Теоретические вопросы к экзамену

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

Под Web-технологиями определяют всю совокупность средств для организации WWW (World Wide Web). Поскольку в каждом сеансе взаимодействуют две стороны - сервер и клиент, Web-технологии разделяются на две группы - технологии стороны сервера (server-side) и технологии стороны клиента (client-side).

К технологиям со стороны клиента относятся все технологии создания Web-страниц (HTML, JavaScript), а к технологиям со стороны сервера относятся технологии доступа к базам данных в сети Интернет (CGI, PHP).

В случае WWW клиентами выступают Web-браузеры, а web-серверы, обрабатывают запросы клиентов и высылают им нужные данные.

Сравнение и отличия

WEB технология - это сервисно-ориентированная технология, являющаяся удаленным вызовом процедур.

HTTP сервисы основаны практически на голом HTTP, и эта технология ресурсно-ориентированная. Нет описания, нет проверки типов, нет проверки входящих и исходящих данных — есть только заголовки, параметры и тело запроса. И исторически используется формат данных JSON.

WEB	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

2. Основные сетевые протоколы, использующиеся в вебе.

Протокол — это набор соглашений, который определяет обмен данными между различными программами. Протоколы задают способы передачи сообщений и обработки ошибок в сети, а также позволяют разрабатывать стандарты, не привязанные к конкретной аппаратной платформе.

Наиболее известные протоколы, используемые в сети Интернет:

- FTP — это протокол передачи файлов со специального файлового сервера на компьютер пользователя. FTP дает возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети.
- POP — это стандартный протокол почтового соединения. Серверы POP обрабатывают входящую почту, а протокол POP предназначен для обработки запросов на получение почты от клиентских почтовых программ.
- Стандарт SMTP задает набор правил для передачи почты. Сервер SMTP возвращает либо подтверждение о приеме, либо сообщение об ошибке, либо запрашивает дополнительную информацию.
- HTTP (HyperText Transfer Protocol, протокол передачи гипертекста) изначально предназначался для передачи данных в виде HTML-документов, а сегодня отвечает за передачу любых данных в клиент-серверном приложении.

3. История развития веб-технологий.

1. Самый первый сайт появился в 1991 году благодаря Тиму Бернерсу-Ли, который представил новую технологию World Wide Web, основанную на протоколе HTTP.
2. 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.
3. Текущая версия HTML 4.01 была стандартизирована 24 декабря 1999 года. Она отличается законченностью и полнотой, а также поддерживает усовершенствованную версию CSS.

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. Позднее к проекту подключились другие авторы, которые стали накладывать свои патчи (программы-дополнения). В 1995 году Брайан Белендорф объединил эти патчи и создал первую версию сервера Apache (сокращение от «a patchy server»), который по настоящее время занимает лидирующую позицию по популярности.

В 1999 году была создана некоммерческая организация Apache Software Foundation, а в марте 2000 года состоялась первая конференция разработчиков под названием ApacheCon, где была представлена версия Apache 2.0. В ней была переработана предыдущая серверная архитектура. На сегодняшний день существуют 2 ветки Apache — 1.3 и 2.0.

Благодаря HTML, CSS, JS, серверным технологиям (Apache, Nginx, AJAX) и языкам программирования (Python, Java, PHP и др.) стало возможным делать крупные массовые web-приложения, обладающие большим функционалом.

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

Клиентская часть приложения — это скрипты, написанные на языке программирования Javascript (JS) и исполняемые в браузере пользователя. Раньше вся клиентская логика основывалась на использовании библиотеки JQuery, которая позволяет работать с DOM, анимацией на странице и делать AJAX запросы. Приложение может состоять только из клиентской части, если не требуется хранить данные пользователя дольше одной сессии.

- DOM (document object model) — это структура HTML-страницы. Работа с DOM — это поиск, добавление, изменение, перемещеие и удаление HTML-тегов.
- AJAX (asynchronous javascript and XML) — это общее название для технологий, которые позволяют делать асинхронные (без перезагрузки страницы) запросы к серверу и обмениваться данными. Так как клиентская и серверная части веб-приложения написаны на разных языках программирования, то для обмена информацией необходимо преобразовывать структуры данных (например, списки и словари), в которых она хранится, в JSON-формат.
- JSON (JavaScript Object Notation) — это универсальный формат для обмена данными между клиентом и сервером. Он представляет собой простую строку, которая может быть использована в любом языке программирования.

С помощью манипуляций с DOM можно полностью управлять содержимым страниц. С помощью AJAX можно обмениваться данными между клиентом и сервером. С этими технологиями уже можно создать SPA. Но при создании сложного приложения код фронтенда, основанного на JQuery, быстро становится запутанным и трудно поддерживаемым.

На смену JQuery пришли Javascript-фреймворки: Angular, React, Vue и другие. У них разная философия и синтаксис, но все они позволяют с гораздо большим удобством управлять данными на фронтенде, имеют шаблонизаторы и инструменты для создания навигации между страницами.

HTML-шаблон — это «умная» HTML-страница, в которой вместо конкретных значений используются переменные и доступны различные операторы: if, цикл for и другие. Процесс получения HTML-страницы из шаблона, когда подставляются переменные и применяются операторы, называется рендерингом шаблона.

Полученная в результате рендеринга страница показывается пользователю. Если необходимо использовать в шаблоне другие данные, то они запрашиваются у сервера.

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

Серверная часть веб-приложения - это программа или скрипт, обрабатывающая запросы пользователя (точнее, запросы браузера). Она работает на удаленном компьютере и может быть написана на разных языках программирования. Когда браузер отправляет запрос к серверу, сервер обрабатывает его и формирует ответ, который отсылается клиенту по сети. Браузер отображает полученный результат в виде очередной веб-страницы.

Веб-браузеры взаимодействуют с веб-серверами при помощи гипертекстового транспортного протокола (HTTP). Запрос включает в себя URL, определяющий затронутый ресурс, метод, определяющий требуемое действие и может включать дополнительную информацию, закодированную в параметрах URL, как POST запрос (данные, отправленные методом HTTP POST) или в куки-файлах.

Веб-серверы ожидают сообщений с клиентскими запросами, обрабатывают их по прибытию и отвечают веб-браузеру при помощи ответного HTTP сообщения (HTTP-ответ). Ответ содержит строку состояния, показывающую, был ли запрос успешным или нет (например, «HTTP/1.1 200 OK» в случае успеха), и тело, если запрашивались данные и они были успешно обработаны сервером. В теле может содержаться сразу веб-страница, описанная языком HTML, или другая структура данных, необходимая клиенту.

Для формирования ответа сервер может совершать запросы к СУБД - структуре хранения данных, которые при запросе пользователя извлекаются и отображаются в веб-приложении.

6. Необходимое программное обеспечения для работы веб-приложений.

Необходимое ПО зависит от харакетра разрабатываемого приложения. Если приложение представляет собой лишь статический документ .html, то для обеспечения работы достаточно разместить файл на веб-сервере Nginx или Apache. Если же приложение предствляет собой не просто статический сайт, а нечто более масштабируемое, то все сводится к следующей конфигурации:

1. Бекенд часть решения
2. СУБД для хранения данных
3. Фронтенд часть решения
4. ОС машины, где работает приложение и т.д.

Бекенд

- Python - Django Rest Framework, Flask, FastAPI
- Java - Spring, Quarkus
- PHP - Symfony
- Node.js - Express.js, Nest.js, Hapi.js

В качестве основного веб-сервера обычно используют Nginx или Apache. Nginx сейчас используют чаще т.к. конфигурировать его проще, удобная система реверс-проксирования и потребляет меньше ресурсов (RAM, CPU) сервера.

СУБД

1. Реляционные - MySQL, MariaDB, Oracle, Postgres - чаще используют в качестве основного хранилища т.к. многие движки поддерживают дополнительную проверку на целостность данных, но особенно плохо масштабируются;
2. Тип “ключ-значение” - Redis, MongoDB - могут выступать как мехнаизм кеширования данных для более оперативного доступа к ним, намного лучше масштабируются для высоконагруженных систем.

Фронтенд

1. Angular
2. React.js
3. Vue.js

Каждый имеет свои особенности, но главное, что они хорошо зарекомендовали себя.

Сервер, контейнеризация, ОС и т.д.

Обычно всегда выбирают сервер на базе какого-либо популярного дистрибутива Linux т.к. они достаточно стабильные, бесплатные, хорошо масштабируемые и POSIX-совместимые. Windows серверы используются в основном для специфических задач по типу хостинга старых .ASP Net приложений или

1С (не уверен насчет 1С). Так же сейчас достаточно популярно использование изолированных контейнеров на базе docker, что повышает стабильность и безопасность основной хост-машины.

7. Необходимое программное обеспечение для веб-разработки.

Для веб-разработки прежде всего необходим браузер для просмотра результата. В зависимости от функционала предполагаемого продукта также потребуются: веб-сервер (Nginx или Apache) и текстовые редакторы для языков программирования, которые планируются быть использованными.

IDE (интегрированная среда разработки) — продвинутые текстовые редакторы с расширенной функциональностью. Они помогают ориентироваться в проекте, запускают автоматические тесты, предоставляют интерфейс для работы с системой контроля версий и автоматически исправляют ошибки. В полноценные IDE можно превратить VS Code и Sublime, но, чтобы этого добиться, придётся подключить к ним множество плагинов, поэтому проще завести самостоятельную программу. IDE бывают платные и бесплатные.

Visual Studio Code:

Visual Studio Code — 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 с помощью технологии IntelliSense дописывает названия объявленных 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.

Atom

Atom — текстовый редактор с множеством настроек, но даже со стандартной конфигурацией помогает работать продуктивно. В Atom встроен менеджер пакетов, при помощи которого можно найти, установить и даже создать собственные пакеты. Предустановлены четыре пользовательских интерфейса и восемь синтаксических тем.

Основные возможности:

- умное автозавершение,
- файловый менеджер, с помощью которого легко просматривать как отдельные файлы, так и целые проекты,
- мультипанельный интерфейс позволяет разделить интерфейс для удобства сравнения и редактирования кода в нескольких файлах,
- функция поиска и замены, предварительный просмотр и замена текста в одном файле или в проекте.

Sublime Text

Его основным преимуществом являются широкие возможности по установке плагинов. Созданием таких утилит занимаются и большие компании, и индивидуальные разработчики, желающие максимально облегчить воркфлоу. Они выполняют разные функции, общая цель которых — оптимизация. Благодаря

поддержке колоссального количества языков и сравнительно небольшого веса, Sublime Text 3 может стать весьма рабочей альтернативой IDE.

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

URI – Uniform Resource Identifier (унифицированный идентификатор ресурса) имя и адрес ресурса в сети, включает в себя URL и URN
URL – Uniform Resource Locator (унифицированный определитель местонахождения ресурса) адрес ресурса в сети, определяет местонахождение и способ обращения к нему
URN - Unifrom Resource Name (унифицированное имя ресурса)

Рассмотрим примеры:

URI – https://wiki.merionet.ru/images/vse-chto-vam-nuzhno-znat-pro-devops/1.png
URL - https://wiki.merionet.ru
URN - images/vse-chto-vam-nuzhno-znat-pro-devops/1.png

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).

URL содержит следующую информацию:

- Протокол, который используется для доступа к ресурсу – http, https, ftp
- Расположение сервера с использованием IP-адреса или имени домена - wiki.merionet.ru / https://192.168.1.17
- 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/
- Необязательный идентификатор фрагмента. Например, https://www.google.com/search?ei=qw3eqwe12e1w&q=URL, где q = URL - это строка запроса, введенная пользователем.

Схемы:

Uniform Resource Locator: <схема>://<логин>:<пароль>@<хост>:<порт>/<путь>?<парам>#<якорь>

Uniform Resource Identifier: /<путь>?<парам>#<якорь>

9. Схема соединения по протоколу HTTP.

Работа по протоколу HTTP происходит следующим образом: программа-клиент устанавливает TCP-соединение с сервером (стандартный номер порта-80) и выдает ему HTTP-запрос. Сервер обрабатывает этот запрос и выдает HTTP-ответ клиенту.

HTTP-запрос состоит из заголовка запроса и тела запроса, разделенных пустой строкой. Тело запроса может отсутствовать.

Заголовок запроса состоит из главной (первой) строки запроса и последующих строк, уточняющих запрос в главной строке. Последующие строки также могут отсутствовать. Запрос в главной строке состоит из трех частей, разделенных пробелами:

1) Метод (иначе говоря, команда HTTP):

- GET - Метод GET служит для получения любой информации, в соответствии URI-запроса.
- HEAD - запрос заголовка документа. Отличается от GET тем, что выдается только заголовок запроса с информацией о документе. Сам документ не выдается.
- POST - этот метод применяется для передачи данных CGI-скриптам. Сами данные следуют в последующих строках запроса в виде параметров.
- PUT - поместить документ на сервере. Запрос с этим методом имеет тело, в котором передается сам документ.
- DELETE - используется для удаления ресурсов, идентифицированных с помощью URI-запроса

2) Ресурс - это путь к определенному файлу на сервере (называется URI), который клиент хочет получить (или разместить - для метода PUT). Если ресурс - просто какой-либо файл для считывания, сервер должен по этому запросу выдать его в теле ответа. Если же это путь к какому-либо CGI-скрипту, то сервер запускает скрипт и возвращает результат его выполнения. Кстати, благодаря такой унификации ресурсов для клиента практически безразлично, что он представляет собой на сервере.

3) Версия протокола - версия протокола HTTP, с которой работает клиентская программа.

Строки после главной строки запроса имеют следующий формат:

Параметр: значение.

Таким образом, задаются параметры запроса. Это является необязательным, все строки после главной строки запроса могут отсутствовать; в этом случае сервер принимает их значение по умолчанию или по результатам предыдущего запроса (при работе в режиме Keep-Alive).

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:

1. Метод HTTP, глагол (например, GET, PUT или POST) или существительное (например, HEAD или OPTIONS), описывающие требуемое действие. Например, GET указывает, что нужно доставить некоторый ресурс, а POST означает отправку данных на сервер (для создания или модификации ресурса, или генерации возвращаемого документа).
2. Цель запроса, обычно URL, или абсолютный путь протокола, порт и домен обычно характеризуются контекстом запроса. Формат цели запроса зависит от используемого HTTP-метода.
3. Версия HTTP, определяющая структуру оставшегося сообщения, указывая, какую версию предполагается использовать для ответа.

HTTP запросы и ответы имеют близкую структуру. Они состоят из:

1. Стартовой строки, описывающей запрос, или статус (успех или сбой). Это всегда одна строка.
2. Произвольного набора HTTP заголовков, определяющих запрос или описывающих тело сообщения.
3. Пустой строки, указывающей, что вся мета информация отправлена.
4. Произвольного тела, содержащего пересылаемые с запросом данные (например, содержимое HTML-формы) или отправляемый в ответ документ. Наличие тела и его размер определяется стартовой строкой и заголовками HTTP.

HTTP-методы: протокол HTTP поддерживает несколько методов. В самой первой версии HTTP/1.0 было три метода: GET, POST и HEAD. В HTTP/1.1 появилось несколько новых методов (см. RFC 2616): OPTIONS, CONNECT, TRACE, PUT и DELETE. В RFC 5789, появившегося в 2010 году, добавился метод PATCH.

- GET используется для простого запроса ресурсов с веб-сервера. Параметры для этого метода передаются через URL.
- POST используется для отсылки данных на веб-сервер через тело HTTP-запроса.
- HEAD схож с методом GET, но выводит только заголовки HTTP-ответа, который возвращает сервер.
- OPTIONS используется для получения списка методов, принимаемых веб-сервером, которые хранятся в заголовке ‘Allow’ в HTTP-ответе.
- PUT предназначен для замены существующего или создания нового ресурса на веб-сервере.
- TRACE используется при тестировании и отсылает полное сообщение, полученное веб-сервером, обратно клиенту, что позволяет увидеть конкретное содержимое, полученное веб-сервером.
- CONNECT используется редко и совместно с прокси-сервером, который может динамически переключаться в режим туннеля.
- PATCH схож с методом PUT. Отличие заключается в этом, что PATCH поддерживает частичную модификацию, в то время как метод PUT поддерживает только полную замену ресурса.

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

Структура HTTP ответа сервера состоит из:

1. Строки состояния HTTP ответа, в которой сервер указывает версию HTTP протокола и код состояния.
2. Нуля или нескольких полей HTTP заголовка, разделенных между собой символом CRLF.
3. Пустой строки (в этой строке должен быть только символ CRLF), эта строка обозначает окончание полей заголовка.
4. Необязательное тело HTTP сообщения.

Стартовая строка ответа HTTP, называемая строкой статуса, содержит следующую информацию:

1. Версию протокола, обычно HTTP/1.1.
2. Код состояния (status code), показывающая, был ли запрос успешным. Примеры: 200, 404 или 302
3. Пояснение (status text). Краткое текстовое описание кода состояния, помогающее пользователю понять сообщение HTTP..

Пример строки статуса: HTTP/1.1 404 Not Found.

Заголовки:

Заголовки ответов HTTP имеют ту же структуру, что и все остальные заголовки: не зависящая от регистра строка, завершаемая двоеточием (':') и значение, структура которого определяется типом заголовка. Весь заголовок, включая значение, представляет собой одну строку.

Поля заголовка объекта HTTP сообщения используются для того, чтобы клиент и сервер могли договориться между собой о том, как они будут обмениваться полезной для пользователя информацией и в каком виде эта информация будет представлена.

Существует множество заголовков ответов. Их можно разделить на несколько групп:

- Основные заголовки (General headers), например, Via (en-US), относящиеся к сообщению в целом.
- Заголовки ответа (Response headers), например, Vary и Accept-Ranges, сообщающие дополнительную информацию о сервере, которая не уместилась в строку состояния.
- Заголовки сущности (Entity headers), например, Content-Length, относящиеся к телу ответа. Отсутствуют, если у запроса нет тела.

12. Методы HTTP. Использование в современных API.

Application Programming Interface (API), или программный интерфейс приложения — это набор инструментов, который позволяет одним программам работать с другими. API предусматривает, что программы могут работать в том числе и на разных компьютерах. В этом случае требуется организовать интерфейс API так, чтобы ПО могло запрашивать функции друг друга через сеть.

Также API должно учитывать, что программы могут быть написаны на различных языках программирования и работать в разных операционных системах.

REST API позволяет использовать для общения между программами протокол HTTP (зашифрованная версия — HTTPS), с помощью которого мы получаем и отправляем большую часть информации в интернете.

