимает один аргумент-объект «props» с данными и возвращает элемент React с контретным свойством, взятым из props.

class Welcome extends React.Component { render() { return <h1>Привет, {this.props.name}</h1>; }}

В классовом компоненте для обращения ко свойствам элемента используется ключевое слово this. Однако props не передается в качестве параметра. Независимо от того, объявляете ли компонент как функцию или класс, он не должен изменять свои свойства.

Мы можем определить для свойств значения по умолчанию на тот случай, если их значения не передаются извне. Для установки значений применяется статическое свойство defaultProps. Пример: Welcome.defaultProps = {name: "Сара"};

**69.React.js. Общая характерист. Установка, запуск.**

ReactJS - это открытая JavaScript библиотека для создания интерфейсов, которая призвана решать проблемы частичного обновления содержания веб-страницы, с которыми сталкиваются в разработке одностраничных приложений. Бибилиотека разрабатывается компанией Facebook и сообществом индивидуальных разработчиков.

Основная цель React - минимизировать ошибки, возникающие при разработке пользовательских интерфейсов. Это достигается за счёт использования компонентов - автономных логических фрагментов кода, которые описывают часть пользовательского интерфейса. А уже эти компоненты объединяются для создания полноценного пользовательского интерфейса. React абстрагирует большую часть работы по визуализации, оставляя вам возможность сосредоточиться на дизайне.

Для установки React для начала понадобится установить и настроить локальную среду разработки. Если вы занимаетесь разработкой, то у вас уже явно установлен локальный сервер. Также, можно попробовать React онлайн, прямо в браузере, подключив внешние файлы библиотеки. Для установки React на локальном сервере, потребуется установка Node JS >= 6 версии.

Создание проекта my-app: npm install -g create-react-app

create-react-app my-app

Его запуск: cd my-app

npm start

Чтобы установить React с npm, запустите: npm init

npm install --save react react-dom

Если вы не хотите использовать npm для управления пакетами клиента, react и react-dom пакеты, то можете подключать файлы из CDN.

UMD-сборки React и ReactDOM доступны через CDN.

<script crossorigin src="https://unpkg.com/react@16/umd/react.development.js "></script>

<script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>

Вышеупомянутые версии предназначены только для разработки и не подходят для производства.

**72.React.js. JSX.**

ReactJS - это открытая JavaScript библиотека для создания интерфейсов, которая призвана решать проблемы частичного обновления содержания веб-страницы, с которыми сталкиваются в разработке одностраничных приложений. Рассмотрим это объявление переменной: const element = <h1>Привет, мир!</h1>;

Синтаксис этого странного тега не является ни строкой, ни HTML. Он называется JSX, и это расширение синтаксиса JavaScript. Мы рекомендуем использовать его с React для описания того, как должен выглядеть пользовательский интерфейс. JSX может напомнить вам о языке шаблонов, но со всей полной мощью JavaScript.

JSX создаёт «элементы» React.

React принимает тот факт, что логика отрисовки по сути связана с другой логикой пользовательского интерфейса: как обрабатываются события, как изменяется состояние со временем и как данные подготовлены для отображения.

Вместо искусственного разделения технологий, помещая разметку и логику в отдельные файлы, React [разделяет ответственности] с слабо связанными модулями, называемыми «компонентами», которые содержат вместе разметку и логику.

React не требует использование JSX, но большинство людей находят его полезным в качестве визуальной демонстрации при работе с пользовательским интерфейсом внутри кода JavaScript. Он также позволяет React показывать более полезные сообщения об ошибках и предупреждения.

В приведённом ниже примере мы объявляем переменную с именем name, а затем используем ее внутри JSX, обернув ее в фигурные скобки: const name = 'Josh Perez';

const element = <h1>Привет, {name}</h1>;

В фигурных скобках JSX вы можете поместить любое корректное выражение JavaScript. Например, 2 + 2, user.firstName, или formatName(user) — всё это допустимые выражения в JavaScript.

В приведённом ниже примере мы вставляем результат вызова функции JavaScript, formatName(user), в элемент <h1>.

const element = ( <h1>

Hello, {formatName(user)}!

</h1>);

После компиляции выражения JSX становятся обычными вызовами функций JavaScript и вычисляются в объекты JavaScript. Это означает, что вы можете использовать JSX внутри операторов if и for, присваивать его переменным, принимать его в качесиве аргументов и возвращать из функций:function getGreeting(user) {

if (user) { return <h1>Привет, {formatName(user)}!</h1>; }

return <h1>Привет, незнакомец.</h1>;}

По умолчанию DOM React экранирует любые значения, встроенные в JSX, перед их отрисовкой. Таким образом, гарантируется, что вы никогда не сможете внедрить то, чего явно нет в вашем приложении. Перед отрисовкой все преобразуется в строку.

**73.React.js. Структура проекта. Основные файлы.**

ReactJS - это открытая JavaScript библиотека для создания интерфейсов, которая призвана решать проблемы частичного обновления содержания веб-страницы, с которыми сталкиваются в разработке одностраничных приложений. Команда npx create-react-app my-app создаст нам новое React-приложение.

В результате у нас появится каталог внутри текущей папки и в нём все необходимые файлы. В данном случае каталог my-app. Он будет содержать всё необходимое для разработки этого проекта на React.

В корне проекта расположены следующие файлы и папки: •node\_modules/ •public/ •src/ •.gitignore •package-lock.json •package.json •README.md

node-modules – это папка, которая содержит пакеты (зависимости), которые требуются нашему приложению. Их загружает и обновляет npm.

.gitignore – нужен для скрытия файлов и папок от системы контроля версий GIT.

package.json – это набор метаданных о вашем приложении: название проекта, версия, пакеты от которых зависит проект и т.д. package.json является ключевым файлом для любых приложений, основанных на Node.js.

package-lock.json – файл, который создаёт npm автоматически при установке зависимостей. Он содержит в себе полную информацию обо всех установленных зависимостях, включая их точные версии.

README.md - это краткая справочное руководство по использованию инструмента create-react-app.

В папке public находятся: •favicon.ico, logo192.png, logo512.png – иконки; •index.html – HTML шаблон страницы; •manifest.json – манифест веб-приложения; •robots.txt – текстовый файл, который содержит указания для роботов поисковых систем; Файл index.html как раз является той единственной страницей, которая будет входной для всего нашего приложения. Изменить название этого файла нельзя. В этом файле устанавливаются некоторые базовые настройки для нашего приложения: язык, кодировка, favicon, viewport, заголовок, manifest, title, description и т.д.

manifest.json – это манифест, в котором описывается информация о веб-приложении. Он представляет собой текстовый файл, который в формате JSON содержит название приложения, некоторое количество иконок, стартовый URL, как оно будет выглядеть и некоторые другие параметры.

Папка src (сокращенно от source) – это место, в котором хранятся исходные коды разрабатываемого проекта и где веб-разработчик проводит почти всё время работая над ним.

В этой папке имеется файл index.js. Этому файлу нельзя изменять имя, т.к. он является точкой входа для JavaScript. Начинается файл с импорта модулей JavaScript и других ресурсов. import позволяют index.js использовать код, определенный в другом месте. На первой строчке импортируется библиотека React, а на второй – ReactDOM, которая предоставляет методы для взаимодействия с DOM или другими словами с HTML-страницей. Эти две библиотеки составляют ядро React. Далее импортируются стили, которые применяются ко всему нашему приложению. Затем – React-компонент App из App.js. После этого создаётся корень приложения посредством вызова метода ReactDOM.createRoot(), которому в качестве аргумента передаем DOM-элемент, который мы получили с помощью document.getElementById('root'). Затем выполняется рендеринг React-компонента <App /> в DOM.

React-компонент App, который используется в src/index.js, находится в src/App.js. Этот компонент в данном случае представляет всё наше приложение, которое мы видим в браузере. Этот файл можно разделить на 3 основные части: импорт различных ресурсов, компонент App и инструкция export для экспортирования App.

**70.React.js. Понятие компонента. Способы задания.**

ReactJS - это открытая JavaScript библиотека для создания интерфейсов, которая призвана решать проблемы частичного обновления содержания веб-страницы, с которыми сталкиваются в разработке одностраничных приложений. Компоненты позволяют разделить пользовательский интерфейс на независимые, повторно используемые части и работать с каждой из частей отдельно. Концептуально компоненты похожи на функции JavaScript. Они принимают произвольные входные данные (называемые «props» или свойствами) и возвращают React-элементы, описывающие, что должно появиться на экране.

Самый простой способ определить компонент — написать JavaScript-функцию:

function Welcome(props) { return <h1>Привет, {props.name}</h1>; }

Данная функция — корректный компонент React, потому что он принимает один аргумент-объект «props» (который обозначает свойства) с данными и возвращает элемент React. Такие компоненты мы называем «функциональными», потому что они являются буквально функциями JavaScript.

Вы также можете использовать класс из ES6 для определения компонента:

class Welcome extends React.Component { render() { return <h1>Привет, {this.props.name}</h1>; }}

Два вышеуказанных компонента эквивалентны с точки зрения React. Однако у классов есть дополнительные возможности.

Отрисовка элемента, который является пользовательским компонентом:

const element = <Welcome name="Sara" />;

ReactDOM.render(

element,

document.getElementById('root'));

Когда React видит элемент, представляющий пользовательский компонент, он передаёт JSX-атрибуты этому компоненту в виде единственного объекта. Мы называем этот объект «props».

Компоненты могут ссылаться на другие компоненты в своём выводе. Это позволяет использовать одну и ту же абстракцию компонента для любого уровня детализации.