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.

Таким образом, мы получаем четыре функции, которые одна программа может использовать при обращении к данным ресурса, в примере — это ресурс для работы со счетами /invoices.

Построение API-системы с использованием ресурсов, HTTP и различных запросов к ним как раз и будет Representational State Transfer (REST API) — передачей состояния представления.

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.

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

HTML (HyperText Markup Language) – это язык, принятый в World Wide Web для создания и публикации Веб-страниц. HTML предоставляет авторам средства для:

- включения в Веб-документы заголовков, текста, таблиц, списков, фотографий и т. п.;
- перехода к другим Веб-страницам посредством щелчка кнопки мыши по гипертекстовой ссылке;
- создания и заполнения форм для транзакций с удаленными службами, например, для поиска информации, бронирования билетов, оформления заказов на товары и т. п.
- непосредственного включения в Веб-документы видеоклипов, звука и других внешних объектов.

HTML — это теговый язык разметки документов. Иными словами, любой документ на языке HTML представляет собой набор элементов, причем начало и конец каждого элемента обозначается специальными пометками, называемыми тегами.

Основы HTML содержат основные правила языка HTML, описание структуры HTML-страницы, отношения в структуре HTML-документа между HTML-элементами.

HTML-документ — это обычный текстовый документ, может быть создан как в обычном текстовом редакторе (Блокнот), так и в специализированном, с подсветкой кода (Notepad++, Visual Studio Code и т.п.). HTML-документ имеет расширение .html.

Тег — это специальное служебное слово, заключенное в угловые скобки. Его ещё называют «элемент HTML». Тегов в языке HTML много и каждый что-то делает с контентом, который обычно находится внутри скобок или между тегами.

Если тег парный, то тегу <ТЕГ> соответствует </ТЕГ>

HTML-теги — основа языка HTML. Каждый HTML-документ состоит из дерева HTML-элементов и текста. Каждый HTML-элемент обозначается начальным (открывающим) и конечным (закрывающим) тегом. Открывающий и закрывающий теги содержат имя элемента.

Все HTML-теги делятся на пять типов:

1. пустые элементы — <area>, <base>,
, <col>, <embed>, <hr>, <img>, <input>, <link>, <menuitem>, <meta>, <param>, <source>, <track>, <wbr>;
2. элементы с неформатированным текстом — <script>, <style>;
3. элементы, выводящие неформатированный текст — <textarea>, <title>;
4. элементы из другого пространства имён — MathML и SVG;
5. обычные элементы — все остальные элементы.

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

Язык для структурирования и представления содержимого всемирной паутины. Это пятая версия HTML. Цель разработки HTML5 - улучшение уровня поддержки мультимедиатехнологий с одновременным сохранением обратной совместимости, удобочитаемости кода для человека и простоты анализа для парсеров.

HTML5 возвращает нас к стилю десятилетней давности, когда практиковалось не закрывать некоторые теги, писать значения без кавычек и по желанию набирать теги в верхнем или нижнем регистре. Отход от жёсткого синтаксиса XHTML позволяет сосредоточиться на содержании сайта, а не на соблюдении пустых формальностей, большинство из которых вызывает раздражение из-за своего несущественного значения и ненужности.

В HTML5 прямого разделения тегов на блочные и строчные фактически нет. Сейчас говорят о категориях контента, который содержится внутри элемента.

Контентная модель HTML5 позволяет более точно описать типы содержимого, с которым сталкиваются авторы при разработке сайтов. Она включает такие типы:

- Поток (flow) – отображает основное содержимое страницы (большая часть тегов включается именно сюда. Например: a, button, header, nav, ol, section);
- Метаданные (metadata) – применяются в заголовке документа и определяют поведение остального содержимого и связь с иными данными (Пример тегов: title, meta, script, style);
- Текст (phrasing) – включает непосредственно текст страницы и используемые для его форматирования теги. Фактически, это строчные элементы: a, button, i, img, span, textarea;
- Встроенный контент (embedded) – часть текстового, строчного содержимого, включающая импортированный контент (Пример тегов: audio, video, img, svg, canvas);
- Интерактивные элементы (interactive) – позволяют посетителю сайта взаимодействовать с ресурсом (Включает теги: button, select, video, a, input, textarea);
- Заголовки (heading) – для определения заголовков сайта, статей, подразделов (теги от h1 до h6);
- Секции (sectioning) – выделение изолированного контента в блоки. Включает теги: article, aside, nav, section.

Пример тегов:

Первый сайт, к слову, создал Тим Бернес-Ли

Текст
 текст - одиночный тег, устанавливает перевод строки в том месте, где этот тег находится.

Семантические элементы

Семантика — это наука о значениях слов и фраз в языке. Таким образом, семантические элементы — это элементы со значением.

Семантические элементы HTML5 доступно описывают свой смысл или назначение как для браузеров, так и для веб-разработчиков.

Семантические элементы четко описывают, что они означают, как браузеру, так и веб-разработчику. В качестве примера не семантических элементов можно привести теги и . 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.

До появления стандарта HTML5 вся разметка страниц осуществлялась преимущественно с помощью элементов , которым присваивали классы class или идентификаторы id для наглядности разметки (например,). С их помощью в HTML-документе размещали верхние и нижние колонтитулы, боковые панели, навигацию и многое другое.

Стандарт HTML5 предоставил новые элементы для структурирования, группировки контента и разметки текстового содержимого. Новые семантические элементы позволили улучшить структуру веб-страницы, добавив смысловое значение заключенному в них содержимому (было , стало <header>). Для отображения внешнего вида элементов не задано никаких правил, поэтому элементы можно стилизовать по своему усмотрению. Для всех элементов доступны глобальные атрибуты.

15. HTML. Теги заголовка веб-страницы.

Под тегом заголовка может пониматься группа тегов **h1-h6** или же тег **title**.

Тег 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. Пример использования:

```
<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>
```

Группа тегов h1-h6

Те <h1>-<h6> в HTML используются для определения заголовков на веб-странице. Всего существует шесть тегов заголовков HTML:

- <h1> - заголовок первого уровня;
- <h2> - заголовок второго уровня;
- <h3> - заголовок третьего уровня;
- <h4> - заголовок четвертого уровня;
- <h5> - заголовок пятого уровня;
- <h6> - заголовок шестого уровня.

HTML заголовок <h1> имеет наибольшее значение, является главным заголовком HTML документа.

Заголовки имеют строгую иерархию. Заголовок HTML <h2> является подзаголовком <h1> и имеет меньшее значение, чем <h1>, но большее, чем <h3> и т.д.

Использовать HTML заголовки необходимо в последовательности от <h1> до <h6>. Использование заголовка меньшего значения, без наличия заголовка большего значения (например использовать <h4>, если не использован <h3>) является ошибкой.

Обычно на странице сайта используется один заголовок <h1>, который может содержать несколько подразделов <h2>, разделенных на подзаголовки <h3> и т.д.

16. HTML. Теги форматирования текста.

**Теги <b> и **

HTML теги <b> и задают полужирное начертание шрифта. Разница между ними заключается в том, что тег <b> является тегом физической разметки, и выделяет текст без акцента на его важность. Тег же определяет текст, которому придают особую важность. Содержимое тега имеет большой вес для поисковиков, а устройства, считывающие с экрана, выделяют его определенной интонацией.

**Теги <i> и **

Теги <i> и задают курсивное начертание шрифта. Тег <i> текст является элементом физической разметки, то есть вложенный текст отличается только визуально и не воспринимается как важный браузерами и поисковыми машинами. Тег экспрессивно-эмоционально выделяет фрагмент текста.

Тег <pre>

Тег <pre> используется для включения в HTML-документ предварительно отформатированного текста. Во вложенном в тег тексте сохраняются все пробелы и разрывы строк (как известно, браузеры по умолчанию любое количество идущих подряд пробелов показывают как один).

Тег <mark>

Тег <mark> определяет выделенный / подсвеченный текст. Визуально содержимое тега выглядит как отмеченное маркером желтого цвета.

Тег <small>

Тег <small> определяет размер шрифта текста на один размер меньше, чем у родительского элемента. В HTML5 тег используется для хранения информации об авторских правах, а также определения мелкого, либо юридического шрифта.

Тег <del> и <s>

Тег <del> выделяет часть текста, которая была удалена из документа.

Тег <ins> и <u>

Тег <ins> используется для определения части текста, которая была добавлена в документ. Содержимое тега в браузере отображается как подчеркнутый текст.

Теги <sub> и <sup>

Тег <sub> используется для определения текста с нижним индексом. Тег выравнивает элемент как подстрочный. Тег <sup> используется для определения текста в верхнем индексе.

Тег <dfn>

Тег <dfn> используется для выделения термина, который упоминается впервые. В браузере содержимое тега выделяется курсивом.

**Теги <p>,
 и <hr>**

Тег <p> определяет абзац в тексте. Друг от друга абзацы отделяются пустой строкой. Браузер автоматически добавляет верхний и нижний отступ, равный 1em, при этом отступы соседних абзацев «схлопываются».

17. HTML. Встроенные и блочные элементы.

Блочные элементы (block)

Блочные элементы в отличие от строчных элементов занимают по умолчанию всю ширину блока-контейнера, в которую они помещены. По этому признаку можно легко определить, к какому способу отображения относится элемент. Высота блочного элемента по умолчанию определяется автоматически и зависит от содержимого, которое в него помещено. Блочный элемент можно представить как прямоугольник, который имеет ширину (width) и высоту (height). Ширину и (или) высоту блочного элемента можно задавать вручную с помощью CSS свойств width и (или) height. Также блочные элементы имеют границы, которые можно оформлять с помощью стилей CSS. Кроме этого блочным элементам можно задавать отступы внешние и внутренние. Внешний отступ (margin) – это отступ от границы до элемента контейнера или до соседних блочных элементов. Внутренний отступ (padding) – это отступ для содержимого блока, который задаётся от границы.

**Элемент **

Элемент часто используется в качестве контейнера для других элементов HTML. Элемент не имеет обязательных атрибутов, но style, class и id являются общими. При использовании вместе с CSS элемент может использоваться для стилей блоков содержимого

**Элемент **

Элемент часто используется в качестве контейнера для некоторого текста. Элемент не имеет обязательных атрибутов, но style, class и id являются общими.

18. HTML. Теги форматирования списков.

Язык гипертекстовой разметки поддерживает три типа списков, для каждого из которых используются свои теги.

Маркированный список

Маркированный список содержит ненумерованные списки элементов без определенной последовательности. Для создания маркированного списка используется блочный элемент <ul>.

Каждый элемент списка начинается с тега открывающего тега <li> и заканчивается закрывающим тегом . Маркером для всех пунктов по умолчанию является маленький черный кружок.

Элементы в маркированных списках по умолчанию отмечены марками (маленькие черные круги).

Нумерованный список

Нумерованный список содержит элементы в определенной последовательности. Список помещается в блочный элемент <ol>.

Каждый элемент нумерованного списка начинается с тега открывающего тега <li> и заканчивается закрывающим тегом . Пункты списка автоматически нумеруются.

Если вы хотите создать нумерованный список с римскими цифрами или же список, где последовательность указана буквами, то просто добавьте к элементу <ol> type="a" или type="I" соответственно.

Список определений (описаний)

В списке определений указываются термины/названия и их определения. Такого рода списки используются для создания словарей, справочников и т.д.

Для создания списка определений используется парный элемент <dl>, в котором термины/названия мы записываем в теге <dt>, а их определения в теге <dd>.

19. HTML. Таблицы.

В HTML для создания таблиц используются теги группы table. К ним относятся:

- <table> - тег обертка таблицы;
- <tr> - тег строки (ряда) таблицы;
- <td> - тег обычной ячейки таблицы;
- <th> - тег ячейки-заголовка таблицы;
- <col> - тег колонки таблицы;
- <colgroup> - тег группы колонок таблицы;
- <thead> - тег верхнего колонтитула таблицы;
- <tbody> - тег основной части таблицы;
- <tfoot> - тег нижнего колонтитула таблицы;
- <caption> - тег подписи таблицы.

Простая HTML таблица

Чтобы создать простую таблицу HTML достаточно 3 тега: <table>, <tr> и <td>.

Тег <table> является корневым контейнером таблицы. Все содержимое таблицы должно находится внутри него.

Далее необходимо определить строки и ячейки - структуру таблицы.

В HTML таблицах строка (ряд) <tr> является контейнером для ячеек. Колонки таблицы определяются позицией ячеек: первая ячейка <td> внутри строки <tr> будет в первой колонке, второй элемент <td> - во второй колонке и так далее.

Для разделения таблицы на колонтитулы (об этом ниже) и основную часть, как обертку строк <tr> основной части таблицы используют тег <tbody>. Его использование не обязательно в простых таблицах, однако некоторые браузеры и HTML редакторы добавляют его автоматически, поэтому в примерах ниже мы также будем его использовать. Если ваша таблица не имеет колонтитулов, вы можете не использовать тег <tbody>.

Заголовки таблицы HTML

В HTML таблицах существует 2 типа ячеек. Тег <td> определяет ячейку обычного типа. Если ячейка выполняет роль заголовка, она определяется с помощью тега <th>.

Объединение ячеек в таблице HTML

В HTML таблицах есть возможность объединить ячейки по горизонтали и вертикали.

Чтобы объединить ячейки по горизонтали используйте атрибут colspan="x", у ячейки <td> или <th>, где x - количество ячеек для объединения.

Чтобы объединить ячейки по вертикали используйте атрибут rowspan="x", у ячейки <td> или <th>, где x - количество ячеек для объединения.

Колонтитулы и подпись в HTML таблицах

HTML таблицы можно поделить на 3 области: верхний колонтитул, основная часть, нижний колонтитул.

Делается это при помощи обвертки строк <tr> выбранной части таблицы тегами. <thead> определяет область верхнего колонтитула, <tfoot> - область нижнего колонтитулы, <tbody> - основную часть таблицы.

По умолчанию, колонтитулы не отличаются стилями (это можно сделать через CSS при необходимости), но могут быть использованы браузерами. Например, при печати многостраничной таблицы колонтитулы могут дублироваться на каждой напечатанной странице.

Правильный порядок размещения тегов областей в коде HTML таблицы <table> следующий: вначале верхний колонтитул <thead>, за ним нижний колонтитул <tfoot>, после них основная часть <tbody>. При этом на странице основная часть будет выведена между колонтитулами.

Колонки и группы колонок

HTML таблицу можно делить на колонки и группы колонок с помощью тегов <col> и <colgroup>.

Такое разделение позволяет задать стили для таблицы используя минимальное количество CSS свойств, тем самым уменьшая объем кода таблицы (вместо определения стилей для каждой ячейки колонки, можно задать стили для одной или нескольких колонок сразу).

Теги <col>и <colgroup> ставятся внутри тега <table> перед тегами <thead>, <tfoot>, <tbody>, <tr> и после тега <caption>.

Оба тега могут определять стили для одной или нескольких колонок. Атрибут span="число", указывает количество колонок, на которые будет влиять тег. Если атрибут span не указан, его значение приравнивается к единице.

20. HTML. Гиперссылки и якоря.

HTML-ссылки создаются с помощью элементов <a>, <area> и <link>. Ссылки представляют собой связь между двумя ресурсами, одним из которых является текущий документ.

Ссылки можно поделить на две категории:

1. ссылки на внешние ресурсы — создаются с помощью элемента <link> и используются для расширения возможностей текущего документа при обработке браузером;
2. гиперссылки — ссылки на другие ресурсы, которые пользователь может посетить или загрузить.

Гиперссылки создаются с помощью элемента <a>. Внутрь помещается текст, который будет отображаться на веб-странице. Текст ссылки отображается в браузере с подчёркиванием, цвет шрифта — синий, при наведении на ссылку курсор мыши меняет вид. Обязательным параметром элемента <a> является атрибут href, который задает URL-адрес веб-страницы.

указатель ссылки

Ссылка состоит из двух частей — **указателя и адресной части. Указатель ссылки** представляет собой фрагмент текста или изображение, видимые для пользователя. **Адресная часть ссылки** пользователю не видна, она представляет собой адрес ресурса, к которому необходимо перейти.

Якоря, или внутренние ссылки, создают переходы на различные разделы текущей веб-страницы, позволяя быстро перемещаться между разделами. Это оказывается очень удобным в случае, когда на странице слишком много текста. Внутренние ссылки также создаются при помощи элемента <a> с разницей в том, что атрибут href содержит имя указателя — так называемый якорь, а не URL-адрес. Перед именем указателя всегда ставится знак #.

Следующая разметка создаст оглавление с быстрыми переходами на соответствующие разделы:

```
<h1>Времена года</h1>
<h2>Оглавление</h2>
<a href="#p1">Лето</a> <!--создаём якорь, указав #id элемента→
<a href="#p2">Осень</a>
<a href="#p3">Зима</a>
<a href="#p4">Весна</a>
<p id="p1">...</p> <!--добавляем соответствующий id элементу→
<p id="p2">...</p>
<p id="p3">...</p>
<p id="p4">...</p>
```

Если нужно сделать ссылку с **одной страницы** сайта на определенный раздел **другой страницы**, то необходимо задать id для этого раздела страницы, а затем добавить его к абсолютному адресу ссылки:

```
<th id="about-color">color</th>
<a href="https://html5book.ru/css-shrifty/#about-color" class="site" target="_blank">color</a>
```

21. HTML. Универсальные атрибуты тегов.

Универсальные атрибуты применяются практически ко всем тегам, поэтому выделены в отдельную группу, чтобы не повторять их для всех тегов.

accesskey - `<a accesskey="c">...</a>`

Позволяет получить доступ к элементу с помощью заданного сочетания клавиш.

class - `class="имя"`

Определяет имя класса, которое позволяет связать тег со стилевым оформлением.

contenteditable - `contenteditable="true | false"`

Сообщает, что элемент доступен для редактирования пользователем.

contextmenu - `contextmenu="идентификатор"`

Устанавливает контекстное меню для элемента.

dir - `dir={ltr | rtl}`

Задает направление и отображение текста — слева направо или справа налево.

hidden - `<E hidden>`

Скрывает содержимое элемента от просмотра.

id - `id="имя"`

Указывает имя стилевого идентификатора.

lang - `lang="код языка"`

Браузер использует значение параметра для правильного отображения некоторых национальных символов.

spellcheck - `spellcheck="true | false"`

Указывает браузеру проверять или нет правописание и грамматику в тексте.

style - `style="правила описания стилей"`

Применяется для определения стиля элемента с помощью правил CSS.

tabindex - `tabindex="число"`

Устанавливает порядок получения фокуса при переходе между элементами с помощью клавиши Tab.

title - `title="текст"`

Описывает содержимое элемента в виде всплывающей подсказки.

xml:lang - `xml:lang="код языка"`

Этот атрибут по своему действию похож на lang, но применяется только в XHTML-документах и указывает язык всего текста или его фрагмента.