Например, мы можем создать компонент App, который многократно отрисовывает Welcome:

function App() { return ( <div>

<Welcome name="Сара" /> <Welcome name="Кахаль" />

</div> );}

Свойства объекта props доступны только для чтения.

**71.React.js. Состояние компонента. Назначение, использование.**

ReactJS - это открытая JavaScript библиотека для создания интерфейсов, которая призвана решать проблемы частичного обновления содержания веб-страницы, с которыми сталкиваются в разработке одностраничных приложений. Состояния используются только в классовых компонентах. Для функциональных компонентов используются Хуки.

props (намеренно сокращённо от англ. «properties» — свойства) и state — это обычные JavaScript-объекты. Несмотря на то, что оба содержат информацию, которая влияет на то, что увидим после рендера, есть существенное различие: props передаётся в компонент (служат как параметры функции), в то время как state находится внутри компонента (по аналогии с переменными, которые объявлены внутри функции).

Конструктор — единственное место, где вы можете присвоить что-либо this.state.

Метод setState() планирует изменение объекта состояния (state) компонента. Когда состояние меняется, компонент рендерится повторно.

Состояние может содержать несколько независимых переменных:

constructor(props) { super(props);

this.state = { posts: [], comments: [] }; }

Затем вы можете самостоятельно их обновлять с помощью отдельных вызовов setState().

В React как this.props, так и this.state представляют значения, которые уже были отрендерены, например, то, что видите на экране.

Вызовы setState являются асинхронными, поэтому не стоит рассчитывать, что this.state отобразит новое значение мгновенно после вызова setState. Необходимо добавить функцию, которая сработает только после обновления состояния, если нужно получить новое значение, основанное на текущем состоянии (ниже подробный пример).

Добавление функции даёт вам доступ к текущему состоянию внутри самой функции. Так как setState вызовы «сгруппированы», это помогает связать изменения и гарантирует, что они будут выполняться друг за другом, а не конфликтовать.

Пример кода, который не будет работать так, как ожидаем:

incrementCount() { // Примечание: это \*не\* сработает, как ожидалось.

this.setState({count: this.state.count + 1});}

Корректный код:

incrementCount() { this.setState((state) => { // Важно: используем `state` вместо `this.state` при обновлении.

return {count: state.count + 1} });}

Как говорилось ранее, React намеренно «ждёт» пока все компоненты вызовут setState() в своих обработчиках событий прежде чем начать повторный рендер. Это избавляет от ненужных повторных рендеров.

**75.React.js. Задание стилей.**

ReactJS - это открытая JavaScript библиотека для создания интерфейсов, которая призвана решать проблемы частичного обновления содержания веб-страницы, с которыми сталкиваются в разработке одностраничных приложений. Некоторые примеры в документации используют style для удобства, но использование атрибута style в качестве основного способа для стилизации элементов, как правило, не рекомендуется. В большинстве случаев className следует используется для указания классов, определённых во внешней таблице стилей CSS. style чаще всего используется в приложениях React для добавления динамических стилей, применяемых во время отрисовки. С использованием классов:

Передайте название CSS класса в свойство className:

render() { return <span className="menu navigation-menu">Menu</span> }

Часто CSS классы могут зависеть от props или state компонента:

render() { let className = 'menu'; if (this.props.isActive) { className += ' menu-active'; }

return <span className={className}>Menu</span> }

Инлайн стили:

Атрибут style принимает объект JavaScript со свойствами в стиле написания camelCase, а строку с CSS. Это соответствует свойству JavaScript DOM style, что более эффективно и предотвращает дыры в безопасности при атак типа XSS. Например:const divStyle = { color: 'blue',

backgroundImage: 'url(' + imgUrl + ')',};

function HelloWorldComponent() { return <div style={divStyle}>Привет, мир!</div>;}

Обратите внимание, что к стилям не добавляются браузерные префиксы. Чтобы поддерживать старые браузеры, вам необходимо задать соответствующие свойства для стиля:

const divStyle = { WebkitTransition: 'all', // обратите внимание на прописную букву 'W' здесь

msTransition: 'all' // 'ms' — единственный браузерный префикс в нижнем регистре};

function ComponentWithTransition() { return <div style={divStyle}>Это должно работать во всех браузерах</div>;}

Ключи стилей пишутся в стиле camelCase для совместимости с доступом к свойствам на DOM-узлах с JS-кода (например, node.style.backgroundImage). Префиксы браузеров кроме ms начинаются с прописной буквы. Вот почему у WebkitTransition заглавная буква «W».

React автоматически добавит суффикс «px» к определённым числовым встроенным свойствам стиля. Если вы хотите использовать единицы, отличные от «px», укажите значение в виде строки с нужным единицей измерения. Например:// Результат стиля: '10px'

<div style={{ height: 10 }}> Привет, мир</div>

// Результат стиля: '10%'

<div style={{ height: '10%' }}> Привет, мир</div>

Однако не все свойства стиля преобразуются в строки с уже заданными пикселями. Некоторые из них остаются без каких-либо единиц (например, zoom,order, flex).

**76.React.js. Маршрутизация.**

ReactJS - это открытая JavaScript библиотека для создания интерфейсов, которая призвана решать проблемы частичного обновления содержания веб-страницы, с которыми сталкиваются в разработке одностраничных приложений. Сам по себе React не предоставляет такой возможности (маршрутизации), это задача специальных библиотек. Как правило, используя API такой библиотеки мы подключаем компоненты страниц нашего положения, сопоставляя их с определёнными путями. После этого, переходя с одной страницы на другую мы будем видеть в адресной строке, как изменяется текущий путь.

Рассмотрим библиотеку react-router. Эта библиотека популярна, довольна проста в использовании и обладает хорошей документацией.

В React Router существует 3 категории компонентов: •роутеры (routers), например, <BrowserRouter> или <HashRouter> •маршруты (route matchers), например, <Routes> или <Switch> •и навигация (navigation), например, <Link>, <NavLink> или <Redirect>

Все компоненты, используемые в веб-приложении, должны быть импортированы из react-router-dom.

Любая маршрутизация начинается с роутера. Для веб-проектов react-router-dom предоставляет <BrowserRouter> и <HashRouter>. Основное отличие между ними состоит в способе хранения URL и взаимодействия с сервером. <BrowserRouter> использует обычные URL. <HashRouter> хранить текущую локацию в хэш-части URL (после символа "#").

Существует 2 вида компонентов для поиска совпадений с URL: Switch и Routes. Обратите внимание, что <Route path> ищет совпадение с началом, а не со всем URL. При рендеринге <Switch> определяет первый <Route>, соответствующий текущему URL. Routes просматривает все и ищет наилучшее соответствие.

React Router предоставляет компонент <Link> для создания ссылок в приложении. При использовании <Link> в HTML рендерится <a>. <NavLink> - это специальный тип <Link>, позволяющий определять стили для активного состояния ссылки. Для принудительной навигации используется <Redirect>. При рендеринге <Redirect> выполняется перенаправление.

В приведенном ниже примере у нас имеется 3 страницы, обрабатываемые роутером: главная (home), контакты (about) и страница с пользователями (users). При клике по <Link> (ссылке) роутер рендерит соответствующий <Route> (маршрут, путь).

import { BrowserRouter as Router, Routes, Route, Link,} from 'react-router-dom';

export const App = () => (

<Router> <header>

<Link to="/">Главная</Link> <Link to="/about">Контакты</Link> <Link to="/users">Пользователи</Link> </header>

<main>

<Routes> <Route path="/" element={<Home />} /> <Route path="/about" element={<About />} />

<Route path="/users" element={<Users />} /> </Routes>

</main> </Router>);

**78.React.js. Redux.**

ReactJS - это открытая JavaScript библиотека для создания интерфейсов, которая призвана решать проблемы частичного обновления содержания веб-страницы, с которыми сталкиваются в разработке одностраничных приложений. Redux — библиотека для JavaScript с открытым исходным кодом, предназначенная для управления состоянием приложения. Содержит ряд инструментов, позволяющих значительно упростить передачу данных хранилища через контекст.

Redux удобно использовать, когда: •Требуется синхронизировать состояние между многими компонентами; •Постоянно приходится контролировать изменение состояния во времени — time travel debug; •Приходится перехватывать и вмешиваться в процесс изменения состояния используя middleware.

Ключевые моменты Redux: •Хранилище (store): хранит состояние приложения •Действия (actions): некоторый набор информации, который исходит от приложения к хранилищу и который указывает, что именно нужно сделать. Для передачи этой информации у хранилища вызывается метод dispatch().

•Создатели действий (action creators): функции, которые создают действия

•Reducer : функция (или несколько функций), которая получает действие и в соответствии с этим действием изменяет состояние хранилища

Глобальное хранилище приложения создаётся в отдельном файле, который как правило называется store.js: // Код файла store.js

import { createStore } from 'redux'; const store = createStore(reducer); export default store;

reducer — чистая функция, которая будет отвечать за обновление состояния. Здесь реализовывается логика, в соответствие с которой будет происходить обновление полей store.

Так выглядит базовая функция reducer: function reducer(state, action) {

switch(action.type) {

case ACTION\_1: return { value: action.value\_1 };

case ACTION\_2: return { value: action.value\_2 };

default: return state; }}

Функция принимает значение текущего состояния и обьект события (action). Обьект события содержит два свойства — это тип события (action.type) и значение события (action.value).

К примеру если нужно обработать событие onChange для поля ввода то объект события может выглядеть так:

{ type: "ACTION\_1", value: "Здесь значение поля формы"}

Что бы обновить store необходимо вызвать метод dispatch(). Он вызывается у объекта store который вы создаёте в store.js. Этот объект принято называть store поэтому обновление состояния в моём случае выглядит так: store.dispatch({ type: ACTION\_1, value\_1: "Some text" });

Генераторы действий (actions creators) — это функции, создающие действия.

function addItem(t) { return { type: ADD\_ITEM, title: t }}

**79.Node.js. Общая характеристика, назначение, использование, установка.**

Node.js — это опенсорсная кроссплатформенная среда выполнения для JavaScript, которая работает на серверах. Платформа Node.js построена на базе JavaScript движка V8 от Google, который используется в браузере Google Chrome. Node.js используется при создании сервисов с постоянным обменом информацией с пользователем: социальных сетей, онлайн-игр, чатов, систем совместной работы над проектом, онлайн-редакторов текста и пр.

Node.js лежит в основе интернет вещей. Платформа помогает управлять приборами и создавать серверы, способные одновременно обрабатывать большое количество запросов.

Особенности:•Скорость - одной из основных привлекательных особенностей Node.js является скорость. Причиной является неблокирующая архитектура платформы. •Простота. •JavaScript - в среде Node.js выполняется код, написанный на JavaScript. В браузере и на сервере используются одинаковые концепции языка. •Движок V8 - в основе Node.js, помимо других решений, лежит опенсорсный JavaScript-движок V8 от Google, применяемый в браузере Google Chrome и в других браузерах. Это означает, что Node.js пользуется наработками тысяч инженеров, которые сделали среду выполнения JavaScript Chrome невероятно быстрой и продолжают работать в направлении совершенствования V8. •Асинхронность - в традиционных языках программирования (C, Java, Python, PHP) все инструкции, по умолчанию, являются блокирующими, если только разработчик явным образом не позаботится об асинхронном выполнении кода. JavaScript значительно упрощает написание асинхронного и неблокирующего кода с использованием единственного потока, функций обратного вызова (коллбэков) и подхода к разработке, основанной на событиях Если вы создавали когда-нибудь обработчик события нажатия на кнопку, то вы уже пользовались методиками асинхронного программирования. Асинхронные механизмы позволяют единственному Node.js-серверу одновременно обрабатывать тысячи подключений, не нагружая при этом программиста задачами по управлению потоками и по организации параллельного выполнения кода. Подобные вещи часто являются источниками ошибок.

•Библиотеки - благодаря простоте и удобству работы с менеджером пакетов для Node.js, который называется npm, экосистема Node.js прямо-таки процветает.

Платформа Node.js может быть установлена несколькими способами: •официальные установочные файлы для разных ОС; •менеджер пакетов операционной системы; •nvm. macOS:

Brew install node

Ubuntu (x64)

Для установки последней ветки (0.12) лучше скачать дистрибутив с сайта:

wget http://nodejs.org/dist/v0.12.4/node-v0.12.4-linux-x64.tar.gz

sudo tar -C /usr/local --strip-components 1 -xzf node-v0.12.4-linux-x64.tar.gz

**77.React.js. Хуки.**

ReactJS - это открытая JavaScript библиотека для создания интерфейсов, которая призвана решать проблемы частичного обновления содержания веб-страницы, с которыми сталкиваются в разработке одностраничных приложений. Хуки — это функции, с помощью которых вы можете «подцепиться» к состоянию и методам жизненного цикла React из функциональных компонентов. Хуки не работают внутри классов —дают возможность использовать React без классов.

Хуки налагают два дополнительных правила: •Хуки следует вызывать только на верхнем уровне. Не вызывайте хуки внутри циклов, условий или вложенных функций. •Хуки следует вызывать только из функциональных компонентов React. Не вызывайте хуки из обычных JavaScript-функций. Есть только одно исключение, откуда можно вызывать хуки — это ваши пользовательские хуки.

React содержит несколько встроенных хуков, таких как useState, useEffect . Вы также можете создавать собственные хуки, чтобы повторно использовать их в других своих компонентах.

Хук состояния.

useState — это хук . Мы вызываем его, чтобы наделить наш функциональный компонент внутренним состоянием. React будет хранить это состояние между рендерами. Вызов useState возвращает массив с двумя элементами, который содержит: текущее значение состояния и функцию для его обновления. Эту функцию можно использовать где угодно, например, в обработчике событий. Она схожа с this.setState в классах, но не сливает новое и старое состояние.

Единственный аргумент useState — это начальное состояние. В отличие от this.state, в нашем случае состояние может, но не обязано, быть объектом. Исходное значение аргумента используется только при первом рендере.

const [count, setCount] = useState(0); setCount(count + 1)}>

Хук состояния можно использовать в компоненте более одного раза.

const [age, setAge] = useState(42); const [fruit, setFruit] = useState('банан');

Хук эффекта.

Вам скорее всего доводилось ранее запрашивать данные, делать подписки или вручную менять DOM из React-компонента. Мы расцениваем эти операции как «побочные эффекты» (или сокращённо «эффекты»), так как они могут влиять на работу других компонентов и их нельзя выполнить во время рендера.

С помощью хука эффекта useEffect вы можете выполнять побочные эффекты из функционального компонента. Он выполняет ту же роль, что и componentDidMount, componentDidUpdate и componentWillUnmount в React-классах, объединив их в единый API. Таким образом, хуки дают вам возможность организовать побочные эффекты в компоненте по связанным частям (например, добавление или отмена подписки), вместо того, чтобы принуждать вас делить всё согласно методам жизненного цикла.

К примеру, этот компонент устанавливает заголовок документа после того, как React обновляет DOM: const [count, setCount] = useState(0);

// По принципу componentDidMount и componentDidUpdate: useEffect(() => { // Обновляем заголовок документа, используя API браузера

document.title = `Вы нажали ${count} раз`; });

Когда вы вызываете useEffect, React получает указание запустить вашу функцию с «эффектом» после того, как он отправил изменения в DOM. Поскольку эффекты объявляются внутри компонента, у них есть доступ к его пропсам и состоянию. Так же как и useState, вы можете использовать более одного эффекта в компоненте.

По умолчанию, React запускает эффекты после каждого рендера, включая первый рендер. Вы можете сделать так, чтобы React пропускал повторные подписки если использующийся props, который мы передали в компонент, остался без изменений.

При необходимости вы можете вернуть из эффекта функцию, которая указывает эффекту, как выполнить за собой «сброс». React будет отписываться от компонента перед тем, как компонент размонтируется и перед тем, как перезапустить эффект при повторном рендере.

Создание собственных хуков🡪Иногда нужно повторно использовать одинаковую логику состояния в нескольких компонентах. Традиционно использовались два подхода: компоненты высшего порядка и рендер-пропсы. С помощью пользовательских хуков эта задача решается без добавления ненужных компонентов в ваше дерево.

function useFriendStatus(friendID) { const [isOnline, setIsOnline] = useState(null);

function handleStatusChange(status) { setIsOnline(status.isOnline); }

useEffect(() => { ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);

return () => { ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange); }; });

return isOnline;}

Состояния каждого компонента никаким образом не зависят друг от друга. Хуки — это способ использовать повторно логику состояния, а не само состояние. Более того, каждое обращение к хуку обеспечивает совершенно изолированное состояние. Вы даже можете использовать один и тот же хук несколько раз в одном компоненте. Пользовательские хуки — это в большей степени соглашение, чем дополнение. Если имя функции начинается с ”use” и она вызывает другие хуки, расцениваем как пользовательский хук. Если вы будете придерживаться соглашения useSomething при именовании хуков, это позволит нашему плагину для линтера найти баги в коде, который использует хуки.

**80.Node.js. Структура проекта на node. package.json.**

Node.js — это опенсорсная кроссплатформенная среда выполнения для JavaScript, которая работает на серверах. Предположим, что имеется приложение, предоставляющее API для оформления заказов на зарегистрированных пользователей, со следующей структурой файлов.

•config -- •controllers -- •services -- •routes -- •app.js

В файле app.js содержатся только основные функции промежуточной обработки и функция запуска вашего Node.js приложения.

В директории config находятся файлы для каждой возможной среды окружения, в которой может быть запущено приложение. В такие файлы выносится конфигурация приложения, которая меняется в зависимости от среды запуска.

Директория controllers хранит классы, методы которого выступают обработчиками маршрутов. При этом контроллер создается для каждой взаимосвязанной совокупности маршрутов, например, для всех маршрутов, отвечающих за выполнение действий над заказами.

В services также находятся классы, но их методы отвечают за поставку данных контроллерам, например, с их помощью происходит обращение к базе данных или стороннему API.

В директории routes находятся описания всех маршрутов, причем любая взаимосвязанная совокупность маршрутов выносится в отдельный файл. В routes присутствует файл index.js, который служит для структурированного построения маршрутов. С его помощью можно задать префикс маршрутам отдельной группы.

Package.json

Файл package.json является важнейшим элементов множества проектов, основанных на экосистеме Node.js.

Package.json представляет собой нечто вроде файла-манифеста для проекта. Он даёт в распоряжение разработчика множество разноплановых возможностей. Например, он представляет собой центральный репозиторий настроек для инструментальных средств, используемых в проекте. Кроме того, он является тем местом, куда npm и yarn записывают сведения об именах и версиях установленных пакетов.

Вот пример простейшего файла package.json(может быть пустым):

{

}

Единственное требование к структуре файла заключается в том, что она должна следовать правилам формата JSON.