22. HTML. Изображения, рисунки и мультимедиа.

HTML Синтаксис изображения:

HTML тег <img> используется для встраивания изображения в веб страницу в графическом формате GIF, JPEG или PNG.

Изображения технически не вставляются на веб страницу; изображения связаны с веб страницами. Тег <img> создает удерживающее пространство для ссылочного изображения.

Тег <img> пуст, он содержит только атрибуты и не имеет закрывающего тега. Тег <img> имеет два обязательных атрибута:

- src - Указывает путь к изображению
- alt - Задает альтернативный текст для изображения

```
<img src="img_chania.jpg" alt="Цветы в Ханье">
```

Атрибут src обязательный, указывает путь (URL) к изображению.Атрибут alt содержит альтернативный текст для изображения, если изображение по какой-то причине не отображается (из-за медленного соединения или ошибки в атрибуте src, тогда изображение используется для чтения с экрана).

Рисунки также могут применяться в качестве карт-изображений, когда картинка содержит активные области, выступающие в качестве ссылок. Такая карта по внешнему виду ничем не отличается от обычного изображения, но при этом оно может быть разбито на невидимые зоны разной формы, где каждая из областей служит ссылкой.

Без задания размеров изображения рисунок отображается на странице в реальном размере. Отредактировать размеры изображения можно с помощью атрибутов width и height. Если будет задан только один из атрибутов, то второй будет вычисляться автоматически для сохранения пропорций рисунка.

Мультимедиа:

В HTML 5 имеются два элемента для работы с мультимедиа:

1. audio.
2. video.

Первый элемент, как следует из названия, служит для добавления аудио-файлов. Простейший пример использования элемента audio в HTML5 приведён ниже:

```
<audio src="http://myrusakov.ru/music.wav" autoplay="autoplay" loop="3">
  <p>Сейчас проигрывается мелодия...</p>
</audio>
```

Как можно видеть, у тега <audio> имеются три атрибута:

1) Атрибут src - указывает путь к файлу для проигрывания.

2) Атрибут autoplay - сообщает браузеру о том, что файл необходимо воспроизвести сразу после загрузки страницы.

3) Атрибут loop - указывает, сколько раз необходимо проиграть файл (разумеется, если пользователь не закроет страницу).

Обратите внимание, что внутри тега <audio> имеется текст, который служит для информации о мелодии, что является полезным, например, для людей с ограниченными возможностями.

Элемент **video** используется для добавления на страницу видео-файлов.

```
<video src="film.avi">Фильм...</video>
```

src - тоже самое, что и в аудио.

Вообще тег audio лучше не использовать на современных сайтах, это отпугивает людей.

23. HTML. Формы и поля ввода.

HTML-формы требуются для сбора данных от посетителей сайта. Например, при регистрации пользователь вводит свое имя, почту и пароль.

Элемент Form

Элемент Form (<form></form>) оборачивает все элементы внутри HTML-формы.

```
<form action="./check.php" method="get" class="form-example">
    .
    .
</form>
```

Атрибуты:

- action – это веб-адрес (URL) программы, которая обрабатывает данные формы.
- method – это HTTP метод, который используется браузером для отправки формы. Возможные значения: POST или GET. POST – отправляет данные формы на сервер. GET – данные отправляются внутри URL, параметры разделяются знаком «?».

Нельзя создать форму внутри формы. То есть использование элемента <form> внутри другого элемента <form> недопустимо.

Элемент Input

Это самый популярный элемент HTML-формы. Используется для создания текстовых полей, в которые пользователь вводит информацию (например: пароль, адрес почты и т. д.).

```
<input type="text" id="name" name="student_name">
```

В примере выше над тегом input были добавлены три атрибута.

type - Указывает на тип ввода. При значении text пользователь должен вводить текстовые данные. У этого атрибута имеется множество значений, например, email, tel (для номера телефона), password и т.д.

id - Это не обязательное, но весьма полезное поле HTML-формы. Например, для определения элементов в CSS/JavaScript. Идентификаторы используются для сопоставления меток с нужными элементами управления формой.

name - Обязательный атрибут. При передаче HTML-формы в серверный код сервер должен интерпретировать данные из формы и правильно их обработать.

Элемент textarea

Иногда одной строки текста бывает недостаточно, а простой элемент input совершенно не подходит. Например, на некоторых сайтах добавляются формы, в которых пользователи оставляют свои отзывы и задают вопросы. В таких случаях лучше прибегнуть к элементу textarea.

```
<textarea id="w3review" name="w3review" rows="4" cols="50">
   Коты работают на заводе
</textarea>
```

Элемент <textarea> не является самозакрывающимся тегом, поэтому требует открывающей и закрывающей скобки.

Атрибуты:

- id – см. <input/>.
- name - см. <input/>.
- cols - задает видимую ширину текстовой области
- rows - задает видимое количество строк в текстовой области

Элемент button

```
<button type="button">Пойти на завод</button>
```

Один из важнейших элементов HTML-формы. Без кнопки вы не сможете отправить и обработать данные формы на сервере. В элементе задается атрибут type, который может принимать три разных значения: submit, reset и button.

Атрибуты:

- type=”reset” - при нажатии очищает все данные формы.
- type=”button” - в нем нет какого-то стандартного поведения. В основном используется в JavaScript для программирования настраиваемого поведения.
- type=”submit” - стандартное поведение кнопки «Отправить», т. е. передача данных на сервер.

Элемент label

```
<form action="/action_page.php">
   <input type="radio" id="html" name="fav_language" value="HTML">
   <label for="html">HTML</label><br>
</form>
```

Пока что пользователь не сможет понять, для чего нужен каждый элемент HTML-формы. Вы не можете предугадать, куда вводить адрес почты, а куда – пароль. Формы смотрятся недоделанными и непонятными. Каждую строку HTML-формы можно отметить элементом label. Наиболее популярным атрибутом label является for.

Атрибуты: for – связывает метку строки с определенным элементом формы. Соответствие проверяется по ID. Значением атрибута ID для элемента input в примере выше является email. Оно совпадает со значением атрибута for для элемента label. Таким образом, оба элемента связаны.

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

DOM — это объектная модель документа, которую браузер создает в памяти компьютера на основании HTML-кода, полученного им от сервера. Иными словами, это представление HTML-документа в виде дерева тегов.

Браузер запрашивает у сервера веб-страницу и получает в ответ ее исходный HTML-код. Браузеру такой код сначала нужно разобрать на элементы. В процессе разбора он строит на основе HTML-кода DOM-дерево. После этого браузер отрисовывает страницу, используя созданное им DOM-дерево, а не исходный HTML-документ.

Такое дерево нужно для правильного отображения сайта и внесения изменений на страницах с помощью JavaScript.

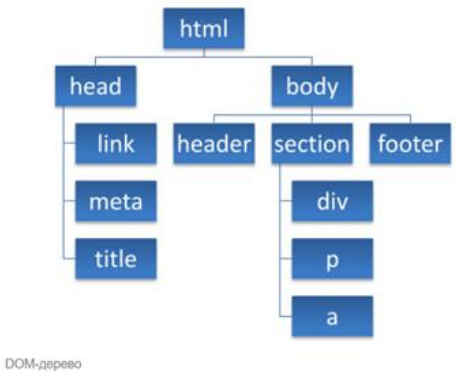
Из чего состоит HTML-код страницы

Страница на HTML состоит из тегов, вложенных в друг друга. Самый общий тег — это HTML. В него вкладываются два дочерних тега head и body.

Тег head используется для подключения информации, которая не будет отображаться непосредственно на странице, но будет использоваться для подключения важных файлов. В body находится значимое содержимое. Обычно в body выделяют три части: шапка сайта, основное содержимое и подвал. В шапке обычно содержится верхнее меню сайта, за это отвечает тег header. Для содержимого нет определенного тега, но обычно используется section. Для подвала используется footer, там обычно содержатся контактная информация, ссылки на ключевые страницы сайта и копирайт. Теги header и footer должны быть единственными на странице, а section может бесконечно повторяться.

Как строится DOM-дерево

Для описания структуры DOM потребуются термины: корневой, родительские и дочерние элементы. Корневой элемент находится в основании всей структуры и не имеет родительского элемента. Дочерние элементы не просто находятся внутри родительских, но и наследуют различные свойства от них. На картинке ниже изображено DOM-дерево.



DOM-дерево

Корневой элемент здесь html — без него страница не будет скомпилирована. Он не имеет родительского (вышестоящего) элемента, но имеет два наследника или дочерних элемента — head и body.

По отношению друг к другу элементы head и body являются сиблингами (братьями и сестрами). В каждый из них можно вложить еще много дочерних элементов. Например, в head обычно находятся link, meta, script или title.

Все эти теги не являются уникальными, и в одном документе может быть по несколько экземпляров каждого из них.

В body могут находиться разнообразные элементы. Например, в родительском body — дочерний элемент header, в элементе header — дочерний элемент section, в родительском section — дочерний div, в div — элемент h3, и наконец, в h3 — элемент span. В этом случае span не имеет дочерних элементов, но их можно добавить в любой момент.

Графическое представление элементов HTML-страницы

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

Что это такое

CSS/Cascading Style Sheets/Каскадные таблицы стилей – это язык таблиц стилей. Его назначение – описание внешнего вида HTML-страницы.

Как подключить

Наиболее традиционный способ подключения css происходит с внешнего файла style.css в секции head с помощью stylesheet. Называется **внешней таблицей.**

```
<head>
  <meta                                                    charset="UTF-8">
  <meta          name="viewport"          content="width=device-width,          initial-scale=1.0">
  <title>Основы                                                    CSS</title>
  <link                          rel="stylesheet"                          href="style.css">
</head>
```

Нетрадиционный способ подключения заключается в использовании тега <style> в секции head по типу:

```
<head>
  <meta                                                    charset="UTF-8">
  <meta          name="viewport"          content="width=device-width,          initial-scale=1.0">
  <title>Основы                                                    CSS</title>
  <style>
    body                                                                    {
        font-size:                                                        16px;
    }
  </style>
</head>
```

Такой способ называется **внутренней таблицей**

Если вы решили сделать так, чтоб все разработчики шарахались от вас, то можете использовать **встроенные стили.** Они пишутся непосредственно внутри HTML-тегов в атрибуте style. Такие стили, действуют только на тот тег, в котором написаны.

.

.

```
<body>
    <p style="font-size: 16px;" Я работаю на заводе и прекрасно себя чувствую в свои 22 года.</p>
</body>
```

Селекторы в CSS

В CSS все завязано по сути селекторах. Они указывают, к каким элементам применять объявления стилей. В качестве селектора могут выступать:

- элементы (имена тегов), например h2
- класс, перед ним ставится точка, например .my-class
- идентификатор, перед ним ставится решетка, например #main
- псевдоклассы и псевдоэлементы

Элементы

```
h2 {
  color: blue;
}
```

Классы

```
<h3 class="big">Текст заголовка</h3>
<p class="big">Текст абзаца</p>
```

Идентификаторы

```
<div id="main">...</div>
```

Псевдоклассы

```
a:hover {
  color: red;
}
```

Псевдоэлементы

```
h2::first-letter {
  color: red;
}
```

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

1) **Связанные стили.** 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:

```
<html>
 <head>
  <link rel="stylesheet" href="mysite.css">
  <link rel="stylesheet" href="http://www.htmlbook.ru/main.css">
 </head>
```

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.

2) **Глобальные стили**. 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, прямо на той же странице с помощью контейнера <style>. В примере задан стиль тега <h1>, который затем можно повсеместно использовать на данной веб-странице.

Пример:

```
<html>
 <head>
  <style>
   H1 {
        font-size: 120%;
        font-family: Verdana, Arial, Helvetica, sans-serif;
        color: #333366;
   }
  </style>
 </head>
</html>
```

3) **Внутренние стили.** 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:

```
<html>
 <body>
  <p style="font-size: 120%; font-family: monospace; color: #cd66cc">Пример текста</p>
</body>
</html>
```

Внутренние стили рекомендуется применять на сайте ограниченно или вообще отказаться от их использования. 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.

4) **Импорт CSS**. В текущую стилевую таблицу можно импортировать содержимое CSS-файла с помощью команды @import. Этот метод допускается использовать совместно со связанными или глобальными стилями.

Пример:

```
@import url("имя файла") типы носителей;
@import "имя файла" типы носителей;
```

После ключевого слова @import указывается путь к стилевому файлу одним из двух приведенных способов — с помощью url или без него.

27. CSS. Селекторы. Виды селекторов.

CSS-селекторы - это специальные выражения описанные с помощью CSS (каскадные таблицы стилей), которые задают правила, как браузеру применять CSS-стили для HTML-элементов, которые определяются внутри блока CSS-стилей.

Различают следующие типы селекторов CSS:

1) универсальные селекторы;

Универсальные селекторы CSS обозначаются обыкновенной звездочкой «*». Под звездочкой может подразумеваться любой элемент разметки HTML-страницы.

```
* {
font-size: 16px;
}
```

2) селекторы по названию тега;

```
h1 {
color: red;
}
```

3) селекторы по классу;

Название класса может быть каким угодно, главное, чтобы оно соответствовало тому элементу, который описывается CSS-правилом. При создании CSS-правила точка «.» перед именем класса обязательна.

```
.main-header {
background-color: #99ffcc;
}
```

4) селекторы по id;

К атрибутам HTML-тегов относится и id. Атрибут этот применяется к индивидуальному элементу, т.е. элемент с id может быть в единственном экземпляре на странице согласно правилам семантики.

```
#heading {
font-size: 35px;
}
```

5) селекторы по атрибуту;

```
a[href = "https://loftblog.ru/"] {
color: black;
}
```

6) селекторы потомков, или контекстные селекторы;

Под CSS селекторами потомков, или контекстными селекторами подразумевается ситуация, когда один элемент HTML-страницы вложен в другой, причем не обязательно он должен быть первым потомком.

```
.wraping p {
padding: 15px;
}
```

7) селекторы дочерние (только первые);

Дочерние селекторы CSS отличаются от селекторов потомков тем, что под дочерним подразумевается только первый потомок, никакие «внуки» и «правнуки» не попадают в эту категорию.

```
.wraping>p {
`padding: 15px;
}
```

8) селекторы сестринские, или соседние (расположенные на одном уровне);

При помощи сестринского селектора создается CSS-правило для HTML-элементов, расположенных на одном уровне, причем тот элемент, который записывается как сестринский должен быть самым ближайшим из себе подобных к тому элементу, к которому он записывается в «сестры».

```
h1 + p {
padding-bottom: 30px;
}
```

9) селекторы псевдоклассов (селекторы состояния);

К селекторам псевдоклассов относятся селекторы CSS, которые описывают состояния элементов при совершении над ними какого-либо действия. Это могут быть наведение на элемент курсора мыши, клик по элементу и т.п. Записываются они через двоеточие к элементу, к которому применяются.

```
a:hover {
text-decoration: none;
}
```

10) селекторы псевдоэлементов.

Их прелесть заключается в том, что с их помощью (::before, ::after) можно добавлять различные элементы на веб-странички, но при этом псевдоэлементы никак не будут отражены в HTML-разметке этих страниц. Записываются они по отношению к элементу через двойное двоеточие.

```
p::first-letter{
color: red;
}
```

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

На уровне элементов HTML, всему можно присвоить цвет. С точки зрения отдельных составляющих элементов, таких как текст, границы и т.д., существует ряд свойств CSS, с помощью которых можно присвоить цвет.

Эти свойства используются для определения цвета текста, его фона и любого оформления текста:

- color (свойство color применяется к тексту и любому оформлению текста)
- background-color (цвет фона текста)
- text-shadow (добавляет и устанавливает параметры тени для текста. Один из параметров тени - это основной цвет, который размывается и смешивается с цветом фона на основе других параметров)
- text-decoration-color (по умолчанию, элементы оформление текста (подчёркивание, перечёркивание) используют цвет свойства color. Но вы можете присвоить другой цвет с помощью свойства text-decoration-color)

Для блочных элементов. Каждый элемент представляет собой прямоугольный блок с каким-то содержимым, фоном и границей.

- border (описание CSS свойство border позволяет одновременно установить ширину, стиль и цвет для границы блока.)
- background-color
- column-rule-color (цвет линий, которые разделяют колонки текста)
- outline-color (цвет контура, вокруг границы элемента. Этот контур отличается от границы элемента тем, что он не занимает место в документе и может перекрывать другой элемент)

Границы

Вокруг любого элемента можно создать границу, т.е. линию вокруг содержимого элемента. Существует краткая запись border, которая позволяет задать сразу все свойства границы, включая даже не связанные с цветом свойства, такие как толщина линии (width), стиль линии (style): сплошная (solid), штриховая (dashed).

- border-color (задаёт единый цвет для всех сторон границы элемента.)
- border-block-start-color и border-block-end-color (с помощью этих свойств вы можете установить цвет границ, которые расположены ближе всего к началу и концу блока)
- border-inline-start-color и border-inline-end-color (эти свойства определяют цвет границы, расположенной ближе всего к началу и концу текста в блоке.)

Как задать цвет:

Для того чтобы задать цвет в CSS, необходимо найти способ как перевести понятие "цвета" в цифровой формат, который может использовать компьютер. Обычно это делают разбивая цвет на компоненты, например какое количество единиц основных цветов содержится в данном цвете или степень яркости. Соответственно, есть несколько способов как можно задать цвет в CSS. Существует набор названий

цветов стандартной палитры, который позволяет использовать ключевые слова вместо числового значения цвета. Ключевые слова включают основные и вторичные цвета (такие как красный (red), синий (blue), или оранжевый (orange)), оттенки серого (от чёрного (black) к белому (white).

Можно также задать с помощью rgb. Шестнадцатеричная запись передаёт цвет, используя шестнадцатеричные числа, которые передают каждый компонент цвета (красный, зелёный и синий). Запись также может включать четвёртый компонент: альфа-канал (или прозрачность). Каждый компонент цвета может быть представлен как число от 0 до 255 (0x00 - 0xFF) или, опционально, как число от 0 до 15 (0x0 - 0xF). Цвет в шестнадцатеричной записи всегда начинается с символа "#". После него начинаются шестнадцатеричные числа цветового кода.