Вот пример посложнее, который я взял из приложения-примера, написанного с использованием Vue.js:

{ "name": "test-project", "version": "1.0.0", "description": "A Vue.js project", "main": "src/main.js",

"private": true, "scripts": {…}, "dependencies": { "vue": "^2.5.2" }, "devDependencies": {…},

"engines": {…}, "browserslist": […]}

Выделим следующие свойства: •name — задаёт имя приложения (пакета). •version — содержит сведения о текущей версии приложения. •description — краткое описание приложения. •main — задаёт точку входа в приложение. •private — если данное свойство установлено в true, это позволяет предотвратить случайную публикацию пакета в npm. •scripts — задаёт набор Node.js-скриптов, которые можно запускать. •dependencies — содержит список npm-пакетов, от которых зависит приложение. •devDependencies — содержит список npm-пакетов, используемых при разработке проекта, но не при его реальной работе. •engines — задаёт список версий Node.js, на которых работает приложение. •browserlist — используется для хранения списка браузеров (и их версий), которые должно поддерживать приложение.

**81.Node.js. Установка и импорт модулей. Зависимости.**

Node.js — это опенсорсная кроссплатформенная среда выполнения для JavaScript, которая работает на серверах.

Рассмотрим это на примере. В вашем приложении локатора вы будете использовать библиотеку axios, которая поможет вам выполнять запросы HTTP. Установите ее, введя следующее в оболочке: npm install package --save

Вы начинаете эту команду с npm install — это установит данный пакет (для краткости можно использовать npm i). Затем перечисляете пакеты, которые хотите установить, разделяя их пробелом. В заключение вы заканчиваете команду опциональным параметром --save, который указывает, что пакет будет сохранен как зависимость проекта.

Пакеты, используемые для разработки проекта, но не для его создания или запуска, называются зависимостями разработки. Они не требуются для вашего модуля или приложения в производственной среде, но могут оказаться полезными при написании кода.

Установите инструмент статического анализа кода в качестве зависимости разработки для вашего проекта. Попробуйте это в оболочке: npm i пакет@6.0.0 --save-dev

В этой команде вы использовали флаг --save-dev. Это сохранит пакет как зависимость, которая необходима только для разработки. Обратите внимание, что вы добавили @6.0.0 к своему имени зависимости. После обновления модулей на них ставится тег версии. @ указывает npm искать конкретный тег модуля, который вы устанавливаете. Без указанного тега npm устанавливает последнюю версию с тегами. Откройте package.json еще раз:

nano package.json

пакет сохранился как devDependencies, вместе с номером версии, который вы указали ранее

"devDependencies": {

"eslint": "^6.0.0" }

**82.Node.js. Семантическое версионирование.**

Node.js — это опенсорсная кроссплатформенная среда выполнения для JavaScript, которая работает на серверах.

В экосистеме npm принят стандарт версионирования пакетов semver (от слов Semantic Versioning (семантическое версионирование)).

Суть стандарта заключается в том, что версия пакета состоит из трех чисел: основной (major) версии, младшей (minor) версии и patch-версии:

Major.minor.patch

3.12.1

Семантическим этот вид версионирования называется потому, что за каждым числом версии, а точнее, за увеличением числа стоит определенный смысл.

Увеличение patch-версии означает, что в пакет были внесены незначительные исправления или улучшения, которые не добавляют новой функциональности и не нарушают обратную совместимость.

Увеличение minor-версии означает, что в пакет была добавлена новая функциональность, но совместимость сохранилась.

Увеличение же major-версии означает, что в пакет были внесены серьезные изменения API, которые привели к потере обратной совместимости и пользователю пакета, возможно, необходимо внести изменения в свой код, чтобы перейти на новую версию. О таких изменениях и порядке миграции на новую версию обычно можно прочитать в файле CHANGELOG в корне пакета.

Нестабильные версии

Версии пакетов до 1.0.0, например, 0.0.3 или 0.1.2, в системе semver также имеют определенный смысл: такие версии считаются нестабильными и повышение первого ненулевого числа версии должно расцениваться как изменение с потенциальным нарушением обратной совместимости.

**83.Технологии разработки серверных веб-приложений.**

Bеб-приложение — это сайт с элементами интерактива. Они позволяют пользователям взаимодействовать: нажимать кнопки, заполнять формы, запрашивать прайс, совершать покупки. Почтовые клиенты, соцсети, поисковики, интернет-магазины, программы для управления проектами — это всё примеры таких приложений.

С точки зрения архитектуры веб-приложения состоят из двух частей: клиентской и серверной. Клиентская часть также называется фронтэнд. По сути, это то, что пользователи видят на экране устройства. Серверная часть, или бэкенд — программно-аппаратная часть сервиса. Это набор средств, которые реализуют логику приложения.



Серверная часть приложения

Под серверной частью понимают набор аппаратно-программных средств, с помощью которых реализована логика работы приложения. Это то, что происходит вне браузера и компьютера пользователя. К бэкенду относится панель администрирования, управление данными, логика их передачи по запросам фронтенда.

Задача серверной разработки — сделать так, чтобы ответ от сервера доходил до клиента и спроектированные блоки функционировали нужным образом. А также создать для заказчика удобную и безопасную среду для наполнения и обновления контента на сайте.

PHP

PHP — один из самых распространенных языков веб-разработки. PHP в основном применяется, чтобы «оживить» статичные HTML страницы. Почти всегда пользователи приходят на сайт за информацией, которая всё время меняется, и нужно отображать её актуальное состояние. Например, показать прогноз погоды. Если использовать только HTML, то решить такие задачи не получится. Здесь может помочь PHP: он принимает входящий запрос от веб-сервера, выполняет сценарий и возвращает результат в виде готового HTML-кода. Сервер отправляет этот результат в браузер пользователю, который, в свою очередь, отображает её пользователю. После этого пользователь видит актуальный прогноз погоды.

Java

Java — это универсальный язык программирования, который часто используется для веб-разработки, а также для разработки под Android. Среди преимуществ веб- приложений на Java — кроссплатформенность, мультифункциональность, надёжность и гибкость. Для разработки веб-приложений на Java мы используем фреймворк Spring.

Node.js

Node.js — кроссплатформенная среда, которая выполняет код JavaScript вне браузера. Node.js позволяет разработчикам использовать JavaScript, чтобы получить инструменты командной строки. На стороне сервера с его помощью можно запускать сценарии для обработки динамического содержимого веб-страницы, перед тем как она будет доступна в веб-браузере пользователя.

Базы данных в разработке веб-приложений

Также веб-приложению требуется место для хранения данных, и для этого используется база данных. Существует два типа баз данных: реляционные и нереляционные.

Реляционные БД хранят структурированные данные, которые обычно представляют объекты реального мира. Например, это могут быть сведения о человеке или о содержимом корзины для товаров в магазине, сгруппированные в таблицах, формат которых задан на этапе проектирования хранилища.

Нереляционные БД устроены иначе. Например, документо-ориентированные базы хранят информацию в виде иерархических структур данных. Речь может идти об объектах с произвольным набором атрибутов. То, что в реляционной БД будет разбито на несколько взаимосвязанных таблиц, в нереляционной может храниться в виде целостной сущности.

Внутреннее устройство различных систем управления базами данных влияет на особенности работы с ними. Например, нереляционные базы лучше поддаются масштабированию.

Наиболее распространенные базы данных для разработки веб-проектов: MySQL, PostgreSQL, MongoDB.

**84.Серверные фреймворки. Общая характеристика, назначение, примеры, сравнение.**

Серверные веб-фреймворки (или «фреймворки веб-приложений») — это программные среды, которые упрощают создание, поддержку и масштабирование веб-приложений. Они предоставляют инструменты и библиотеки, которые упрощают общие задачи веб-разработки, маршрутизацию URL-адресов для соответствующих обработчиков, взаимодействие с базами данных, поддержку сеансов и авторизацию пользователей, форматирование вывода (HTML, JSON, XML) и улучшение защиты от веб-атак.

Фреймворк🡪Язык программирования🡪Известные варианты использования

Django🡪Python🡪Instagram, Pinterest, Coursera

Laravel🡪PHP🡪Deltanet Travel, Neighborhood Lender, MyRank

Ruby on Rails🡪Ruby🡪ZendDesk, Shopify, GitHub

ExpressJS🡪NodeJS🡪MySpace, GeekList, Storify

Flask🡪Python🡪Red Hat, Rackspace, Reddit

Asp .NET🡪C#🡪Microsoft, Godaddy, Ancestry

Django — это ведущая серверная среда с открытым исходным кодом, основанная на языке программирования Python. Он следует шаблону контроллера представления модели (MVC). Django подходит для разработки сложных и многофункциональных веб-сайтов на основе баз данных.

Преимущества Django: •Многофункциональный •Оптимальная безопасность •Высокая масштабируемость •Универсальный фреймворк

Lavarel — это веб-фреймворк PHP с открытым исходным кодом для разработки веб-приложений на основе Symfony, которые следуют архитектуре модель – представление – контроллер (MVC). Он предлагает модульную упаковочную систему со специальным менеджером зависимостей. Laravel также предоставляет своим пользователям несколько способов доступа к реляционным базам данных, а также утилит для обслуживания и развертывания приложений.

Преимущества Laravel: •Аутентификация •Простой API •Бэкэнды кеширования •Журналы •Тестирование

Ruby on Rails, также известный как Rails, представляет собой платформу серверных веб-приложений на основе Ruby с лицензией MIT. Rails — это среда MVC, предлагающая структуры базы данных по умолчанию, веб-страницы и веб-службы. Ruby on Rails продвигает использование веб-стандартов, таких как XML или JSON, для передачи данных и CSS, JavaScript и HTML для взаимодействия. Rails отдает приоритет использованию шаблонов программной инженерии, таких как шаблон активной записи, соглашение по конфигурации (CoC) и не повторяться (DRY).