29. CSS. Задание параметров шрифтов.

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*).

В CSS при описании шрифта используют следующие характеристики:

- **Стиль**

  Стиль шрифта определяет тип начертания для представления текста: наклонный, курсивный или обычный. Атрибут **font style** может принимать такие значения как: normal, italic, oblique.

- **Вариант шрифта**

  Свойство **font variant** может иметь одно из двух возможных значений: normal и small-caps (шрифт с малыми прописными буквами).

- **Вес шрифта**

  В CSS жирный шрифт или наоборот, более светлый, задаётся параметром font weight. Некоторыми браузерами поддерживаются числовые значения в описании веса. Каждый шрифт может принимать следующие значения: normal, bold, bolder, lighter. **p {font-family: georgia, garamond serif;}**

- **Размер шрифта**

  Размер шрифта — это расстояние от нижней кромки текста до границ кегельного пространства. Если в HTML данный параметр варьируется от 1 до 7 условных единиц, что не далеко не всегда является удобным, то в CSS с помощью свойства font size размер шрифта можно указать с точностью до пикселя. **p {font-size: 20px;}**

- **Цвет шрифта**

  Возможности CSS позволяют задать цвет текста и его фона с помощью следующих свойств: color(цвет самого текста) и background color(цвет фона). Цветовые значения рекомендуется указывать в общепринятом шестнадцатеричном виде цветовой модели RGB. Первая пара цифр отвечает за уровень красного цвета, вторая — за уровень зелёного цвета, а третья — за уровень синего.

30. CSS. Задание параметров границ элементов.

Под заданием параметров границ элемента понимаются border-подобные свойства. Вот некоторые из них:

Border-width - задает ширину границы;

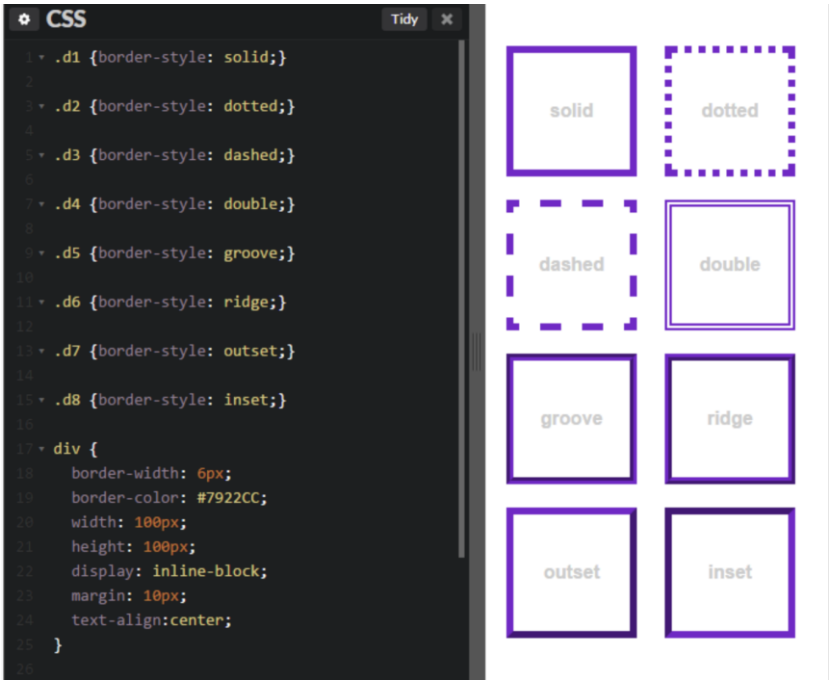
Пример использования:
```
/* все 4 границы имеют ширину 2px: */
border-width: 2px;
/* верхняя и нижняя границы имеют ширину 2px, левая и правая — 4px: */
border-width: 2px 4px;
/* верхняя граница — 2px, левая и правая — 6px, нижняя — 3px: */
border-width: 2px 6px 3px;
/* верхняя граница — 2px, правая — 3px, нижняя — 4px, левая — 5px: */
border-width: 2px 3px 4px 5px;
```

Border-color - задает цвет границы

Пример использования:
```
border-color: #FFFF00;
border-color: transparent
```

Border-style: стиль границы

solid — сплошная граница;
dotted — граница из точек;
dashed — пунктирная граница;
double — двойная граница
и другие (см. фото, если надо)



31. CSS. Задание размеров блочных элементов.

Размеры элементов задаются с помощью свойств **width** (ширина) и **height** (высота). Значение по умолчанию для этих свойств - auto, то есть браузер сам определяет ширину и высоту элемента. Можно также явно задать размеры с помощью единиц измерения (пикселей, em) или с помощью процентов. Процентные значения для свойства width вычисляются на основании ширины элемента-контейнера. Пиксели определяют точные ширину и высоту. Единица измерения em зависит от высоты шрифта в элементе. Если размер шрифта элемента, к примеру, равен 16 пикселей, то 1 em для этого элемента будет равен 16 пикселям.

Если, к примеру, ширина элемента body на веб-странице составляет 1000 пикселей, а вложенный в него элемент имеет ширину 75%, то фактическая ширина этого блока составляет 1000 * 0.75 = 750 пикселей.

Процентные значения для свойства height работают аналогично свойству width, только теперь высота вычисляется по высоте элемента-контейнера.

С помощью дополнительного набора свойств можно установить минимальные и максимальные размеры:

- **min-width:** минимальная ширина
- **max-width:** максимальная ширина
- **min-height:** минимальная высота
- **max-height:** максимальная высота

Например:

min-width: 200px;

width:50%;

max-width: 300px;

Переопределение ширины блока

Свойство box-sizing позволяет переопределить установленные размеры элементов. Оно может принимать одно из следующих значений:

- **content-box**: значение свойства по умолчанию, при котором браузер для определения реальных ширины и высоты элементов добавляет берет соответственно значения свойств width и height элемента
- **padding-box**: указывает веб-браузеру, что ширина и высота элемента должны включать внутренние отступы как часть своего значения. Например, пусть у нас есть следующий стиль

- **border-box**: указывает веб-браузеру, что ширина и высота элемента должны включать внутренние отступы и границы как часть своего значения. Например, пусть у нас есть следующий стиль

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

Внешний отступ (margin)

Внешний отступ — это невидимое пространство вокруг вашего элемента.

```
.box {
  margin-top: -40px;
  margin-right: 30px;
  margin-bottom: 40px;
  margin-left: 4em;
}
<div class="container">
  <div class="box">Change my margin.</div>
</div>
```

Внутренний отступ (padding)

Внутренний отступ расположен между рамкой и областью контента блока.

```
.box {
  padding-top: 0;
  padding-right: 30px;
  padding-bottom: 40px;
  padding-left: 4em;
}
.container {
  padding: 20px;
}
```

```
<div class="container">

  <div class="box">Change my padding.</div>

</div>
```

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

Пиксель px – это самая базовая, абсолютная и окончательная единица измерения.

Количество пикселей задаётся в настройках разрешения экрана, один px – это как раз один такой пиксель на экране. Все значения браузер в итоге пересчитает в пиксели.

Достоинство: Главное достоинство пикселя – чёткость и понятность

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

Относительно шрифта: em

Размеры в em – *относительные*, они определяются по текущему контексту. Размеры, заданные в em, будут уменьшаться или увеличиваться вместе со шрифтом. С учётом того, что размер шрифта обычно определяется в родителе, и может быть изменён ровно в одном месте, это бывает очень удобно. **Достоинства:** 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.

Недостатки:Браузеры округлят не целое значение размера , что не сказывается при больших размерах, но на маленьких деталях можно увидеть неточность. Конфликтует с препроцессорами(То есть,после компиляции не получиться динамически поменять значениеем).

Единица rem: смесь px и em:

Единица rem задаёт размер относительно размера шрифта элемента <html>.

Как правило, браузеры ставят этому элементу некоторый «разумный» (reasonable) размер по умолчанию, который мы, конечно, можем переопределить и использовать rem для задания шрифтов внутри относительно него

Достоинства: единицы измерения rem, которые вычисляются по свойству font-size корневого элемента

Недостатки: Единицы измерения rem поддерживаются только в IE9 и выше.

Относительно экрана: vw, vh, vmin, vmax

Их основное **преимущество** – в том, что любые размеры, которые в них заданы, автоматически масштабируются при изменении размеров окна.

Во всех современных браузерах, **исключая** IE8-, поддерживаются новые единицы из черновика стандарта CSS Values and Units 3

34. CSS. Выравнивание и позиционирование блочных элементов.

В последних реализациях CSS вы можете также использовать возможности Уровня 3 (CSS3), позволяющие выровнять по центру абсолютно позиционированные элементы:

- Выравнивание по вертикали в уровне 3
- Выравнивание по вертикали и горизонтали в уровне 3
- Выравнивание в области просмотра в уровне 3

ЦЕНТРИРОВАНИЕ ПО ВЕРТИКАЛИ В CSS УРОВНЯ

Для документа, который выглядит вот так:

```
<div class=container3>
  <p>Этот абзац…
</div>
```

Основные правила:

1. Сделайте контейнер *относительно позиционированным* (position: relative), что превратит его в контейнер для абсолютно позиционированных элементов.
2. Сам элемент сделайте *абсолютно позиционированным* (position: absolute).

ВЕРТИКАЛЬНОЕ И ГОРИЗОНТАЛЬНОЕ ЦЕНТРИРОВАНИЕ В CSS УРОВНЯ 3

Побочный эффект абсолютного позиционирования абзаца состоит в том, что абзац становится ровно таким широким, каким он должен быть (если только мы явно не укажем ширину). В примере ниже мы именно этого и хотим: мы размещаем по центру абзац из одного слова (“Центр!“), так что ширина абзаца должна быть равна ширине этого слова.

```
<div class=container4>
  <p>Центр!
</div>
```

ЦЕНТРИРОВАНИЕ В ОБЛАСТИ ПРОСМОТРА В CSS УРОВНЯ 3

Контейнером по умолчанию для абсолютно позиционированных элементов является область просмотра. (В случае с браузером это окно браузера). Таким образом, центрирование элемента в области просмотра не вызывает сложности. Далее приведен полный пример. (В этом примере использован синтаксис HTML5.)

```
<html>
 <style>
  body {
     background: white }
  section {
     background: black;
     color: white;
     border-radius: 1em;
     padding: 1em;
     position: absolute;
     top: 50%;
     left: 50%;
     margin-right: -50%;
     transform: translate(-50%, -50%) }
 </style>
 <section>
  <h1>Красиво выровнен по центру</h1>
  <p>Этот текстовый блок выровнен вертикально по центру.
  <p>И горизонтально, если окно достаточно широкое.
 </section>
```

Правило 'margin-right: -50%' необходимо для компенсации 'left: 50%'. Правило 'left' уменьшает доступную для элемента ширину на 50%. Поэтому визуализатор постарается сделать линии длиною не более половины ширины контейнера.

35. CSS. Способы позиционирования блочных элементов.

Базовый поток документа

HTML-документ состоит из большого количества элементов, вложенных друг в друга. По умолчанию размещение всех элементов на странице осуществляется в нормальном или базовом потоке. Элементы выводятся в том порядке, в котором они следуют в HTML коде. При этом слой элемента тем выше, чем данный элемент является более вложенным в коде. Положение элемента в потоке также зависит от значения свойства display (block - в столбец, inline - построчно, с переносом на новую строку, если минимально возможная ширина всех элементов для отображения контента меньше ширины страницы).

Кроме block, inline есть и другие варианты отображения элементов, но все они располагаются в базовом потоке документа. В CSS есть свойства, с помощью которых элементы можно «вырвать» из основного потока документа и задать им другое положение вне базового потока элементов. К этим свойствам относятся position и float.

Каждый элемент в потоке занимает определенную область. Но область элемента не всегда сохраняется за ним при его позиционировании. Например, при задании элементу position: absolute или position: fixed. место не сохраняется за элементом. Другие элементы его «не видят» и располагаются, игнорируя его присутствие в коде.

Поплавки (float).

В этом случае, контейнер изначально позиционируется в соответствии с потоком, а затем сдвигается вправо или влево насколько возможно.

CSS-свойство position

Position — это одно из свойств с помощью которого можно изменить базовое поведение элементов в потоке и разместить его в другом месте относительно окна браузера или других элементов на веб-странице. Свойство position имеет 5 значений:

- static (статичное позиционирование) - значение по умолчанию, элемент является не позиционированным, т.е. отображается как обычно (в потоке). 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;
- relative (относительное) - 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;
- absolute (абсолютное) - позволяет разместить элемент именно там, где вы хотите, с помощью CSS-свойств top, bottom, left и right. Позиционирование выполняется относительно ближайшего позиционированного предка. Под позиционированным элементом понимается элемент с position: relative, absolute, fixed или sticky;
- fixed (фиксированное) - 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» и др.;

- sticky (липкое) - элемент будет становиться плавающим, как только область видимости достигнет определенной позиции, например top: 0px. То есть элемент зависнет на этой позиции, но только в рамках контейнера-родителя.

Относительное позиционирование очень часто используется вместе с абсолютным позиционированием.

36. CSS. Псевдоклассы и псевдоэлементы.

Стили CSS, которые мы привыкли использовать, применяются для элементов, которые можно обнаружить в структуре документа и с ними можно взаимодействовать. Но есть ряд элементов и состояний элементов, которые не отображаются в дереве документа, но к ним точно также необходимо применять определенные стили CSS. Например, не существует какого-то конкретного элемента в исходном коде для первой буквы в слове, для области, которая располагается перед определенным элементом, и т.д. Конкретными тэгами мы не можем описать состояние элемента, например, ссылку на которую наведен курсор мыши. Для решения этой проблемы используются специальные селекторы, которые называются псевдоэлементы и псевдоклассы.

Псевдоклассы – это селекторы, которые определяют **состояние** уже существующих элементов, которое может меняться при определенных условиях.

Псевдоэлементы – это селекторы, которые определяют **область элементов**, которая изначально отсутствует в дереве документа. Эта область создается искусственно с помощью CSS.

Ключевое отличие между ними в том, что псевдоклассы определяют именно состояние элементов, которые уже существуют на странице, а псевдоэлементы создают области, которых изначально на веб-странице не было. Но и те и другие отсутствуют в исходном коде документа.

Список псевдоклассов (элемент E):

- E:root
- E:nth-child(n)
- E:nth-last-child(n)
- E:nth-of-type(n)
- E:nth-last-of-type(n)
- E:first-child
- E:last-child
- E:first-of-type
- E:last-of-type
- E:only-child
- E:only-of-type
- E:empty
- E:link
- E:visited
- E:active
- E:hover
- E:focus
- E:target
- E:lang(fr)
- E:enabled
- E:disabled
- E:checked

- E:not(s)

Псевдоэлементы пишутся с двойным двоеточием (элемент E):

- E::first-line
- E::first-letter
- E::before
- E::after

37. CSS. Каскадность, наследование, приоритеты стилей.

Каскадность (Cascade)

Каскадность подразумевает возможность применения стилей из разных источников к одному и тому же объекту. Браузеру необходимо разобраться с тем, каким образом отображать элемент. Для этого выстраивается очередь приоритетов в следующем порядке (от самого низкого до максимально значимого):

1. Таблицы стилей браузера (у многих просмотрщиков для разных элементов определены стили по умолчанию – шрифты, отступы, границы, размеры);
2. Таблицы стилей пользователя (посетитель сайта может заранее настроить отображение тегов у себя в браузере. Если они не переопределены разработчиками, то будут использованы);
3. Таблицы стилей разработчика (автор сайта задает настройки элементам, которые по приоритету выше, чем два предыдущих типа. Это рассчитано на максимальную схожесть внешнего вида ресурса у пользователей);
4. Стили браузера !important (значение !important у любого свойства имеет повышенный приоритет);
5. Стили пользователя !important (пользователь также может принудительно вызвать определенный стиль, установив значение !important);
6. Стили разработчика !important (максимально высокий приоритет).

Специфичность (Specificity)

Отвечает за приоритет применения стилей к элементам в зависимости от типа селектора.

Специфичность подразумевает задание весов конкретным правилам. Более высокая его величина определяет итоговое значение отображаемого свойства.

Цепочка приоритетов выглядит так (в порядке увеличения значимости):

1. Селекторы элементов и псевдоэлементов (выбирается конкретный тег или его часть);
2. Селекторы классов, атрибутов и псевдоклассов (выбор на основании класса, специфичного атрибута или состояния класса);
3. Селекторы идентификаторов (так как задаются в единственном экземпляре на странице, то имеют максимальный вес).

Наследование (Inheritance)

Еще один способ определения конечных свойств элемента – на основании наследования. Как известно, DOM-дерево представлено родителями и потомками, поэтому при вычислении свойств дочерних элементов используются параметры предков, если они не указаны для потомков.

Не все CSS свойства по умолчанию наследуются, о чем можно узнать из документации. К таковым, например, относят границы у элементов, их цвет и толщина, отступы. Тем не менее, если необходимо, можно принудительно задать наследование через значение inherit.