Преимущества Ruby on Rails: •Экономия времени •Последовательность •Рентабельность •Качественная разработка •Масштабируемость

Express.js, также известный как Express, представляет собой платформу веб-приложений Node.js и программное обеспечение с открытым исходным кодом, доступное по лицензии MIT. Он используется для создания API-интерфейсов и веб-приложений и считается стандартной серверной средой Node.js. Express — это серверный компонент стека MEAN вместе с интерфейсной структурой AngularJS и базами данных.

Преимущества Express JS: •Простота обучения •Это единый язык программирования — Node.js позволяет разработчикам использовать JavaScript для написания серверных приложений. Он позволяет разработчикам Node.js создавать как внутренние, так и внешние веб-приложения, используя среду выполнения JavaScript. От разработчиков не требуется использовать какой-либо другой серверный язык. Развертывание веб-приложений становится намного более простым, поскольку большинство широко используемых браузеров поддерживают JavaScript.

•Использует функции Full-stack JS •Обеспечивает высокую производительность

Flask — это микро-веб-фреймворк на основе Python, не требующий специальных библиотек и инструментов. Этот бэкэнд не имеет проверки формы, уровня абстракции базы данных или компонентов, которым требуются функции из внешних источников. Flask обеспечивает поддержку расширений, которые могут добавлять функции так, как будто они реализованы во Flask.

Преимущества Flask: •Простота разработки •Он гибкий •Оптимальная производительность •Модульный характер

**85.Express.js. Основные понятия. Установка, запуск, простейшее приложение.**

Express — это простейшая, гибкая и оптимизированная платформа Node.js, упрощающая разработку веб-приложения, которое может обрабатывать различные типы запросов, например GET, PUT, POST и DELETE. Express поставляется с генератором приложений, который автоматически создает файловую архитектуру для приложения. Он предоставляет следующие механизмы: •Написание обработчиков для запросов с различными HTTP-методами в разных URL-адресах (маршрутах). •Интеграцию с механизмами рендеринга «view», для генерации ответов, вставляя данные в шаблоны. •Установка общих параметров веб-приложения, такие как порт для подключения, и расположение шаблонов, которые используются для отображения ответа. •«промежуточное ПО» для дополнительной обработки запроса в любой момент в конвейере обработки запросов. Устанавливаем express глобально: $ sudo npm install -g express

Создаем директорию для наших учебных проектов: $ mkdir -p ~/projects/node-tutorial $ cd ~/projects/node-tutorial

Создаем проект и устанавливаем зависимости: $ express first-app

$ cd first-app && npm install

Теперь приложение можно запустить:$ node app

Проект состоит из:

bin. Содержит исполняемый файл, который запускает приложение. Он запускает сервер (через порт 3000, если нет альтернативы) и настраивает базовую обработку ошибок.

public. Содержит все общедоступные файлы, в том числе файлы JavaScript, каскадные таблицы стилей (CSS), файлы шрифтов, изображения и другие ресурсы, необходимые пользователям при подключении к вашему веб-сайту.

routes. Содержит все обработчики маршрутов для приложения. В этой папке автоматически создаются два файла, index.js и users.js, которые служат примерами отделения конфигурации маршрутов вашего приложения.

views. Содержит файлы, используемые модулем создания шаблонов. На платформе Express настроен поиск подходящего представления при вызове метода преобразования для просмотра. По умолчанию используется модуль создания шаблонов Jade, но он устарел по сравнению с Pug. Поэтому мы используем флаг --view для изменения модуля (шаблона) представления. Просмотреть параметры как флага --view, так и других флагов можно с помощью команды express --help.

app.js. Начальная точка вашего приложения. Этот файл загружает все необходимые компоненты и начинает обслуживать запросы пользователей. Его можно назвать связующим компонентом, который объединяет все части.

package.json. Содержит описание вашего проекта, диспетчер сценариев и манифест приложения. Его основная цель — отслеживать зависимости приложения и их соответствующие версии.

**86.Express.js. Статический сервер.**

Express — это простейшая, гибкая и оптимизированная платформа Node.js, упрощающая разработку веб-приложения, которое может обрабатывать различные типы запросов, например GET, PUT, POST и DELETE.

Шаг 1. Установить движок Node.js в соответствии с инструкциями

Шаг 2. Создать пустой каталог и поместить в нём файл «site.js» следующего содержания: // Перечисление зависимостей:

var path = require('path'); var express = require('express');

// Описание настроек: var staticSiteOptions = { portnum: 80, // слушать порт 80

maxAge: 1000 \* 60 \* 15 // хранить страницы в кэше пятнадцать минут };

// Запуск сайта: express().use(express.static( path.join(\_\_dirname, 'static'), staticSiteOptions)).listen(staticSiteOptions.portnum);

Настройки staticSiteOptions можно переменить по собственному вкусу.

Шаг 3. В том же каталоге подать команду «npm install express», которая автоматически создаст подкаталог «node\_modules» и установит туда Express, скачав из Интернета.

Шаг 4. В том же каталоге создать подкаталог «static» и поместить в нём файлы статического сайта. Этот подкаталог станет корнем сайта. Файлы с именами «index.html» будут использоваться как файлы-индексы. При ответе сервера на запрос с адресом, содержащим только имя некоторого каталога (будь то корень сайта или один из его подкаталогов), будет показан файл с именем «index.html» из указанного каталога (если есть).

Шаг 5. В том же каталоге подать команду «node site», которая запустит сервер. Убедившись в её работоспособности, обеспечить автозапуск этой команды в этом каталоге после каждой перезагрузки операционной системы

**88.Express.js. Middleware.**

Express — это простейшая, гибкая и оптимизированная платформа Node.js, упрощающая разработку веб-приложения, которое может обрабатывать различные типы запросов, например GET, PUT, POST и DELETE.

Функции промежуточной обработки (middleware) - это функции, имеющие доступ к объекту запроса (req), объекту ответа(res) и к следующей функции промежуточной обработки в цикле “запрос-ответ” приложения. Следующая функция промежуточной обработки обозначается переменной next.

Функции промежуточной обработки могут выполнять следующие задачи: •Выполнение любого кода. •Внесение изменений в объекты запросов и ответов. •Завершение цикла “запрос-ответ”. •Вызов следующей функции промежуточной обработки из стека.

Если текущая функция промежуточной обработки не завершает цикл “запрос-ответ”, она должна вызвать next() для передачи управления следующей функции промежуточной обработки. В противном случае запрос зависнет.

Приложение Express может использовать следующие типы промежуточных обработчиков: •Промежуточный обработчик уровня приложения, •Промежуточный обработчик уровня маршрутизатора, •Промежуточный обработчик для обработки ошибок, •Встроенные промежуточные обработчики, •Промежуточные обработчики сторонних поставщиков ПО

Когда вы определяете маршрут в Express, функция-обработчик маршрута, которую вы указываете для этого маршрута, является функцией Middleware: app.get("/user", function routeHandlerMiddleware(request, response, next) {

// execute something});

Middleware для обработки ошибок🡪Разница между middleware для обработки ошибок и обычным middleware заключается в том, что функции middleware для обработки ошибок задают четыре параметра вместо трех, т.е. (error, request, response, next).

Вот пример функции middleware для обработки ошибок:

function errorHandlingMiddlewareFunction(error, request, response, next) {

console.log(error.message);

/\*\* \* Ensure the next error handling middleware is called. \*/

next(error);}

**90.Express.js. Создание RESTfull API.**

Express — это простейшая, гибкая и оптимизированная платформа Node.js, упрощающая разработку веб-приложения, которое может обрабатывать различные типы запросов, например GET, PUT, POST и DELETE.

REST состоит из ряда стандартов или ограничивающих требований по обмену данными между различными системами, и для систем, которые работают согласно принципам REST, используется термин RESTful. REST – абстрактная концепция, это не язык, не фреймворк или тип программного обеспечения.

RESTful API – просто API, который работает согласно принципам и понятиям REST. В случае с API, работающим в сети, сервер получает запрос при помощи конечной точки, где указан URL-адрес, и отправляет обратно ответ, которым часто являются данные в формате вроде JSON.

Независимость от состояния (Statelessness)

Первый принцип – независимость от состояния. Проще говоря, RESTful сервер не должен отслеживать, хранить и тем более использовать в работе текущую контекстную информацию о клиенте. С другой стороны клиент должен взять эту задачу на себя. Другими словами не заставляйте сервер помнить состояние мобильного устройства, использующего API.

Кэшируемая и многоуровневая архитектура

Второй принцип заключается в предоставлении клиенту информации о том, что ответ сервера может быть кэширован на определенный период времени и использоваться повторно без новых запросов к серверу. Этим клиентом может быть как само мобильное устройство, так и промежуточный прокси сервер. Я расскажу подробнее о кэшировании во второй части.

Клиент – серверное разделение и единый интерфейс

RESTful сервер должен прятать от клиента как можно больше деталей своей реализации. Клиенту не следует знать о том, какая СУБД используется на сервере или сколько серверов в данный момент обрабатывают запросы и прочие подобные вещи. Организация правильного разделения функций важна для масштабирования если ваш проект начнет быстро набирать популярность.

Это пожалуй три самых важных принципа, которым нужно следовать в ходе разработки RESTful сервера. Далее будут описаны три менее важных принципа, но все они имеют непосредственное отношение к тому, о чем мы тут говорим.

REST запросы и четыре HTTP метода: •GET •POST •PUT •DELETE

**91.Express.js. Использование роутера.**

Express — это простейшая, гибкая и оптимизированная платформа Node.js, упрощающая разработку веб-приложения, которое может обрабатывать различные типы запросов, например GET, PUT, POST и DELETE.

Маршрутизация определяет, как приложение отвечает на клиентский запрос к конкретному адресу (URI).

Приведенный ниже код служит примером одного из самых простых маршрутов.

// respond with "hello world" when a GET request is made to the homepage

app.get('/', function(req, res) { res.send('hello world');});

Методы Route🡪Метод route является производным от одного из методов HTTP и присоединяется к экземпляру класса express.

Приведенный ниже код служит примером маршрутов, определенных для методов запросов GET и POST к корневому каталогу приложения.

// GET method route

app.get('/', function (req, res) { res.send('GET request to the homepage');});

// POST method route

app.post('/', function (req, res) { res.send('POST request to the homepage');});

Пути маршрутов

Пути маршрутов, в сочетании с методом запроса, определяют конкретные адреса (конечные точки), в которых могут быть созданы запросы. Пути маршрутов могут представлять собой строки, шаблоны строк или регулярные выражения.

app.route()🡪Метод app.route() позволяет создавать обработчики маршрутов, образующие цепочки, для пути маршрута. Поскольку путь указан в одном расположении, удобно создавать модульные маршруты, чтобы минимизировать избыточность и количество опечаток. Дополнительная информация о маршрутах приводится в документации Router(). Пример объединенных в цепочку обработчиков маршрутов, определенных с помощью app.route().

app.route('/book')

.get(function(req, res) { res.send('Get a random book'); })

.post(function(req, res) { res.send('Add a book'); })

.put(function(req, res) { res.send('Update the book'); });

**92.Express.js. Шаблонизатор handlebars.**

Express — это простейшая, гибкая и оптимизированная платформа Node.js, упрощающая разработку веб-приложения, которое может обрабатывать различные типы запросов, например GET, PUT, POST и DELETE.

Что такое Handlebars: В заголовке к части уже и так написано, что это такое, то это движок представления, но если более понятным языком, то это шаблоны, к которым вы можете передавать различные данные и зависимо от них, генерировать страницу.

Зачем нужен Handlebars: Как, наверное, уже многие поняли, он нужен для создания страниц зависим от контента, например, вы сможете легко создать шаблон для страницы пользователя, передав туда информацию нужного вам пользователя, или любую другую подобную вещь.

Самое простое handlebars-выражение представляет собой простой идентификатор: {{some\_contents}}

Работа с Handlebars: Для начала давайте создадим сам рендер, для этого создадим не большой рутинг на который будет отправлять рендер, а вмести с ним и данные для него: JavaScript

const Router = require('express');

const router = new Router()

router.get('/', function(req, res) { res.render("user.hbs", { title: "Страница пользователя", name: "Vova", }); });

module.exports = router

Для отправки шаблонов мы используем метод res.render(), который первым параметром принимает в себя название шаблона, который берётся из папки «views», второй параметр — это объект с данными, которые мы хотим передать.

Теперь создадим сам шаблон, для этого как возможно вы уже поняли, создадим папку «views», если её у вас нет, и создаём файл «user.hbs», вот что в нём пишем:

<!DOCTYPE html> <html> <head> <title>{{title}}</title> <meta charset="utf-8" /></head> <body> <h1>{{title}}</h1> <p>Имя пользователя: {{name}}</p></body> <html>

Для вывода в шаблоне данных из объекта, который передали при рендаре, достаточно просто их заключить в двойные фигурные скобки.

**93.Express.js. Работа с базой данных.**

Express — это простейшая, гибкая и оптимизированная платформа Node.js, упрощающая разработку веб-приложения, которое может обрабатывать различные типы запросов, например GET, PUT, POST и DELETE.

Express-приложение может использовать любые базы данных, поддерживаемые Node (сам по себе Express не определяет каких-либо конкретных дополнительных свойств и требований для управления базами данных). Есть много популярных вариантов -- PostgreSQL, MySQL, Redis, SQLite, и MongoDB.

Существует два подхода при работе с базой данных: •Использование родного языка запросов баз данных (т.е. SQL) •Использование объектной модели данных (ODM) или объектно-реляционной модели (ORM). ODM / ORM представляют данные веб-сайта как объекты JavaScript, которые затем отображаются на поддерживающую базу данных. Некоторые ORM привязаны к определённой базе данных, тогда как другие не зависят от конкретной базы данных.

Наилучшую производительность можно получить с помощью SQL или другого языка запросов, поддерживаемого базой данных. Объектные модели (ODM) часто медленнее, потому что требуют перевода объектов в формат базы данных

Mongoose: -- это средство моделирование объектов базы данных MongoDB, предназначенное для асинхронной работы

// Импортировать модуль mongoose

var mongoose = require('mongoose');

// Установим подключение по умолчанию

var mongoDB = 'mongodb://127.0.0.1/my\_database';

mongoose.connect(mongoDB);

// Позволим Mongoose использовать глобальную библиотеку промисов

mongoose.Promise = global.Promise;

// Получение подключения по умолчанию

var db = mongoose.connection;

// Привязать подключение к событию ошибки (получать сообщения об ошибках подключения)

db.on('error', console.error.bind(console, 'MongoDB connection error:'));

**94.MongoDB. Общая характеристика, сравнение.**

MongoDB — кроссплатформенная документоориентированная база данных с открытым ПО и ведущая база данных NoSQL, написанная на C++. Она опирается на концепции коллекций и документов .

MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.

В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

Коллекция🡪 Коллекция — набор документов MongoDB. Она представляет собой эквивалент таблицы в РСУБД. Коллекция размещается внутри одной базы данных и не требует описания схемы. Документы в коллекции могут иметь разные поля, и, как правило, их объединяет схожее или общее назначение.

Документ🡪 Документ — это набор пар ключ-значение. Он обладает динамической схемой — документы одной коллекции необязательно должны иметь одинаковый набор полей или структуру, а их общие поля могут содержать разные типы данных.

Формат данных в MongoDB🡪 Способ хранения данных в MongoDB в этом плане похож на JSON, хотя формально JSON не используется. Для хранения в MongoDB применяется формат, который называется BSON (БиСон) или сокращение от binary JSON.

BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. BSON в отличие от хранения данных в формате JSON занимает чуть больше места

Кроссплатформенность🡪 MongoDB написана на C++, поэтому ее легко портировать на самые разные платформы. MongoDB может быть развернута на платформах Windows, Linux, MacOS, Solaris. Можно также загрузить исходный код и самому скомпилировать MongoDB, но рекомендуется использовать библиотеки с офсайта.

Простота в использовании🡪 Отсутствие жесткой схемы базы данных и в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему значительно облегчают работу с базами данных MongoDB и дальнейшим их масштабированием. Кроме того, экономится время разработчиков. Им больше не надо думать о пересоздании базы данных и тратить время на построение сложных запросов.

Преимущества MongoDB перед РСУБД: •Отсутствие схемы. MongoDB — это документоориентированная база данных, в которой одна коллекция содержит различные документы. Количество полей, содержимое и размер документов могут отличаться. •Понятная структура объекта. •Отсутствие сложных соединений Join. •Расширенные возможности запросов. MongoDB поддерживает динамические запросы документов, используя язык запросов на их основе, практически не уступающий в эффективности SQL. •Настройка. •Легкость масштабирования. •Не требуется преобразование объектов приложения в объекты базы данных. •Использует внутреннюю память для хранения (оконного) рабочего набора, обеспечивая ускоренный доступ к данным.

**95.Express.js. CRUD операции с MongoDB.**

Express - это фреймворк для создания веб-приложений поверх Node.js. Это упрощает процесс создания сервера, который уже доступен в Node. Если вам интересно, Node позволяет использовать JavaScript в качестве серверного языка.

MongoDB - это база данных. Это место, где вы храните информацию для своих веб-сайтов (или приложений).

CRUD – CREATE, READ, UPDATE, DELETE

•Создать (POST) •Читать (GET) •Обновление (PUT) •Удалить (DELETE)

Создание🡪Есть два метода, которые нам нужны для создания документов в коллекции. insertOne insertMany

> db.collection.insertOne({name: "Pikachu",type: "Electric",health: 35})

> db.pokemons.insertMany([{ name: "Squirtle", type: "Water", health: 44 },{ name: "Poliwag", type: "Water", health: 40 }])

Чтение🡪Для чтения данных из нашей коллекции нам могут понадобиться всего два метода. У этих методов есть несколько операторов и опций, которые могут помочь нам фильтровать наши данные в соответствии с нашими потребностями.find: возвращает несколько документов findOne: возвращает только первый документ

В этих двух операциях мы можем передавать фильтры. Если не указан фильтр, он вернет все документы в случае find и первый документ в случае FineOne.

# найди один

> db. collection.findOne({ name: "George" })

# найти с job == Prog и salary больше 40

> db. collection.find({ job: "Prog", salary: { $gt: 40 }})

$gt – специальный оператор

Обновление🡪Опять же, у нас есть два метода обновления документов. updateOne. updateMany

Оба могут использовать фильтр, как и find, чтобы найти документ, который нам нужно обновить, и объект, который определяет, что мы хотим обновить. Мы используем оператор $set для обновления полей в документе.

# обновление одного

> db.collection.updateOne({ name: "George" },{ $set: { salary: 45 }})

# обновление нескольких

> db. collection.updateMany({ job: "Prog" },{ $set: { salary: 50 }})

Удаление🡪Наша последняя операция — Удалить. deleteOne. deleteMany

Оба похожи на методы обновления и поиска. Они так же могут использовать фильтр, чтобы определить документ, который мы хотим удалить. Давайте посмотрим на несколько примеров:

# удаление одного покемона

> db. collection.deleteOne({ name: "Pikachu" })

# удаление всех покемонов !!

> db. collection.deleteMany({})

$gte означает больше, или равно.