38. CSS. Адаптивная верстка.

Адаптивная верстка (Adaptive Layout) позволяет подстраиваться основному контейнеру и любому другому элементу сайта под разрешение экрана, делая возможным менять размер шрифта, расположение объектов, цвет и т. д. Происходит это динамически, например, с использованием медиа-запросов (@media), позволяющих автоматически определять разрешение монитора, тип устройства и подставлять указанные значения в автоматическом режиме. В примере ниже задается ширина div равная 960px для всех устройств, ширина которых меньше 1200px и 320px для всех устройств, ширина которых меньше 480px.

```
@media all and (max-width:1200px){
  div{
    width: 960px;
  }

@media all and (max-width:480px){
  div{
    width: 320px;
  }
```

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

CSS Flexbox — это технология для создания сложных гибких макетов за счёт правильного размещения элементов на странице. Чтобы начать работать с CSS Flexbox, нужно сделать контейнер flex-контейнером. Делается это так:

```
#container {
 display: flex;
}
```

Так у блоков появилось свойство flex-контекст, которое в дальнейшем позволит ими управлять гораздо проще, чем с использованием стандартного CSS. У flex-контейнера есть две оси: главная и перпендикулярная ей. По умолчанию все предметы располагаются вдоль главной оси — слева направо. А вот flex-direction позволяет вращать главную ось:

- row - ось горизонтальна;
- column - ось вертикальна;
- row-reverse - элементы разворачиваются по горизонтальной оси;
- column-reverse - элементы разворачиваются по вертикальной оси;

justify-content отвечает за выравнивание элементов по главной оси:

- flex-start - выравнивание по левому краю;
- flex-end - выравнивание по правому краю;
- center - выравнивание по центру;
- space-between - выравнивание по ширине;
- space-around - расстояние слева и справа каждого элемента равны.

align-items работает с осью, перпендикулярной главной оси:

- flex-start - выравнивание по верхнему краю;
- flex-end - выравнивание по нижнему краю;
- center - выравнивание по центру родительского элемента;
- stretch - растягивание по родительскому элементу (высота блоков должна быть равна auto);
- baseline - выравнивание по верхнему краю всех элементов и по центру относительно друг друга (теги параграфа убирать не нужно)

align-self позволяет выравнивать элементы по отдельности (только первый квадрат будет выровнен по центру)

```
.#container {
 align-items: flex-start;
}
```

```
.square#one {
 align-self: center;
}
```

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

CSS Grid Layout — это модуль CSS, который определяет систему макета на основе двумерной сетки, оптимизированную для дизайна пользовательского интерфейса. В заранее определенной гибкой сетке макета мы можем размещать дочерние элементы. Если мы посмотрим на данные с сайта Can I use то увидим, что CSS Grid Layout на данный момент поддерживается большинством современных браузеров.

Grid Layout позволяет кардинально изменять структуру визуального макета, не требуя соответствующих изменений разметки. Комбинируя медиа-запросы со свойствами CSS, можно легко добавлять адаптивность для отображения на мобильных устройствах, сохраняя при этом более идеальную семантическую структуру HTML кода.

Для того чтобы создать грид, для контейнера с классом .grid мы добавляем свойство display: grid. У display: inline-grid, разница с обычным grid такая же, как у inline-block с block.

```
.grid {
 display: grid;
}
```

С этого момента контейнер .grid будет являться гридом

Структура

- Грид-контейнер — элемент, в котором находится сетка грида.
- Грид-линии — невидимые вертикальные и горизонтальные линии, разделяющие грид на ячейки. У грид линий есть нумерация, а также им можно задавать имена. На изображении помечены красными (вертикальные линии) и фиолетовыми стрелками (горизонтальные линии).
- Грид-полосы — пространство, которое ограничено парой соседних грид-линий. Бывают вертикальные и горизонтальные.
- Грид-ячейки — то, что получается на пересечении двух грид-полос. По аналогии с ячейками таблицы. На картинке это синие блоки с буквами, в количестве шести штук.
- Грид-области — прямоугольники из смежных грид-ячеек. Каждая грид-область ограничена двумя парами грид-линий (парой вертикальных и парой горизонтальных).
- Грид-интервалы — пустые пространства между соседними грид-полосами.

Примеры свойств

Для задания отступов используются свойства grid-gap, row-gap и column-gap:

- grid-gap - отступы между ячейками грида, свойство объединяет ряды и колонки;
- row-gap - это расстояние между рядами;
- column-gap - это расстояние между колонками.

grid-template-columns добавляет колонки (три с одинаковой шириной):

```
.grid {
 display: grid;
 grid-gap: 10px;
 grid-template-columns: 1fr 1fr 1fr;
}
```

Свойство grid-template-rows позволяет добавлять строки в грид, аналогично колонкам:

```
.grid {
 display: grid;
 grid-gap: 10px;
 grid-template-columns: 1fr 1fr 1fr;
 grid-template-rows: 1fr 1fr;
}
```

Если мы не указываем grid-template-rows, это значит, что в случае надобности строки будут добавляться автоматически и будут неявными.

Высота строк и столбцов на неявных гридах должна выставляться с помощью свойств

- grid-auto-rows - высота строк;
- grid-auto-columns - ширина столбцов;
- grid-auto-flow - позволяет указать, что нужно использовать для авто-размещения элементов, строки или колонки. По умолчанию его значение равно row.

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

Понятие

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

Назначение

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

Примеры

Можно выделить три популярных препроцессора:

- Less
- Sass (SCSS)
- Stylus

Примеры кода

Рассмотрим пример кода на препроцессоре SASS

Вы можете хранить в переменных цвета, стеки шрифтов или любые другие значения CSS, которые вы хотите использовать. Чтобы создать переменную в Sass нужно использовать символ $.

Использование переменных

SCSS Sass

```
$font-stack:    Helvetica, sans-serif
$primary-color: #333

body
  font: 100% $font-stack
  color: $primary-color
```

CSS

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Вложенности

Sass позволит вам вкладывать CSS селекторы таким же образом, как и в визуальной иерархии HTML.

SCSS Sass

```
nav
  ul
    margin: 0
    padding: 0
    list-style: none

  li
    display: inline-block

  a
    display: block
    padding: 6px 12px
    text-decoration: none
```

CSS

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Математические операторы

Использовать математику в CSS очень полезно. Sass имеет несколько стандартных математических операторов, таких как +, -, *, / и %. В нашем примере мы совершаем простые математические вычисления для расчета ширины aside и article.

SCSS Sass

```
.container
  width: 100%


article[role="main"]
  float: left
  width: 600px / 960px * 100%


aside[role="complementary"]
  float: right
  width: 300px / 960px * 100%
```

CSS

```
.container {
  width: 100%;
}

article[role="main"] {
  float: left;
  width: 62.5%;
}

aside[role="complementary"] {
  float: right;
  width: 31.25%;
}
```

Больше информации и примеров можно найти тут -> https://sass-scss.ru/guide/

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

CSS-фреймворк — фреймворк, созданный для упрощения работы верстальщика, быстроты разработки и исключения максимально возможного числа ошибок вёрстки (проблемы совместимости различных версий браузеров и т. д.). Как и библиотеки скриптовых языков программирования, CSS-библиотеки, обычно имеющие вид внешнего css-файла, «подключаются» к проекту (добавляются в заголовок веб-страницы). Более функциональные фреймворки также имеют больше функций и дополнительных функций на основе JavaScript, но в основном ориентированы на дизайн и ориентированы на интерактивные шаблоны пользовательского интерфейса.

Существует большое количество различных фреймворков. Здесь приведены 4 примеры известных примеров.

Примеры:

bootstrap

Один из самых известных CSS-фреймворков на сегодняшний день. Имеет в своем составе шаблоны для отрисовки кнопок, сайдбаров, навигационных панелей, форм и других элементов сайта. Включает себя JavaScript-расширения.

Основные инструменты bootstrap — шаблоны, @media, формы, навигация, алерты, типографика и конечно же, сетки. Bootstrap совместим со всеми основными современными браузерами, но в старых версиях браузеров могут быть проблемы. Поддерживает адаптивность. Использует языки Less и Sass.

Skeleton

Основан на JavaScript и CSS, используется для быстрого и безошибочного создания адаптивных сайтов которые корректно отображаются как на больших мониторах, так и на экранах маленьких гаджетов. Большой плюс Skeleton — его можно приспособить под любой дизайн. Сетка разметки содержит 12 колонок и имеет базовую ширину 960 пикселей. Совместим со всеми основными браузерами.

Полная свобода для разработчика без навязывания стилей. Предоставляет максимальную гибкость оформления. Имеет несколько вариантов сетки и позиционируется создателями как фреймворк для профессиональных CSS-разработчиков. Использует Less, поддерживает адаптивный дизайн.

foundation

Распространенный CSS-фреймворк. Наряду с другими основными элементами включает в себя несколько HTML шаблонов с различным расположением блоков на странице. Имеет большой набор компонентов на JavaScript. Очень серьезный по своим возможностям и составу фреймворк. Использует Sass. Совместимость кода со всеми основными браузерами.

Pure

Этот легковесный (3.8 Кб) CSS-фреймворк создан Yahoo в 2014 году. За легкость приходится платить универсальностью. Pure не предлагает богатую библиотеку компонентов – он сконцентрирован на лейаутах и меню. И конечно, ни капли JavaScript.

Главные фичи Pure.css:

- Крошечный размер.
- Чистый CSS – для встраивания нужен лишь один файл.
- Меню на любой вкус – вертикальные, горизонтальные, выпадающие.
- Удобная работа с элементами форм.

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

Bootstrap (также известен как Twitter Bootstrap) — CSS-фреймворк, свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

Bootstrap позволяет верстать сайты в несколько раз быстрее, чем на «чистом» CSS и JavaScript.

Основная область его применения – это фронтенд-разработка сайтов и интерфейсов админок.

Основные инструменты Bootstrap:

- Сетки — заранее заданные размеры колонок, которые можно сразу же использовать, например, ширина колонки 140 px относится к классу .span2 (.col-md-2 в третьей версии фреймворка), который можно использовать в CSS-описании документа.
- Шаблоны — фиксированный или резиновый шаблон документа.
- Типографика — описания шрифтов, определение некоторых классов для шрифтов, таких как код, цитаты и т. п.
- Медиа — предоставляет некоторое управление изображениями и видео.
- Таблицы — средства оформления таблиц, вплоть до добавления функциональности сортировки.
- Формы — классы для оформления форм и некоторых событий, происходящих с ними.
- Навигация — классы оформления для панелей, вкладок, перехода по страницам, меню и панели инструментов.
- Алерты — оформление диалоговых окон, подсказок и всплывающих окон.

Подключить локальную таблицу стилей можно с помощью:

<link rel="stylesheet" href="bootstrap/css/style.js" >

Предварительно создав папку css с файлом style.js.

Файл main.js для будущих скриптов можно подключить следующим образом:

<script src="bootstrap/js/main.js" ></script>

Чтобы создать кнопку на странице с помощью Bootstrap, достаточно к ссылке или элементу button добавить всего несколько классов.

<!-- Чтобы сделать ссылку в виде кнопки добавим к ней 2 класса: btn и btn-success -->

Ссылка, оформленная в виде кнопки

Чтобы создать адаптивное изображение, которое масштабируется вместе с родительским элементом используют такой код:

<img src="..." class="img-fluid" alt="...">

44. Общая характеристика языка программирования JavaScript.

JavaScript – это мультипарадигменный интерпретируемый язык программирования, активно применяющийся в веб-разработке для придания интерактивности HTML-страницам. В то же время он применяется в разработке серверной логики веб-сайтов, различных приложений и игр и даже операционных систем.

JavaScript – это интерпретируемый язык, то есть исходный код скриптов не нуждается в компиляции.

JavaScript, в основном, применяется как объектно-ориентированный язык программирования, однако на самом деле он является мультипарадигменным и поддерживает объектно-ориентированный, императивный и функциональный стили.

К данным применяется слабый (динамический) контроль типов.

Как и Java, интерпретаторы этого языка автоматически управляют памятью и очищают неиспользуемые блоки, то есть программисту не нужно следить за выделением и очисткой памяти.

Функции в этом языке представлены как объекты первого класса, то есть могут создаваться по мере выполнения программы. Также их можно присваивать переменным и полям класса, как и любые другие значения.

JavaScript отличается от большинства объектно-ориентированных языков программирования тем, что в нём отсутствует понятие класса. Все объекты являются копиями уже существующего экземпляра объекта.

JavaScript не поддерживает разделение пространства имён, то есть все переменные в нём являются глобальными.

Все библиотеки и фреймворки подключаются методом непосредственного выполнения кода. У JavaScript отсутствует менеджер пакетов, поэтому для подключения дополнительных модулей их код просто вставляется в HTML-страницу.

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

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.

Web-браузеры, такие как Navigator (2.0 и более поздние версии), 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.

Пример:

```
<script>
function moveon() {
 // Вывести модальный диалог, чтобы получить ответ пользователя
 var answer = confirm("Ready to move on?");
       // Если пользователь щелкнул на кнопке "OK", заставить броузер загрузить новую страницу
       if (answer) window.location = "http://google.com";
 }
 // Запустить функцию, объявленную выше, через 1 минуту (60000 миллисекунд).
 setTimeout(moveon, 60000);
</script>
```

46. JavaScript. Инструкции, комментарии, объявление переменных.

Инструкции

Т.к. JS является полноценным ЯП, хоть и со слабой типизацией, то в нем используются типичные инструкции, характерные и для других языков прграммирования по типу Python, Java, C# и т.д.
break
Прерывает текущую инструкцию цикла, ветвления или инструкцию с меткой и передаёт управление на инструкцию, следующую за прерываемой.
continue
Прерывает выполнение инструкции в текущей итерации текущего цикла или цикла с меткой и продолжает выполнение цикла со следующей итерации.
if...else
Выполняет инструкцию, если указанное условие является истинным. Если условие ложно, выполняет другую инструкцию.
try...catch
Помечает блок инструкций и определяет реакцию на возникновение исключения внутри помеченного блока инструкций.
и множество других

Комментарии

Комментарии в JS существуют двух типов:

- Многострочные: /* пример многострочного комментария */
- Однострочные: // пример однострочного комментария

Используются для каких-либо заметок для себя или других, смысл такой же, как и во всех других ЯП.

Объявление переменных

let

Объявляет локальную переменную в области видимости блока, необязательно инициализирует её значением.

const

Объявляет именованную константу только для чтения.

47. JavaScript. Простые типы данных.

Есть восемь основных типов данных в JavaScript.

В JavaScript есть 8 основных типов.

1. number для любых чисел: целочисленных или чисел с плавающей точкой; целочисленные значения ограничены диапазоном $\pm(2^{53}-1)$.
2. bigint для целых чисел произвольной длины.
3. string для строк. Строка может содержать ноль или больше символов, нет отдельного символьного типа.
4. boolean для true/false.
5. null для неизвестных значений – отдельный тип, имеющий одно значение null.
6. undefined для неприсвоенных значений – отдельный тип, имеющий одно значение undefined.
7. object для более сложных структур данных.
8. symbol для уникальных идентификаторов.

Число (number)

```
let n = 123;
n = 12.345;
```

Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: Infinity, -Infinity и NaN.

Математические операции в JavaScript «безопасны». Мы можем делать что угодно: делить на ноль, обращаться с нечисловыми строками как с числами и т.д. Скрипт никогда не остановится с фатальной ошибкой (не «умрёт»). В худшем случае мы получим NaN как результат выполнения.

BigInt

```
const bigInt = 1234567890123456789012345678901234567890n;
```

Тип BigInt был добавлен в JavaScript, чтобы дать возможность работать с целыми числами произвольной длины. Чтобы создать значение типа BigInt, необходимо добавить n в конец числового литерала.

Строка (string)

Строка (string) в JavaScript должна быть заключена в кавычки.

```
let str = "Привет";
let str2 = 'Одинарные кавычки тоже подойдут';
let phrase = `Обратные кавычки позволяют встраивать переменные ${str}`;
```

В JavaScript существует три типа кавычек.

- Двойные кавычки: "Привет".

- Одинарные кавычки: 'Привет'.
- Обратные кавычки: `Привет`.

Булевый (логический) тип (boolean)

```
let nameFieldChecked = true; // да, поле отмечено
let ageFieldChecked = false; // нет, поле не отмечено
```

Булевый тип (boolean) может принимать только два значения: true (истина) и false (ложь).

Значение «null»

```
let age = null;
```

Специальное значение null не относится ни к одному из типов, описанных выше. Оно формирует отдельный тип, который содержит только значение null. В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках.

Значение «undefined»

```
let age;
alert(age); // выведет "undefined"
```

Специальное значение undefined также стоит особняком. Оно формирует тип из самого себя так же, как и null. Оно означает, что «значение не было присвоено». Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет undefined.

Объекты и символы

Тип object (объект) – особенный. Все остальные типы называются «примитивными», потому что их значениями могут быть только простые значения (будь то строка, или число, или что-то ещё). В объектах же хранят коллекции данных или более сложные структуры.

48. JavaScript. Базовые математические и логические операторы.

Математические операторы

Поддерживаются следующие математические операторы:

- Сложение +
- Вычитание -
- Умножение *
- Деление /
- Взятие остатка от деления %,
- Возведение в степень **.

Первые четыре оператора очевидны, а про % и ** стоит сказать несколько слов.

Взятие остатка %

Оператор взятия остатка %, несмотря на обозначение, никакого отношения к процентам не имеет. Результат a % b – это остаток от целочисленного деления a на b. Например:

```
alert( 5 % 2 ); // 1, остаток от деления 5 на 2
alert( 8 % 3 ); // 2, остаток от деления 8 на 3
```

Возведение в степень **

В выражении a ** b оператор возведения в степень умножает a на само себя b раз. Например:

```
alert( 2 ** 2 ); // 4 (2 умножено на себя 2 раза)
alert( 2 ** 3 ); // 8 (2 * 2 * 2, 3 раза)
alert( 2 ** 4 ); // 16 (2 * 2 * 2 * 2, 4 раза)
```

Логические операторы

В JavaScript есть четыре логических оператора: || (ИЛИ), && (И) и ! (НЕ), ?? (Оператор нулевого слияния)

|| (ИЛИ)

Оператор «ИЛИ» выглядит как двойной символ вертикальной черты:

```
result = a || b;
```

Традиционно в программировании ИЛИ предназначено только для манипулирования булевыми значениями: в случае, если какой-либо из аргументов true, он вернёт true, в противоположной ситуации возвращается false.

&& (И)

Оператор И пишется как два амперсанда &&:

result = a && b;

В традиционном программировании И возвращает true, если оба аргумента истинны, а иначе – false:

Оператор && выполняет следующие действия:

- Вычисляет операнды слева направо.
- Каждый операнд преобразует в логическое значение. Если результат false, останавливается и возвращает исходное значение этого операнда.
- Если все операнды были истинными, возвращается последний.

! (НЕ)

Оператор НЕ представлен восклицательным знаком !. Синтаксис довольно прост:

result = !value;

Оператор принимает один аргумент и выполняет следующие действия:

1. Сначала приводит аргумент к логическому типу true/false.
2. Затем возвращает противоположное значение.

?? (Оператор нулевого слияния)

Оператор нулевого слияния представляет собой два вопросительных знака ??. Так как он обрабатывает null и undefined одинаковым образом, то для этой статьи мы введём специальный термин. Для краткости будем говорить, что значение «определено», если оно не равняется ни null, ни undefined.

Результат выражения a ?? b будет следующим:

- если a определено, то a
- если a не определено, то b.

Иначе говоря, оператор ?? возвращает первый аргумент, если он не null/undefined, иначе второй.