**97.Библиотека Sequelize. Общая характеристика, назначение, установка, подключение.**

Sequelize — это ORM (Object-Relational Mapping — объектно-реляционное отображение или преобразование) для работы с такими СУБД (системами управления (реляционными) базами данных, Relational Database Management System, RDBMS), как Postgres, MySQL, MariaDB, SQLite и MSSQL.

ORM хороши тем, что позволяют взаимодействовать с БД на языке приложения (JavaScript), т.е. без использования специально предназначенных для этого языков (SQL). Установка: yarn add sequelize # или npm i sequelize

Подключение к БД🡪 const { Sequelize } = require('sequelize')

// Вариант 1: передача `URI` для подключения

const sequelize = new Sequelize('sqlite::memory:') // для `sqlite`

const sequelize = new Sequelize('postgres://user:pass@example.com:5432/dbname') // для `postgres`

// Вариант 2: передача параметров по отдельности

const sequelize = new Sequelize({ dialect: 'sqlite', storage: 'path/to/database.sqlite'})

**98.Библ Sequelize. Определение модели данных.**

Sequelize — это инструмент для организации взаимодействия между платформой Node.js и реляционными базами данными без использования специального языка запросов SQL. Sequelize относится к объектно-реляционным сопоставителям (ORM): связывает базы данных (Postgres, MySQL, MariaDB, SQLite и Microsoft SQL Server) с объектами JavaScript, создавая виртуальную объектную базу данных.

Модель — это абстракция, представляющая таблицу в БД. Модель сообщает Sequelize несколько вещей о сущности (entity), которую она представляет: название таблицы, то, какие колонки она содержит (и их типы данных) и др.

У каждой модели есть название. Это название не обязательно должно совпадать с названием соответствующей таблицы. Обычно, модели именуются в единственном числе (например, User), а таблицы — во множественном (например, Users). Sequelize выполняет плюрализацию (перевод значения из единственного числа во множественное) автоматически.

Модели могут определяться двумя способами: •путем вызова sequelize.define(modelName, attributes, options) •путем расширения класса Model и вызова init(attributes, options). После определения, модель доступна через sequelize.model + название модели. sequelize.define

const User = sequelize.define( 'User',

{ // Здесь определяются атрибуты модели

firstName: { type: DataTypes.STRING, allowNull: false, },

lastName: { type: DataTypes.STRING, // allowNull по умолчанию имеет значение true }, },

{ // Здесь определяются другие настройки модели })

// `sequelize.define` возвращает модель

console.log(User === sequelize.models.User) // true

Расширение Model

const { Sequelize, DataTypes, Model } = require('sequelize')

const sequelize = new Sequelize('sqlite::memory:')

class User extends Model {}

User.init( { // Здесь определяются атрибуты модели

firstName: { type: DataTypes.STRING, allowNull: false, },

lastName: { type: DataTypes.STRING, }, },

{ // Здесь определяются другие настройки модели

sequelize, // Экземпляр подключения (обязательно)

modelName: 'User', // Название модели (обязательно) })

console.log(User === sequelize.models.User) // true

**99.Библ Sequelize. Выполнение запросов к бд.**

Sequelize — это инструмент для организации взаимодействия между платформой Node.js и реляционными базами данными без использования SQL. Sequelize относится к объектно-реляционным сопоставителям (ORM): связывает базы данных (Postgres, MySQL, MariaDB, SQLite и Microsoft SQL Server) с объектами JavaScript, создавая виртуальную объектную базу данных.

const firstUser = await User.find() // Получение одного (первого) пользователя

const allUsers = await User.findAll() // SELECT \* FROM ...;

Выборка полей: // Получение полей `foo` и `bar`

Model.findAll({ attributes: ['foo', 'bar'],}) // SELECT foo, bar FROM ...;

// Изменение имени поля `bar` на `baz`

Model.findAll({ attributes: ['foo', ['bar', 'baz'], 'qux'],

}) // SELECT foo, bar AS baz, qux FROM ...;

// Выполнение агрегации

// Синоним `n\_hats` является обязательным

Model.findAll({ attributes: [ 'foo',

[sequelize.fn('COUNT', sequelize.col('hats')), 'n\_hats'], 'bar', ],

}) // SELECT foo, COUNT(hats) AS n\_hats, bar FROM ...;

// instance.n\_hats// Сокращение - не перечислять все атрибуты при агрегации

Model.findAll({ attributes: {

include: [[sequelize.fn('COUNT', sequelize.col('hats')), 'n\_hast']], },})

// Исключение поля из выборки

Model.findAll({ attributes: { exclude: ['baz'], },})

Настройка where позволяет выполнять фильтрацию возвращаемых данных. Существует большое количество операторов, которые могут использоваться совместно с where через Op (см. ниже).

// Выполняем поиск поста по id его автора // предполагается `Op.eq`

Post.findAll({ where: { authorId: 2, }, }) // SELECT \* FROM post WHERE authorId = 2;

// Полный вариант //const { Op } = require('sequelize') /// Post.findAll({ where: { authorId: { [Op.eq]: 2, }, },})

Операторы

[Op.and]: [{ a: 1, b: 2 }], // (a = 1) AND (b = 2)

[Op.or]: [{ a: 1, b: 2 }], // (a = 1) OR (b = 2)

[Op.eq]: 3, // = 3 [Op.ne]: 4, // != 4

[Op.is]: null, // IS NULL [Op.not]: true, // IS NOT TRUE

[Op.or]: [5, 6], // (someAttr = 5) OR (someAttr = 6)

**100.Библ Sequelize. Геттер, сеттер и вирт атрибуты.**

Sequelize — это инструмент для организации взаимодействия между платформой Node.js и реляционными базами данными без использования специального языка запросов SQL. Sequelize относится к объектно-реляционным сопоставителям (ORM): связывает базы данных (Postgres, MySQL, MariaDB, SQLite и Microsoft SQL Server) с объектами JavaScript, создавая виртуальную объектную базу данных.

Sequelize позволяет определять геттеры и сеттеры для атрибутов моделей, а также виртуальные атрибуты — атрибуты, которых не существует в таблице и которые заполняются или наполняются (имеется ввиду популяция) Serquelize автоматически. Последние могут использоваться, например, для упрощения кода.

Геттер — это функция get(), определенная для колонки: const User = sequelize.define('User', { username: { type: DataTypes.STRING,

get() { const rawValue = this.getDataValue(username)

return rawValue ? rawValue.toUpperCase() : null }, },})

Геттер вызывается автоматически при чтении поля.

Обратите внимание: для получения значения поля в геттере мы использовали метод getDataValue(). Если вместо этого указать this.username, то мы попадем в бесконечный цикл.

Сеттер — это функция set(), определенная для колонки. Она принимает значение для установки: const User = sequelize.define('user', { username: DataTypes.STRING, password: { type: DataTypes.STRING, set(value) { this.setDataValue('password', value) }, },})

Сеттер вызывается автоматически при создании экземпляра.

В сеттере можно использовать значения других полей:

const User = sequelize.define('User', { username: DatTypes.STRING,

password: { type: DataTypes.STRING,

set(value) { // Используем значение поля `username`

this.setDataValue('password', this.username + value) }, },})

Представим, что у нас имеется модель User с полями firstName и lastName, и мы хотим получать полное имя пользователя. Для этого мы можем создать виртуальный атрибут со специальным типом DataTypes.VIRTUAL:

const User = sequelize.define('user', { firstName: DataTypes.STRING,

lastName: DataTypes.STRING, fullName: { type: DataTypes.VIRTUAL,

get() { return `${this.firstName} ${this.lastName}` },

set(value) { throw new Error('Нельзя этого делать!') }, },})

В таблице не будет колонки fullName, однако мы сможем получать значение этого поля, как если бы оно существовало на самом деле.

1.Понятие веб-технологий. Чем веб отличается от других сетевых служб?

3.История развития веб-технологий.

2.Основные сетевые протоколы, использующ в вебе.

4.Клиентская часть веб-приложений. Назначение, технологии, схема работы.

5.Серверная часть веб-приложений. Назначение, технологии, схема работы.

8.URL и URI. Понятие, различия, структура, примеры, использование.

96.ORM. Понятие, назначение, использование, примеры.

6.Необходимое программное обеспечения для работы веб-приложений.

7.Необходимое программное обеспечение для веб-разработки.

9.Схема соединения по протоколу HTTP.

12 Методы HTTP. Использование в современных API.

10.HTTP-запросы. Структура, примеры, использование, методы.

11.HTTP-ответы. Структура, примеры, поля ответа, использование.

13.Общая характеристика языка HTML. Назначение, структура. Понятие тега. Виды тегов.

16.HTML. Теги форматирования текста.

14.HTML5. Особенности, примеры тегов, назначение. Понятие семантических элементов.

15.HTML. Теги заголовка веб-страницы.

17.HTML. Встроенные и блочные элементы.

18.HTML. Теги форматирования списков.

19.HTML. Таблицы.

20.HTML. Гиперссылки и якоря.

21.HTML. Универсальные атрибуты тегов.

22.HTML. Изображения, рисунки и мультимедиа.

23.HTML. Формы и поля ввода.

24.HTML. DOM. Дерево элементов. Основные понятия.

25.CSS. Понятие таблицы стилей, назначение, общая характеристика языка.

26.Способы подключения стилей к веб-страницы. Как выбрать правильный?

27.CSS. Селекторы. Виды селекторов.

28.CSS. Задание цвета элементов. Способы задания цветов.

29.CSS. Задание параметров шрифтов.

30.CSS. Задание параметров границ элементов.

31.CSS. Задание размеров блочных элементов.

32.CSS. Задание внутренних и внешних отступов блочных элементов.

34.CSS. Выравнивание и позиционирование блочных элементов.

33.CSS. Единицы измерения размеров. Преимущества и недостатки.

35.CSS. Способы позиционирования блочных элементов.

36.CSS. Псевдоклассы и псевдоэлементы.

37.CSS. Каскадность, наследование, приоритеты стилей.

38.CSS. Адаптивная верстка.

41.CSS. Препроцессоры: основные понятия, назначение, примеры кода на SASS или LESS.

39.CSS. Flexbox. Основные понятия, структура, примеры свойств.

42.Понятие CSS-фреймворков. Примеры, сравнение, назначение.

40.CSS. Grid. Основные понятия, структура, примеры свойств.

43.Bootstrap. назначение, общая характеристика, примеры работы.

44.Общая характеристика языка программирования JavaScript.

45.Клиентский JavaScript. Схема работы, назначение, примеры, способы связывания.

46.JavaScript. Инструкции, комментарии, объявление переменных.

48.JavaScript. Базовые математические и логические операторы.

49.JavaScript. Операции со строками.

47.JavaScript. Простые типы данных.

50.JavaScript. Массивы. Создание, использование, основные операции.

51.JavaScript. Условные операторы.

52. JavaScript. Операторы циклов.

55.JavaScript. Литерация объектов. JSON.

53.JavaScript. Функции.

54.JavaScript. Обработка исключений.

56.JavaScript. Классы в ES6.

60.JavaScript. Стрелочные функции.

57.JavaScript. Функции высших порядков.

58.JavaScript. Наследование.

59.JavaScript. Деструктуризация массивов и объектов.

61.JavaScript. Промисы.

62.DOM. Браузерное окружение в JavaScript.

63.DOM. Навигация. Способы нахождения элементов. Поиск по дереву.

64.DOM. Свойства узлов. Изменение свойств элементов.

65.DOM. Браузерные события. Обработчики событий.

66.DOM. Методы обхода дерева элементов.

87.Express.js. Возврат JSON.

67.DOM. Программное создание нового элемента и добавление его в дерево.

68.Понятие фронтенд фреймворков. Назначение, примеры, использование.

89.REST API. Основные понят, назнач, использован.

74.React.js. Передача параметров как свойств.

69.React.js. Общая характерист. Установка, запуск.

72.React.js. JSX.

73.React.js. Структура проекта. Основные файлы.

70.React.js. Понятие компонента. Способы задания.

71.React.js. Состояние компонента. Назначение, использование.

75.React.js. Задание стилей.

76.React.js. Маршрутизация.

78.React.js. Redux.

79.Node.js. Общая характеристика, назначение, использование, установка.

77.React.js. Хуки.

80.Node.js. Структура проекта на node. package.json.

81.Node.js. Установка и импорт модулей. Зависимости.

82.Node.js. Семантическое версионирование.

83.Технологии разработки серверных веб-приложений.

84.Серверные фреймворки. Общая характеристика, назначение, примеры, сравнение.

85.Express.js. Основные понятия. Установка, запуск, простейшее приложение.

86.Express.js. Статический сервер.

88.Express.js. Middleware.

90.Express.js. Создание RESTfull API.

91.Express.js. Использование роутера.

92.Express.js. Шаблонизатор handlebars.

93.Express.js. Работа с базой данных.

94.MongoDB. Общая характеристика, сравнение.

95.Express.js. CRUD операции с MongoDB.

97.Библиотека Sequelize. Общая характеристика, назначение, установка, подключение.

98.Библ Sequelize. Определение модели данных.

99.Библ Sequelize. Выполнение запросов к бд.

100.Библ Sequelize. Геттер, сеттер и вирт атрибуты.

1.Понятие веб-технологий. Чем веб отличается от других сетевых служб?

3.История развития веб-технологий.

2.Основные сетевые протоколы, использующ в вебе.

4.Клиентская часть веб-приложений. Назначение, технологии, схема работы.

5.Серверная часть веб-приложений. Назначение, технологии, схема работы.

8.URL и URI. Понятие, различия, структура, примеры, использование.

96.ORM. Понятие, назначение, использование, примеры.

6.Необходимое программное обеспечения для работы веб-приложений.

7.Необходимое программное обеспечение для веб-разработки.

9.Схема соединения по протоколу HTTP.

12 Методы HTTP. Использование в современных API.

10.HTTP-запросы. Структура, примеры, использование, методы.

11.HTTP-ответы. Структура, примеры, поля ответа, использование.

13.Общая характеристика языка HTML. Назначение, структура. Понятие тега. Виды тегов.

16.HTML. Теги форматирования текста.

14.HTML5. Особенности, примеры тегов, назначение. Понятие семантических элементов.

15.HTML. Теги заголовка веб-страницы.

17.HTML. Встроенные и блочные элементы.

18.HTML. Теги форматирования списков.

19.HTML. Таблицы.

20.HTML. Гиперссылки и якоря.

21.HTML. Универсальные атрибуты тегов.

22.HTML. Изображения, рисунки и мультимедиа.

23.HTML. Формы и поля ввода.

24.HTML. DOM. Дерево элементов. Основные понятия.

25.CSS. Понятие таблицы стилей, назначение, общая характеристика языка.

26.Способы подключения стилей к веб-страницы. Как выбрать правильный?

27.CSS. Селекторы. Виды селекторов.

28.CSS. Задание цвета элементов. Способы задания цветов.

29.CSS. Задание параметров шрифтов.

30.CSS. Задание параметров границ элементов.

31.CSS. Задание размеров блочных элементов.

32.CSS. Задание внутренних и внешних отступов блочных элементов.

34.CSS. Выравнивание и позиционирование блочных элементов.

33.CSS. Единицы измерения размеров. Преимущества и недостатки.

35.CSS. Способы позиционирования блочных элементов.

36.CSS. Псевдоклассы и псевдоэлементы.

37.CSS. Каскадность, наследование, приоритеты стилей.

38.CSS. Адаптивная верстка.

41.CSS. Препроцессоры: основные понятия, назначение, примеры кода на SASS или LESS.

39.CSS. Flexbox. Основные понятия, структура, примеры свойств.

42.Понятие CSS-фреймворков. Примеры, сравнение, назначение.

40.CSS. Grid. Основные понятия, структура, примеры свойств.

43.Bootstrap. назначение, общая характеристика, примеры работы.

44.Общая характеристика языка программирования JavaScript.

45.Клиентский JavaScript. Схема работы, назначение, примеры, способы связывания.

46.JavaScript. Инструкции, комментарии, объявление переменных.

48.JavaScript. Базовые математические и логические операторы.

49.JavaScript. Операции со строками.

47.JavaScript. Простые типы данных.

50.JavaScript. Массивы. Создание, использование, основные операции.

51.JavaScript. Условные операторы.

52. JavaScript. Операторы циклов.

55.JavaScript. Литерация объектов. JSON.

53.JavaScript. Функции.

54.JavaScript. Обработка исключений.

56.JavaScript. Классы в ES6.

60.JavaScript. Стрелочные функции.

57.JavaScript. Функции высших порядков.

58.JavaScript. Наследование.

59.JavaScript. Деструктуризация массивов и объектов.

61.JavaScript. Промисы.

62.DOM. Браузерное окружение в JavaScript.

63.DOM. Навигация. Способы нахождения элементов. Поиск по дереву.

64.DOM. Свойства узлов. Изменение свойств элементов.

65.DOM. Браузерные события. Обработчики событий.

66.DOM. Методы обхода дерева элементов.

87.Express.js. Возврат JSON.

67.DOM. Программное создание нового элемента и добавление его в дерево.

68.Понятие фронтенд фреймворков. Назначение, примеры, использование.

89.REST API. Основные понят, назнач, использован.

74.React.js. Передача параметров как свойств.

69.React.js. Общая характерист. Установка, запуск.

72.React.js. JSX.

73.React.js. Структура проекта. Основные файлы.

70.React.js. Понятие компонента. Способы задания.

71.React.js. Состояние компонента. Назначение, использование.

75.React.js. Задание стилей.

76.React.js. Маршрутизация.

78.React.js. Redux.

79.Node.js. Общая характеристика, назначение, использование, установка.

77.React.js. Хуки.

80.Node.js. Структура проекта на node. package.json.

81.Node.js. Установка и импорт модулей. Зависимости.

82.Node.js. Семантическое версионирование.

83.Технологии разработки серверных веб-приложений.

84.Серверные фреймворки. Общая характеристика, назначение, примеры, сравнение.

85.Express.js. Основные понятия. Установка, запуск, простейшее приложение.

86.Express.js. Статический сервер.

88.Express.js. Middleware.

90.Express.js. Создание RESTfull API.

91.Express.js. Использование роутера.

92.Express.js. Шаблонизатор handlebars.

93.Express.js. Работа с базой данных.

94.MongoDB. Общая характеристика, сравнение.

95.Express.js. CRUD операции с MongoDB.

97.Библиотека Sequelize. Общая характеристика, назначение, установка, подключение.

98.Библ Sequelize. Определение модели данных.

99.Библ Sequelize. Выполнение запросов к бд.

100.Библ Sequelize. Геттер, сеттер и вирт атрибуты.