49. JavaScript. Операции со строками.

JavaScript — это полноценный динамический язык программирования, который применяется к HTML документу, и может обеспечить динамическую интерактивность на веб-сайтах.

Любые текстовые данные в JavaScript считаются строками. Это примитивный тип, но язык позволяет работать с ним так, будто он является объектом. В том числе — использовать встроенные в JS методы строк. Важно: при использовании методов создается новая строка, которая записывается в ту же переменную вместо старой строки.

Для создания строк мы можем как напрямую присваивать переменной или константе строку:

```
const name = "Tom";
```

Для работы со строками предназначен объект **String**, поэтому также можно использовать конструктор String:

```
const name = new String("Tom");
```

Но как правило, используется первый более краткий способ. В первом случае JavaScript при необходимости автоматически преобразует переменную примитивного типа в объект String. Объект String имеет большой набор свойств и методов, с помощью которых мы можем манипулировать строками.

- .toLowerCase() - Преобразует символы в строке в нижний регистр.
- .toUpperCase() - Преобразует символы в строке в верхний регистр
- .concat - Объединяет две или более строки и возвращает одну строку

  ```
  "Hello".concat("World"); // "HelloWorld"
  ```

- .split - Разбивает строку в массив по указанному разделителю, которым может быть подстрока или регулярное выражение. Вторым параметром можно указать ограничитель.

  ```
  "Hello World".split(" ", 1); //["Hello"]
  ```

- .repeat - Принимает в качестве параметра число и повторяет строку указанное количество раз.

  ```
  "Hello ".repeat(3); // "Hello Hello Hello "
  ```

- .charAt(i) - Возвращает символ по указанному индексу
- .includes - Проверяет, содержит ли строка указанную подстроку. Возвращает значение true или false. Вторым параметром можно указать позицию в строке, с которой следует начать поиск.
- indexOf - Возвращает индекс первого найденного вхождения указанного значения. Поиск ведётся от начала до конца строки. Если совпадений нет, возвращает -1. Вторым параметром можно передать позицию, с которой следует начать поиск.
- endsWith - Проверяет, заканчивается ли строка символами, заданными первым параметром. Возвращает true или false. Есть второй необязательный параметр — ограничитель по диапазону поиска. По умолчанию он равен длине строки. startsWith работает аналогично.

- slice - Извлекает часть строки и возвращает новую строку. Обязательный параметр — начало извлечения. Вторым параметром можно установить границу (по умолчанию — до конца строки).
- replace - Ищет в строке указанное значение или регулярное выражение и возвращает новую строку, в которой выполнена замена на второй параметр. Можно заменить найденные значения другой строкой или передать функцию для работы над совпадениями.
- trim - Обрезает пробелы с обоих концов строки.

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

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.

Создание:

Легче всего создать массив с помощью литерала, который представляет собой простой список разделенных запятыми элементов массива в квадратных скобках. Значения в литерале массива не обязательно должны быть константами - это могут быть любые выражения, в том числе и литералы объектов:

```
var empty = [];                                    // Пустой массив
var numbers = [2, 3, 5, 7, 11];          // Массив с пятью числовыми элементами
var misc = [ 1.1, true, "a", ];              // 3 элемента разных типов + завершающая запятая
var base = 1024;
var table = [base, base+1, base+2, base+3];              // Массив с переменными
var arrObj = [[1,{x:1, y:2}], [2, {x:3, y:4}]]; // 2 массива внутри, содержащие объекты
```

Другой способ создания массива состоит в вызове конструктора Array(). Вызвать конструктор можно тремя разными способами:

- Вызвать конструктор без аргументов:

```
var arr = new Array();
```

В этом случае будет создан пустой массив, эквивалентный литералу []

- Вызвать конструктор с единственным числовым аргументом, определяющим длину массива:

```
var arr = new Array(10);
```

В этом случае будет создан пустой массив указанной длины. Такая форма вызова конструктора Array() может использоваться для предварительного распределения памяти под массив, если заранее известно количество его элементов.

- Явно указать в вызове конструктора значения первых двух или более элементов массива или один нечисловой элемент:

```
var arr = new Array(5, 4, 3, 2, 1, "тест");
```

В этом случае аргументы конструктора становятся значениями элементов нового массива. Использование литералов массивов практически всегда проще, чем подобное применение конструктора Array().

Использование:

Доступ к элементам массива осуществляется с помощью оператора []. Слева от скобок должна присутствовать ссылка на массив. Внутри скобок должно находиться произвольное выражение, возвращающее неотрицательное целое значение. Этот синтаксис пригоден как для чтения, так и для записи значения элемента массива. Следовательно, допустимы приведенные далее JavaScript-инструкции:

```
// Прочитать элемент 0

var value = arr[0];

// Записать значение в элемент 1

arr[1] = 3.14;
```

Массивы являются специализированной разновидностью объектов. Квадратные скобки, используемые для доступа к элементам массива, действуют точно так же, как квадратные скобки, используемые для доступа к свойствам объекта. Интерпретатор JavaScript преобразует указанные в скобках числовые индексы в строки - индекс 1 превращается в строку "1" - а затем использует строки как имена свойств.

Основные операции:

Особенность массивов состоит в том, что при использовании имен свойств, которые являются неотрицательными целыми числами, массивы автоматически определяют значение свойства length.

Самый простой способ добавить элементы в массив заключается в том, чтобы присвоить значения новым индексам. Для добавления одного или более элементов в конец массива можно также использовать метод push():

```
var arr = [];               // Создать пустой массив

arr.push('zero');           // Добавить значение в конец
```

Удалять элементы массива можно с помощью оператора delete, как обычные свойства объектов:

```
var arr = [1,2,'three'];

delete arr[2];
```

51. JavaScript. Условные операторы.

Иногда необходимо выполнять различные действия в зависимости от условий. Для этого можно использовать инструкцию if и условный оператор ?, который также называют оператором «вопросительный знак».

- Инструкция “if”

Инструкция if(...) вычисляет условие в скобках и, если результат true, то выполняется блок кода. Инструкция if (…) вычисляет выражение в скобках и преобразует результат к логическому типу. Правила преобразования типов:

- Число 0, пустая строка "", null, undefined и NaN становятся false.
- Остальные значения становятся true.

Инструкция if может содержать необязательный блок «else» («иначе»). Он выполняется, когда условие ложно. Иногда, нужно проверить несколько вариантов условия. Для этого используется блок else if. Пример:

```
let year = prompt('В каком году была опубликована спецификация ECMAScript-2015?', '');

if (year < 2015) {

  alert( 'Это слишком рано...' );

} else if (year > 2015) {

  alert( 'Это поздновато' );

} else {

  alert( 'Верно!' );

}
```

- Условный оператор “?”

Иногда нужно определить переменную в зависимости от условия.

Оператор представлен знаком вопроса ?. Его также называют «тернарный», так как этот оператор, единственный в своём роде, имеет три аргумента. Синтаксис:

```
let result = условие ? значение1 : значение2;
```

Сначала вычисляется *условие*: если оно истинно, тогда возвращается *значение1*, в противном случае – *значение2*.

52. JavaScript. Операторы циклов.

При написании скриптов зачастую встаёт задача сделать однотипное действие много раз. Для многократного повторения одного участка кода предусмотрены циклы.

- Цикл while

Цикл while имеет следующий синтаксис:

```
while (condition) {
  // тело цикла
}
```

Код из тела цикла выполняется, пока условие condition истинно.

- Цикл do..while

Проверку условия можно разместить под телом цикла, используя специальный синтаксис do..while:

```
do {
  // тело цикла
} while (condition);
```

Цикл сначала выполнит тело, а затем проверит условие condition, и пока его значение равно true, он будет выполняться снова и снова.

- Цикл for

Более сложный, но при этом самый распространённый цикл — цикл for. Выглядит он так:

```
for (начало; условие; шаг) {
  // ... тело цикла ...
}
```

Пример:

```
for (let i = 0; i < 3; i++) { // выведет 0, затем 1, затем 2
  alert(i);
}
```

Любая часть for может быть пропущена.

Обычно цикл завершается при вычислении условия в false. Но мы можем выйти из цикла в любой момент с помощью специальной директивы break. Сочетание «бесконечный цикл + break» – отличная

штука для тех ситуаций, когда условие, по которому нужно прерваться, находится не в начале или конце цикла, а посередине или даже в нескольких местах его тела.

Директива continue позволяет перейти к следующей итерации. При её выполнении цикл не прерывается, а переходит к следующей итерации (если условие все ещё равно true). Её используют, если понятно, что на текущем повторе цикла делать больше нечего.

53. JavaScript. Функции.

Зачастую необходимо повторять одно и то же действие во многих частях программы. Например, необходимо красиво вывести сообщение при приветствии посетителя, при выходе посетителя с сайта, ещё где-нибудь. Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «строительными блоками» программы. Примеры встроенной функции – это alert(message). Но можно создавать и свои.

Для создания функций можно использовать объявление функции.

Вначале идёт ключевое слово function, после него имя функции, затем список параметров в круглых скобках через запятую и, наконец, код функции, также называемый «телом функции», внутри фигурных скобок.

function имя(параметры) {

...тело...

}

Новая функция может быть вызвана по имени: *имя()*. Этот вызов выполняет код функции.

Локальные переменные: Переменные, объявленные внутри функции, видны только внутри этой функции.

Внешние переменные: Функция обладает полным доступом к внешним переменным и может изменять их значение. Внешняя переменная используется, только если внутри функции нет такой локальной. Если одноимённая переменная объявляется внутри функции, тогда она перекрывает внешнюю.

Желательно сводить использование глобальных переменных к минимуму. В современном коде обычно мало или совсем нет глобальных переменных. Хотя они иногда полезны для хранения важнейших «общепроектовых» данных.

Параметры: Можно передать внутрь функции любую информацию, используя параметры (также называемые аргументами функции). Если параметр не указан, то его значением становится undefined. Если мы хотим задать параметру значение по умолчанию, мы должны указать это значение после = при объявлении функции.

54. JavaScript. Обработка исключений.

https://learn.javascript.ru/try-catch

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/try...catch

Конструкция try...catch пытается выполнить инструкции в блоке try, и, в случае ошибки, выполняет блок catch.

Конструкция try содержит блок try, в котором находится одна или несколько инструкций (Блок ({}) обязательно должен присутствовать, даже если выполняется всего одна инструкция), и хотя бы один блок catch или finally. Таким образом, есть три основные формы конструкции try:

try {...} catch {...}

try {...} finally {...}

try {...} catch {...} finally {...}

Блок catch содержит инструкции, которые будут выполнены, если в блоке try произошла ошибка. Если любая инструкция в блоке try выбрасывает исключение, то управление сразу же переходит в блок catch. Если в блок try не было выброшено исключение, то блок catch не выполняется.

Блок finally выполнится после выполнения блоков try и catch, но перед инструкциями, следующими за конструкцией try...catch. Он выполняется всегда, в независимости от того, было исключение или нет.

55. JavaScript. Литерация объектов. JSON.

https://learn.javascript.ru/json

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/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 любом другом языке.

JavaScript предоставляет методы:

JSON.stringify для преобразования объектов в JSON.
JSON.parse для преобразования JSON обратно в объект.
JSON поддерживает следующие типы данных:

- Объекты { ... }
- Массивы [...]
- Примитивы:
    - строки,
    - числа,
    - логические значения true/false,
    - null.

JSON является независимой от языка спецификацией для данных, поэтому JSON.stringify пропускает некоторые специфические свойства объектов JavaScript.

А именно:

- Свойства-функции (методы).
- Символьные ключи и значения.
- Свойства, содержащие undefined.

Чтобы декодировать JSON-строку, нам нужен другой метод с именем JSON.parse.

56. JavaScript. Классы в ES6.

https://learn.javascript.ru/class

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Classes

Классы в JavaScript были введены в ECMAScript 2015 и представляют собой синтаксический сахар над существующим в JavaScript механизмом прототипного наследования. Синтаксис классов не вводит новую объектно-ориентированную модель, а предоставляет более простой и понятный способ создания объектов и организации наследования.

Базовый синтаксис для классов выглядит так:

```
class MyClass {
  prop = value; // свойство
  constructor(...) { // конструктор
    //
  }
  method(...) {} // метод
  get something(...) {} // геттер
  set something(...) {} // сеттер
  [Symbol.iterator]() {} // метод с вычисляемым именем (здесь - символом)
  //
}
```

MyClass технически является функцией (той, которую мы определяем как constructor), в то время как методы, геттеры и сеттеры записываются в MyClass.prototype.

57. JavaScript. Функции высших порядков.

https://habr.com/ru/company/ruvds/blog/428570/

Функции первого класса

Если вы изучаете JavaScript, вы могли слышать, что в языке функции рассматриваются как объекты первого класса. Это так из-за того, что в JavaScript, как и в других языках, поддерживающих функциональное программирование, функции являются объектами.

В частности, в JS функции представлены в виде объектов особого типа — это объекты типа Function

Функции высшего порядка — это функции, которые работают с другими функциями, либо принимая их в виде параметров, либо возвращая их. Проще говоря, функцией высшего порядка называется такая функция, которая принимает функцию как аргумент или возвращает функцию в виде выходного значения.

Например, встроенные функции JavaScript Array.prototype.map, Array.prototype.filter и Array.prototype.reduce являются функциями высшего порядка.

Рассмотрим примеры использования встроенных в JS функций высшего порядка и сравним такой подход с выполнением аналогичных действий без использования таких функций.

Метод Array.prototype.map

Метод map() создаёт новый массив, вызывая, для обработки каждого элемента входного массива, коллбэк, переданный ему в виде аргумента. Этот метод берёт каждое возвращённое коллбэком значение и помещает его в выходной массив.

Функция обратного вызова, передаваемая map(), принимает три аргумента: element (элемент), index (индекс) и array (массив).

58. JavaScript. Наследование.

https://learn.javascript.ru/prototype-inheritance

https://learn.javascript.ru/class-inheritance

В объектно-ориентированном программировании класс – это расширяемый шаблон кода для создания объектов, который устанавливает в них начальные значения (свойства) и реализацию поведения (методы).

1. Чтобы унаследовать от класса: class Child extends Parent:

При этом Child.prototype.__proto__ будет равен Parent.prototype, так что методы будут унаследованы.

2. При переопределении конструктора:

Обязателен вызов конструктора родителя super() в конструкторе Child до обращения к this.

3. При переопределении другого метода:

Мы можем вызвать super.method() в методе Child для обращения к методу родителя Parent.

4. Внутренние детали:

Методы запоминают свой объект во внутреннем свойстве [[HomeObject]]. Благодаря этому работает super, он в его прототипе ищет родительские методы.

Поэтому копировать метод, использующий super, между разными объектами небезопасно.

59. JavaScript. Деструктуризация массивов и объектов.

https://learn.javascript.ru/destructuring-assignment

- Деструктуризация позволяет разбивать объект или массив на переменные при присвоении.
- Полный синтаксис для объекта:
  let {prop : varName = default, ...rest} = object
  Свойства, которые не были упомянуты, копируются в объект rest.
- Полный синтаксис для массива:
  let [item1 = default, item2, ...rest] = array
  Первый элемент отправляется в item1; второй отправляется в item2, все остальные элементы попадают в массив rest.
- Можно извлекать данные из вложенных объектов и массивов, для этого левая сторона должна иметь ту же структуру, что и правая.

60. JavaScript. Стрелочные функции.

https://learn.javascript.ru/arrow-functions-basics

https://learn.javascript.ru/arrow-functions

Стрелочные функции очень удобны для простых действий, особенно для однострочных.

Они бывают двух типов:

1. Без фигурных скобок: (...args) => expression – правая сторона выражение: функция вычисляет его и возвращает результат. Скобки можно не ставить, если аргумент только один: n => n * 2.
2. С фигурными скобками: (...args) => { body } – скобки позволяют нам писать несколько инструкций внутри функции, но при этом необходимо явно вызывать return, чтобы вернуть значение.

Стрелочные функции:

- Не имеют this.
- Не имеют arguments.
- Не могут быть вызваны с new.
- У них также нет super

61. JavaScript. Промисы.

https://learn.javascript.ru/promise-basics

https://learn.javascript.ru/promise-api
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Promise

Синтаксис создания Promise:

```
let promise = new Promise(function(resolve, reject) {
  // функция-исполнитель (executor)
  // "певец"
});
```

Функция, переданная в конструкцию new Promise, называется исполнитель (executor). Когда Promise создаётся, она запускается автоматически. Она должна содержать «создающий» код, который когда-нибудь создаст результат. В терминах нашей аналогии: исполнитель – это «певец».

Её аргументы resolve и reject – это колбэки, которые предоставляет сам JavaScript. Наш код – только внутри исполнителя.

Когда он получает результат, сейчас или позже – не важно, он должен вызвать один из этих колбэков:

- resolve(value) — если работа завершилась успешно, с результатом value.
- reject(error) — если произошла ошибка, error – объект ошибки.

Итак, исполнитель запускается автоматически, он должен выполнить работу, а затем вызвать resolve или reject.

У объекта promise, возвращаемого конструктором new Promise, есть внутренние свойства:

- state («состояние») — вначале "pending" («ожидание»), потом меняется на "fulfilled" («выполнено успешно») при вызове resolve или на "rejected" («выполнено с ошибкой») при вызове reject.
- result («результат») — вначале undefined, далее изменяется на value при вызове resolve(value) или на error при вызове reject(error).

Promise API

- Promise.all(promises) – ожидает выполнения всех промисов и возвращает массив с результатами. Если любой из указанных промисов вернёт ошибку, то результатом работы Promise.all будет эта ошибка, результаты остальных промисов будут игнорироваться.
- Promise.allSettled(promises) (добавлен недавно) – ждёт, пока все промисы завершатся и возвращает их результаты в виде массива с объектами, у каждого объекта два свойства: state: "fulfilled", если выполнен успешно или "rejected", если ошибка, value – результат, если успешно или reason – ошибка, если нет.
- Promise.race(promises) – ожидает первый выполненный промис, который становится его результатом, остальные игнорируются.

- Promise.resolve(value) – возвращает успешно выполнившийся промис с результатом value.
- Promise.reject(error) – возвращает промис с ошибкой error.

62. DOM. Браузерное окружение в JavaScript.

https://learn.javascript.ru/browser-environment

https://developer.mozilla.org/ru/docs/Web/API/Document_Object_Model

Document Object Model, сокращённо DOM – объектная модель документа, которая представляет все содержимое страницы в виде объектов, которые можно менять.

Объект document – основная «входная точка». С его помощью мы можем что-то создавать или менять на странице.

Имеется корневой объект window, который выступает в 2 ролях:

- Во-первых, это глобальный объект для JavaScript-кода, об этом более подробно говорится в главе Глобальный объект.

- Во-вторых, он также представляет собой окно браузера и располагает методами для управления им.

Спецификация DOM описывает структуру документа и предоставляет объекты для манипуляций со страницей. Существуют и другие, отличные от браузеров, инструменты, использующие DOM.

Например, серверные скрипты, которые загружают и обрабатывают HTML-страницы, также могут использовать DOM. При этом они могут поддерживать спецификацию не полностью.

Объектная модель браузера (Browser Object Model, BOM) – это дополнительные объекты, предоставляемые браузером (окружением), чтобы работать со всем, кроме документа.

- Объект navigator даёт информацию о самом браузере и операционной системе. Среди множества его свойств самыми известными являются: navigator.userAgent – информация о текущем браузере, и navigator.platform – информация о платформе (может помочь в понимании того, в какой ОС открыт браузер – Windows/Linux/Mac и так далее).

- location позволяет получить текущий URL и перенаправить браузер по новому адресу.

63. DOM. Навигация. Способы нахождения элементов. Поиск по дереву.

https://learn.javascript.ru/dom-navigation
https://learn.javascript.ru/searching-elements-dom

Получив DOM-узел, мы можем перейти к его ближайшим соседям используя навигационные ссылки.

Есть два основных набора ссылок:

- Для всех узлов: parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling.
- Только для узлов-элементов: parentElement, children, firstElementChild, lastElementChild, previousElementSibling, nextElementSibling.

Некоторые виды DOM-элементов, например таблицы, предоставляют дополнительные ссылки и коллекции для доступа к своему содержимому.

Есть 6 основных методов поиска элементов в DOM:

Метод	Ищет по...	Ищет внутри элемента?	Возвращает живую коллекцию?
`querySelector`	CSS-selector	✓	-
`querySelectorAll`	CSS-selector	✓	-
`getElementById`	`id`	-	-
`getElementsByName`	`name`	-	✓
`getElementsByTagName`	tag or `'*'`	✓	✓
`getElementsByClassName`	class	✓	✓

Безусловно, наиболее часто используемыми в настоящее время являются методы querySelector и querySelectorAll, но и методы getElement(s)By* могут быть полезны в отдельных случаях, а также встречаются в старом коде.

Кроме того:

- Есть метод elem.matches(css), который 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.

64. DOM. Свойства узлов. Изменение свойств элементов.

https://learn.javascript.ru/basic-dom-node-properties
https://learn.javascript.ru/dom-attributes-and-properties

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 это свойство возвращает название тега (записывается в верхнем регистре, за исключением XML-режима). Для узлов-неэлементов nodeName описывает, что это за узел. 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, обычно мы используем data. Можно изменять.

- textContent Текст внутри элемента: HTML за вычетом всех <тегов>. Запись в него 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.

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.

Атрибуты – это то, что написано в HTML.

Свойства – это то, что находится в DOM-объектах.

Методы для работы с атрибутами:

- elem.hasAttribute(name) – проверить на наличие.

- elem.getAttribute(name) – получить значение.

- elem.setAttribute(name, value) – установить значение.

- elem.removeAttribute(name) – удалить атрибут.

- elem.attributes – это коллекция всех атрибутов.

В большинстве ситуаций предпочтительнее использовать DOM-свойства. Нужно использовать атрибуты только тогда, когда DOM-свойства не подходят, когда нужны именно атрибуты, например:

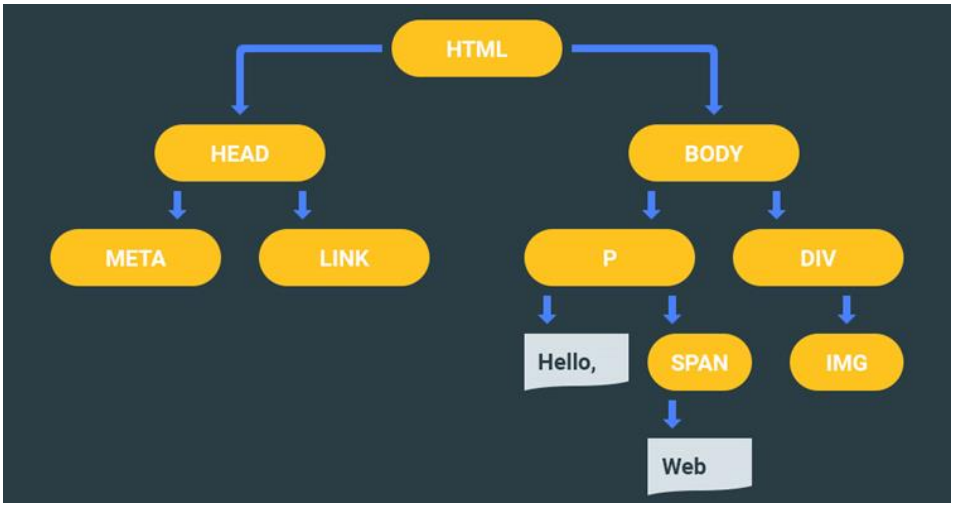
- Нужен нестандартный атрибут. Но если он начинается с data-, тогда нужно использовать dataset.

- Мы хотим получить именно то значение, которое написано в HTML. Значение DOM-свойства может быть другим, например, свойство href – всегда полный URL, а нам может понадобиться получить «оригинальное» значение.

65. DOM. Браузерные события. Обработчики событий.

Основным инструментом работы и динамических изменений на странице является DOM (Document Object Model) – объектная модель, используемая для XML/HTML-документов. Согласно DOM-модели, документ является иерархией, деревом. Каждый HTML-тег образует узел дерева с типом «элемент». Вложенные в него теги становятся дочерними узлами.

DOM – это представление документа в виде дерева объектов, доступное для изменения через JavaScript.



Для реакции на действия посетителя и внутреннего взаимодействия скриптов существуют события. Событие – это сигнал от браузера о том, что что-то произошло. Основные события:

- события мыши:
  - click – происходит, когда кликнули на элемент левой кнопкой мыши
  - contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши
  - mouseover – возникает, когда на элемент наводится мышь
  - mousemove – при движении мыши
- события на элементах управления:
  - submit – посетитель отправил форму <form>
  - reset – посетитель сбросил форму <form>
  - focus – посетитель фокусируется на элементе, например нажимает на <input>
- клавиатурные события:
  - keydown – когда посетитель нажимает клавишу
  - keyup – когда посетитель отпускает клавишу

Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло. Именно благодаря обработчикам JavaScript-код может реагировать на действия посетителя.

Пример:

```
<input value=”Нажми меня” onclick=”alert(‘Клик’)” type=”button”>
```

При клике мышкой на кнопке выполнится код, указанный в атрибуте onclick.

66. DOM. Методы обхода дерева элементов.

Обход по дереву DOM (traversing) в jQuery используется, чтобы "найти" (или отобрать) HTML элементы, основываясь на их положении по отношению к другим элементам. Выбрав некий элемент, затем можно двигаться в любую сторону, пока не будет достигнуто желаемое положение.

jQuery предоставляет множество различных методов, позволяющих перемещаться по DOM.

- Перемещение вверх:
  - parent() – возвращает непосредственный родительский элемент выбранного элемента.
  - parents() – возвращает все родительские элементы вплоть до корневого (<html>) выбранного элемента.
  - parentsUntil() – возвращает все родительские элементы между двумя заданными элементами.
- Перемещение вниз:
  - children() – возвращает все прямые потомки выбранного элемента.
  - find() – возвращает все потомки выбранного элемента вплоть до последнего.
- Проход по одному уровню DOM:
  - siblings() – возвращает все элементы, находящиеся на одном уровне с выбранным элементом.
  - next() – возвращает следующий соседний элемент от выбранного элемента.
  - nextAll() – возвращает все следующие соседние элементы от выбранного элемента.
  - nextUntil() – возвращает все следующие соседние элементы между двумя заданными аргументами.
  - prev(), prevAll(), prevUntil() – аналогично выше описанным методам, но в обратном направлении.
- Обход элементов – фильтрация:
  - Базовые методы фильтрации – это first(), last() и eq(). Они позволяют выбрать заданный элемент основываясь на его позиции в группе элементов.
  - Другие методы фильтрации, вроде filter() и not(), позволяют отобрать элементы, которые либо соответствуют, либо нет, определенным критериям.

67. DOM. Программное создание нового элемента и добавление его в дерево.

DOM-узел можно создать двумя методами:

- document.createElement(tag) – создаёт новый элемент с заданным тегом
- document.createTextNode(text) – создаёт новый текстовый узел с заданным текстом

Пример:

```
let div = document.createElement('div');
div.className = "alert";
div.innerHTML = "<strong>Всем привет!</strong> Вы прочитали важное сообщение.";
```

Пока что элемент создан только в переменной. Мы не можем видеть его на странице, поскольку он не является частью документа. Чтобы увидеть его на странице, нужно вставить его в документ. Делать это можно при помощи различных методов вставок:

- 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.

Пример:

```
let div = document.createElement('div');
div.className = "alert";
div.innerHTML = "<p>Всем привет!</p> ";
document.body.append(div);
```

Если мы хотим вставить HTML именно «как HTML» со всеми тегами и прочим (по аналогии с element.innerHTML), то нужно использовать метод elem.insertAdjacentHTML(where, html).

Первый параметр – это специальное слово, указывающее, куда по отношению к elem производить вставку:

- "beforebegin" – вставить html непосредственно перед elem,
- "afterbegin" – вставить html в начало elem,
- "beforeend" – вставить html в конец elem,
- "afterend" – вставить html непосредственно после elem.

Второй параметр – это HTML-строка.

Пример:

```
<div id="div"></div>
<script>
  div.insertAdjacentHTML('beforebegin', '<p>Привет</p>');
  div.insertAdjacentHTML('afterend', '<p>Пока</p>');
</script>
```

68. Понятие фронтенд фреймворков. Назначение, примеры, использование.

Фреймворк — заготовка, готовая модель в IT для быстрой разработки, на основе которой можно дописать собственный код. Он задает структуру, определяет правила и предоставляет необходимый набор инструментов для создания проекта. В основном фреймворки используются в веб-разработке.

Фронтенд-фреймворки отвечают за внешний вид проекта и не связаны с логикой работы. Позволяют улучшать и добавлять новые пользовательские интерфейсы, создавать одностраничные приложения, анимации и пр. Примерами таких фреймворков могут служить Vue.js, Bootstrap или Foundation.

Так, например, включить Bootstrap в проект можно разными способами:

- Скачать скомпилированные файлы JS и CSS.
- Скачать исходники файлов.
- Включить в проект файлы через CDN.
- Использовать менеджеры проектов: npm, yarn.

Bootstrap использует сеточную структуру, которая делит родительский блок (в котором находится) на 12 одинаковых по размерам частей. Их можно объединять между собой (3-6-3).

Пример:

```
<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>
```

Также этот фреймворк содержит большое число компонентов для создания панели навигации, карусели, форм, поисковой строки и пр.

69. React.js. Общая характеристика. Установка, запуск.

React – это библиотека JavaScript, разработанная Facebook, используемая для создания пользовательских интерфейсов и фронтэнд приложений. Она декларативная, компонентно-ориентированная, более легковесная, чем другие фреймворки - Angular, Vue. Его часто называют фреймворком за его способ совершенно по-новому проектировать приложение. На сегодняшний день React – наиболее популярный фронтенд фреймворк в индустрии.

Назначение:

- Облегчает написание клиентского JavaScript кода
- Использует независимые компоненты, имеющие собственное состояние
- Виртуальный DOM
- JSX - использование JavaScript в разметке
- Помогает при командной разработке

Установка и запуск. Можно воспользоваться пакетом create-react-app, который нужно предварительно установить через npm:

```
npm install -g create-react-app
```

После этого можно уже создать шаблон react-приложения:

```
create-react-app my-app
```

Для запуска проекта перейти в папку и запустить сервер командой start:

```
cd my-app
```

```
npm start
```

70. React.js. Понятие компонента. Способы задания.

React использует независимые компоненты, имеющие собственное состояние (объект, содержащий информацию о том, как этот компонент должен отображаться и вести себя). В случае изменения состояния компонента вызывается метод render, который перерисовывает компонент на экране. Так, можно сказать, компонент реагирует на изменение состояния.

Компоненты позволяют разделить пользовательский интерфейс на независимые, повторно используемые части и работать с каждой из частей отдельно.

Способы задания компонентов:

1) На основе классов:

```
class Welcome extends React.Component {
    render() {
      return <h1>Привет, {this.props.name}</h1>;
    }
   }
```

В данном случае компонентом является класс, который наследуется от класса React.Component.

2) Используя функции:

```
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 называют функциональными.

Два вышеуказанных компонента эквивалентны с точки зрения React.

71. React.js. Состояние компонента. Назначение, использование.

Компоненты React могут иметь состояние, то есть объект, содержащий информацию о том, как этот компонент должен отображаться и вести себя.

В случае изменения состояния компонента вызывается метод render, который перерисовывает компонент на экране. Так, можно сказать, компонент реагирует на изменение состояния.

Также можно оперировать состоянием приложения с помощью менеджера состояний, например Redux или ContextAPI.

Инициализация состояния происходит в конструкторе:

```
constructor(props) {
    super(props)
    this.state = { username: 'johndoe' }
  }
```

Доступ к состоянию можно получить в методе render():

```
render() {
  return(
    <div>
      { this.state.username }
    </div>
  )
}
```

Обновление состояния – setState:

```
handleInputChange(username) {
  this.setState({username})
}
```

72. React.js. JSX.

JSX - JavaScript Syntax Extension – предпроцессор, который добавляет синтаксис XML в код JavaScript.

```
const element = <h1>Привет, мир!</h1>;
```

- Выглядит как XML/HTML.
- Предоставляет простой синтаксис для описания иерархических структур с атрибутами.
- Предоставляет возможность просто включать код JavaScript в описание XML.
- Использовать не обязательно, но он сильно облегчает написание кода.

Пример. Этот JSX-код:

```
<MyButton color="blue" shadowSize={2}>
  Нажми меня
</MyButton>
```

Скомпилируется в:

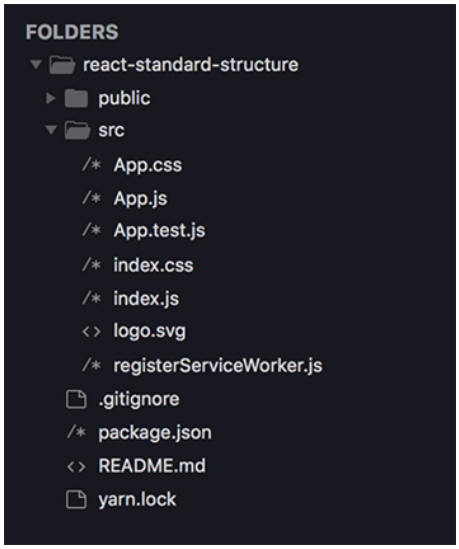
```
React.createElement(
  MyButton,
  { color: 'blue', shadowSize: 2 },
  'Нажми меня'
)
```

73. React.js. Структура проекта. Основные файлы.

Как правило для создания реакт-проекта используется create-react-app, который сам автоматически генерирует базовый проект, в корневой директории которого содержаться файлы: .gitignore, package.json, README.md, yarn.lock.

Кроме того, он создает папки: public и src.

Полная структура проекта выглядит следующим образом:



Важные файлы:

package.json - файл манифеста (зависимости)

public/index.html - одностраничное приложение

src/index.js - точка входа в приложение, рендеринг компонентов

src/index.css - главная таблица стилей.

74. React.js. Передача параметров как свойств.

Передать обработчики событий и другие функции можно через пропсы дочерним компонентам:

<button onClick={this.handleClick}>

Если вы хотите иметь доступ к компоненту-родителю через обработчик, вам нужно привязать функцию к экземпляру компонента .

Способы привязать функцию к экземпляру компонента:

Привязка в конструкторе (ES2015)

```
class Foo extends Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    console.log('По кнопке кликнули');
  }
  render() {
    return <button onClick={this.handleClick}>Нажми на меня</button>;
  }
}
```

Привязка в свойствах класса (предложение-кандидат)

```
class Foo extends Component {
  // Примечание: данный синтаксис находится на стадии разработки и ещё не стандартизирован.
  handleClick = () => {
        console.log('По кнопке кликнули');
  }
  render() {
    return <button onClick={this.handleClick}>Нажми на меня</button>;
  }
}
```

Привязка в методе render()

```
class Foo extends Component {
  handleClick() {
    console.log('По кнопке кликнули');
  }
  render() {
```

```
  return <button onClick={this.handleClick.bind(this)}>Нажми на меня</button>;
 }
```

Стрелочная функция в render()

```
class Foo extends Component {
 handleClick() {
      console.log('По кнопке кликнули');
 }
 render() {
  return <button onClick={() => this.handleClick()}>Нажми на меня</button>;
 }
}
```

75. React.js. Задание стилей.

В HTML можно добавлять ваши стили CSS инлайново. Это почти также, как и в React.

Можно добавлять инлайновые стили компонентам, которые хотим отрендерить. Такие стили записаны как атрибуты и передаются элементу.

Пример:

```
render() {

return(

<div style = {{ backgroundColor: “#44014C”, width: “300px”, minHeight: “200px”}}>

<h2 style={{ padding: "10px 20px", textAlign: "center", color: "white"}}>ToDo</h2>

        )

}
```

Метод 2: Стили через переменные:

```
render() {

        return (

        <div style={itemStyle}>

        <p> {

                this.props.todo.title

        } </p>

        </div>

        );

  }

}


const itemStyle = { backgroundColor: '#f4f4f4' }
```

Метод 3: Стили через функции

```
getStyle = () => {
        if (this.props.todo.completed) {
        return{
        textDecoration: 'line-through'
        }
        }
        else{
        return {
        textDecoration: 'none'
        }
        }
}
render() {
        return (
        <div style={this.getStyle()}>
        <p> {
                this.props.todo.title
        } </p>
        </div>
        );
}
```

Метод 4: styled-components

Для начала надо это установить styled-components :

$ npm install --save styled-components

Затем импортировать пакет:

```
import styled from 'styled-components';
```

Затем создать стилизованный компонент:

```
const TodoComponent = styled.div ‘

background-color: #44014C;

width: 300px;

min-height: 200px;

margin: 30px auto;

box-sizing: border-box;

‘;
```

Затем вставить компонент для использования:

```
return (
    <TodoComponent>
      <h2>ToDo</h2>
        </TodoComponent>
)
```

76. React.js. Маршрутизация.

Маршрутизация в веб-приложении позволяет переходить от одного представления к другому на основе действия или запроса – перейти с одной страницы на другую, когда пользователь кликает на какой-либо элемент в приложении, например ссылку, кнопку, значок, изображение и т. д.

React не поставляет в комплекте стандартный маршрутизатор. Но его можно установить самостоятельно. Ключевым звеном в работе маршрутизации является модуль react-router. Для браузерных приложений предназначен модуль react-router-dom, для мобильных приложений — react-router-native.

Установка: npm install react-router-dom –save

Пример: пусть имеется такая структура проекта:
[src]

App.js

[components]

Header.js

Footer.js

Content.js

[pages]

Home.js

About.js

Contact.js

Теперь в файлу Content.js можно записать:

```
<main className="container">
    <Router>
    <Route exact path="/" component={Home} />
    <Route exact strict path="/about" component={About} />
    <Route exact strict path="/contact" component={Contact} />
    </Router>
</main>
```

Теперь домашняя страница имеет путь – слэш. Если ввести /about в адресной строке, то можно перейти на другую страницу. exact path требует точного совпадения текущего URL и шаблона path, а атрибут strict позволяет избавиться от дублей (пример - /about и /about/).

77. React.js. Хуки.

Хук — это функция javascript, которая позволяет создавать / получать доступ к состоянию и жизненным циклам React, и которая для обеспечения стабильности приложения должна использоваться в соответствии с двумя основными правилами:

- Должна вызываться на верхнем уровне приложения.
- Должен вызываться в функциях или других пользовательских обработчиках React.

Наиболее часто используемые хуки:

1) Хук состояния useState

Этот хук возвращает значение с поддерживаемым состоянием и функцию, которая необходима для его обновления:

```
const [count, setCount] = useState(0)
```

Начальное состояние — это параметр, переданный в useState, в данном случае 0, и это будет состояние до тех пор, пока функция setCount не будет вызвана с новым значением. В примере значение setCount(count + 1) будет увеличиваться, став 1 в следующем рендеринге.

2) Хук эффекта useEffect

Этот хук позволяет нам добавлять побочные эффекты к заданному функциональному компоненту.

Пример:

```
useEffect(() => {
    document.title = `Вы нажали ${count} раз`;
});
```

Другие хуки useContext, useReducer, useCallback, useMemo, useRef useImperativeHandle, useLayoutEffect, useDebugValue. React также предоставляет возможность создавать пользовательские хуки.

78. React.js. Redux.

Redux — библиотека управления состоянием для приложений, написанных на JavaScript. Она помогает писать приложения, которые ведут себя стабильно/предсказуемо, работают на разных окружениях (клиент/сервер/нативный код) и легко тестируемы. Чаще всего его используют с React. Хотя в React есть собственный метод управления состояниями, он плохо масштабируется.

Redux идеально использовать в средних и крупных приложениях. Им стоит пользоваться только в случаях, когда невозможно управлять состоянием приложения с помощью стандартного менеджера состояний в React или любой другой библиотеке. Простым приложениям Redux не нужен.

Использование Redux:

- Дерево состояний. В Redux общее состояние приложения представлено одним объектом JavaScript — state (состояние) или state tree (дерево состояний). Неизменяемое дерево состояний доступно только для чтения, изменить ничего напрямую нельзя. Изменения возможны только при отправке action (действия).

- Действие (action) — это JavaScript-объект, который лаконично описывает суть изменения. Единственное требование к объекту действия — это наличие свойства type, значением которого обычно является строка.

Пример:

```
{type: 'SELECTED_USER',
 userId: 232}
```

В простом приложении тип действия задаётся строкой. По мере разрастания функциональности приложения лучше переходить на константы:

```
const ADD_ITEM = 'ADD_ITEM'
const action = { type: ADD_ITEM, title: 'Third item' }
```

и выносить действия в отдельные файлы. А затем их импортировать:

```
import { ADD_ITEM, REMOVE_ITEM } from './actions'
```

- Генераторы действий (actions creators) — это функции, создающие действия.

```
function addItem(t) {
  return {
    type: ADD_ITEM,
    title: t}}
```

Обычно инициируются вместе с функцией отправки действия:

```
dispatch(addItem('Milk'))
```

- Редуктор (reducer) — это чистая функция, которая вычисляет следующее состояние дерева на основании его предыдущего состояния и применяемого действия. Чистая функция работает независимо от состояния программы и выдаёт выходное значение, принимая входное и не меняя ничего в нём и в остальной программе. Получается, что редуктор возвращает совершенно новый объект дерева состояний, которым заменяется предыдущий.

```
(currentState, action) => newState
```

- Хранилище (store) — это объект, который содержит состояние приложения; отображает состояние через getState(); может обновлять состояние через dispatch(); позволяет регистрироваться (или удаляться) в качестве слушателя изменения состояния через subscribe().

Хранилище в приложении всегда уникально. Так создаётся хранилище для приложения listManager:

```
import { createStore } from 'redux'
import listManager from './reducers'
let store = createStore(listManager)
```

Хранилище можно инициировать через серверные данные:

```
let store = createStore(listManager, preexistingState)
```

Функции хранилища:

1. Получение состояния: store.getState()
2. Обновление состояния: store.dispatch(addItem('Something'))
3. Прослушивание изменений состояния:

```
const unsubscribe = store.subscribe(() => const newState = store.getState())
unsubscribe()
```

- Поток данных. Всегда однонаправлен.

79. Node.js. Общая характеристика, назначение, использование, установка.

Как асинхронное событийное JavaScript-окружение, Node.js спроектирован для построения масштабируемых сетевых приложений. Ниже приведен пример "hello world", который может одновременно обрабатывать много соединений. Для каждого соединения вызывается функция обратного вызова, однако, когда соединений нет Node.js засыпает.

Этот подход контрастирует с более распространенной на сегодняшний день моделью параллелизма, в которой используются параллельные OS потоки. Такой подход является относительно неэффективным и очень сложным в использовании. Кроме того, пользователи Node.js могут не беспокоиться о блокировках процессов, поскольку их не существует. Почти ни одна из функций в Node.js не работает напрямую с I/O, поэтому поток никогда не блокируется. В следствии этого на Node.js легко разрабатывать масштабируемые системы.

Node.js создан под влиянием таких систем как Event Machine в Ruby или Twisted в Python. Но при этом событийная модель, в нем, используется значительно шире, принимая event loop за основу окружения, а не в качестве отдельной библиотеки. В других системах всегда происходят блокировки вызова, чтобы запустить цикл событий.

Обычно, поведение определяется через функции обратного вызова (callback) в начале скрипта и дальнейшим его вызовом через блокирующий вызов, вроде EventMachine::run(). В Node.js нет ничего похожего на вызов начала цикла событий, он автоматически входит в него после запуска скрипта. Node.js выходит из событийного цикла тогда, когда не остается зарегистрированных функций обратного вызова. Такое поведение похоже на поведение браузерного JavaScript, где событийный цикл скрыт от пользователя.

HTTP является объектом первого рода в Node.js, разработанным с поточностью и малой задержкой, что делает Node.js хорошей основой для веб-библиотеки или фреймворка.

То что Node.js спроектирован без многопоточности, не означает, что в нем нет возможности использовать нескольких ядер. Для работы с ними можно создавать и управлять дочерними процессами, с помощью API Child_process.fork(). Модуль cluster построен на этом интерфейсе и позволяет делиться сокетами между процессами и распределять нагрузку между ядрами.

80. Node.js. Структура проекта на node. package.json.

https://habr.com/ru/company/ruvds/blog/423703/

Свойства объекта package.json

- 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.

Все эти свойства используются либо npm либо другими инструментальными средствами, применяемыми в течение жизненного цикла приложения.

81. Node.js. Установка и импорт модулей. Зависимости.

https://habr.com/ru/company/ruvds/blog/423703/

С помощью команды npm можно загружать пакеты из реестра. Ниже мы рассмотрим примеры её использования.

Если в проекте имеется файл package.json, то установить все зависимости этого проекта можно такой командой: npm install

Эта команда загрузит всё, что нужно проекту, и поместит эти материалы в папку node_modules, создав её в том случае, если она не существует в директории проекта.

Отдельный можно установить следующей командой: npm install <package-name>

Часто можно видеть, как эту команду используют не в таком вот простом виде, а с некоторыми флагами. Рассмотрим их:

- Флаг --save позволяет установить пакет и добавить запись о нём в раздел dependencies файла package.json, который описывает зависимости проекта. Эти зависимости используются проектом для реализации его основного функционала, они устанавливаются в ходе его развёртывания на сервере (после выхода npm 5 записи об устанавливаемых пакетах в разделе зависимостей делаются автоматически, и без использования этого флага).
- Флаг --save-dev позволяет установить пакет и добавить запись о нём в раздел, содержащий перечень зависимостей разработки (то есть — пакетов, которые нужны в ходе разработки проекта, вроде библиотек для тестирования, но не требуются для его работы) файла package.json, который называется devDependencies.

Для обновления пакетов служит следующая команда: npm update

Получив эту команду, npm проверит все пакеты на наличие их новых версий, и, если найдёт их новые версии, соответствующие ограничениям на версии пакетов, заданным в package.json, установит их.

Обновить можно и отдельный пакет: npm update <package-name>

82. Node.js. Семантическое версионирование.

https://habr.com/ru/company/ruvds/blog/423703/

В дополнение к стандартной загрузке пакетов, npm поддерживает и загрузку их определённых версий. В частности, можно заметить, что некоторые библиотеки совместимы лишь с некими крупными релизами других библиотек, то есть, если бы зависимости таких библиотек устанавливались бы без учёта версий, это могло бы нарушить их работу. Возможность установить определённую версию некоего пакета полезна и в ситуациях, когда, например, вам вполне подходит самый свежий релиз этого пакета, но оказывается, что в нём имеется ошибка. Ожидая выхода исправленной версии пакета, можно воспользоваться и его более старым но стабильным релизом.

Возможность задавать конкретные версии необходимых проекту библиотек полезна в командной разработке, когда все члены команды пользуются в точности одними и теми же библиотеками. Переход на их новые версии так же осуществляется централизованно, путём внесения изменений в файл проекта package.json.

- ~: если вы задаёте версию в виде ~0.13.0 это означает, что вас интересуют лишь патч-релизы пакета. То есть, пакет 0.13.1 вам подойдёт, а 0.14.0 — нет.
- ^: если номер версии задан в виде ^0.13.0, это означает, что вам подходят новые патч-версии и минорные версии пакета. То есть, вас устроят версии пакета 0.13.1, 0.14.0, и так далее.
- *: воспользовавшись этим символом, вы сообщаете системе, что вас устроят любые свежие версии пакета, в том числе — его новые мажорные релизы.
- >: подходят любые версии пакета, которые больше заданной.
- >=: подходят любые версии пакета, которые равны или больше заданной.
- <=: вас устроят пакеты, версии которых равны заданной или меньше её.
- <: вас интересуют пакеты, версии которых меньше заданной.
- =: вам нужна только заданная версия пакета.
- -: используется для указания диапазона подходящих версий, например — 2.1.0 - 2.6.2.
- ||: позволяет комбинировать наборы условий, касающихся пакетов. Например это может выглядеть как < 2.1 || > 2.6.

83. Технологии разработки серверных веб-приложений.

https://habr.com/ru/post/450282/

1. Чем веб-приложения отличаются от сайтов

Для меня сайт это в первую очередь что-то информационное и статичное: визитка компании, сайт рецептов, городской портал или вики. Набор подготовленных заранее HTML-файлов, которые лежат на удаленном сервере и отдаются браузеру по запросу.

Сайты содержат различную статику, которая как и HTML-файл не генерируется на лету. Чаще всего это картинки, CSS-файлы, JS-скрипты, но могут быть и любые другие файлы: mp3, mov, csv, pdf.

Блоги, визитки с формой для контакта, лендинги с кучей эффектов я тоже отношу для простоты к сайтам. Хотя в отличие от совсем статических сайтов, они уже включают в себя какую-то бизнес-логику.

А веб-приложение — это что-то технически более сложное. Тут HTML-страницы генерируются на лету в зависимости от запроса пользователя. Почтовые клиенты, соцсети, поисковики, интернет-магазины, онлайн-программы для бизнеса, это все веб-приложения.

2. Какие бывают веб-приложения

Веб-приложения можно разделить на несколько типов, в зависимости от разных сочетаний его основных составляющих:

- Backend (бэкенд или серверная часть приложения) работает на удаленном компьютере, который может находиться где угодно. Она может быть написана на разных языках программирования: PHP, Python, Ruby, C# и других. Если создавать приложение используя только серверную часть, то в результате любых переходов между разделами, отправок форм, обновления данных, сервером будет генерироваться новый HTML-файл и страница в браузере будет перезагружаться.
- Frontend (фронтенд или клиентская часть приложения) выполняется в браузере пользователя. Эта часть написана на языке программирования Javascript. Приложение может состоять только из клиентской части, если не требуется хранить данные пользователя дольше одной сессии. Это могут быть, например, фоторедакторы или простые игрушки.
- Single page application (SPA или одностраничное приложение). Более интересный вариант, когда используются и бэкенд и фронтенд. С помощью их взаимодействия можно создать приложение, которое будет работать совсем без перезагрузок страницы в браузере. Или в упрощенном варианте, когда переходы между разделами вызывают перезагрузки, но любые действия в разделе обходятся без них.

84. Серверные фреймворки. Общая характеристика, назначение, примеры, сравнение.

https://developer.mozilla.org/ru/docs/Learn/Server-side/First_steps/Web_frameworks

Веб-фреймворки предоставляют инструменты и библиотеки для упрощения общих операций веб-разработки. Вы не обязаны использовать веб-фреймворк на стороне сервера, но это настоятельно рекомендуется — это сделает вашу жизнь намного проще.

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, включая маршрутизацию URL-адресов для соответствующих обработчиков, взаимодействие с базами данных, поддержку сеансов и авторизацию пользователей, форматирование вывода (например, HTML, JSON, XML) и улучшение защиты от веб-атак.

- Django (Python)
  Django — это веб-фреймворк высокого уровня на языке Python, который способствует быстрой разработке и чистому, прагматичному дизайну. Он бесплатен для использования и имеет открытый исходный код.
  Популярные сайты, использующие Django (с домашней страницы Django), включают в себя: Disqus, Instagram, Knight Foundation, MacArthur Foundation, Mozilla, National Geographic, Open Knowledge Foundation, Pinterest, Open Stack.
- Flask (Python)
  Flask — это микрофреймворк для Python.
  И хотя Flask минималистичен, он может создавать серьёзные веб-сайты из коробки. Он содержит сервер разработки и отладчик, а также поддерживает шаблоны Jinja2, безопасные файлы cookie, модульное тестирование и диспетчеризацию запросов RESTful. У него хорошая документация и активное сообщество.
- Express (Node.js/JavaScript)
  Express — быстрый, непринуждённый, гибкий и минималистский веб-фреймворк для Node.js (node — это серверная среда для запуска JavaScript). Он обеспечивает надёжный набор функций для веб и мобильных приложений и предоставляет полезные HTTP-утилиты и middleware (промежуточные интерфейсы).
  Express используют многие крупные компании, в том числе: Uber, Accenture, IBM и т. д. (список приведён здесь).
- Ruby on Rails (Ruby)
  Rails (обычно именуется «Ruby on Rails») — это веб-фреймворк, написанный для языка программирования Ruby.
  Twitch, SoundCloud, Hulu, Zendesk, Square, Highrise.
- ASP.NET
  ASP.NET — это веб-фреймворк с открытым исходным кодом, разработанный Microsoft для создания современных веб-приложений и сервисов.
  ASP.NET используется Microsoft, Xbox.com, Stack Overflow и многими другими.

85. Express.js. Основные понятия. Установка, запуск, простейшее приложение.

https://expressjs.com/ru/starter/installing.html

https://expressjs.com/ru/starter/hello-world.html

Установка

Предположим, вы уже установили Node.js. Создайте каталог для своего приложения и сделайте его своим рабочим каталогом.

```
$ mkdir myapp
$ cd myapp
```

С помощью команды npm init создайте файл package.json для своего приложения. Дополнительную информацию о работе package.json можно найти в разделе Специфика работы с npm package.json.

```
$ npm init
```

Эта команда выдает целый ряд приглашений, например, приглашение указать имя и версию вашего приложения. На данный момент, достаточно просто нажать клавишу ВВОД, чтобы принять предлагаемые значения по умолчанию для большинства пунктов, кроме следующего:

```
entry point: (index.js)
```

Введите app.js или любое другое имя главного файла по своему желанию. Если вас устраивает index.js, нажмите клавишу ВВОД, чтобы принять предложенное имя файла по умолчанию.

Теперь установите Express в каталоге myapp и сохраните его в списке зависимостей. Например:

```
$ npm install express --save
```

Для временной установки Express, без добавления его в список зависимостей, не указывайте опцию --save:

```
$ npm install express
```

В каталоге myapp создайте файл с именем app.js и добавьте следующий код:

```
const express = require('express')
const app = express()
const port = 3000
app.get('/', (req, res) => {
  res.send('Hello World!')
```

```
})
app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

86. Express.js. Статический сервер.

https://expressjs.com/ru/starter/static-files.html

Для предоставления статических файлов, например, изображений, файлов CSS и JavaScript в Express используется функция промежуточной обработки express.static.

Для того чтобы начать непосредственное предоставление файлов, необходимо передать имя каталога, в котором находятся статические ресурсы, в функцию промежуточной обработки express.static. Например, воспользуйтесь приведенным ниже кодом для предоставления изображений, файлов CSS и JavaScript, расположенных в каталоге public:

app.use(express.static('public'));

Express выполняет поиск файлов относительно статического каталога, поэтому имя статического каталога не является частью URL.

Для использования нескольких каталогов, содержащих статические ресурсы, необходимо вызвать функцию промежуточной обработки express.static несколько раз:

app.use(express.static('public'));

app.use(express.static('files'));

Express выполняет поиск файлов в том порядке, в котором указаны статические каталоги в функции промежуточной обработки express.static.

Для того чтобы создать префикс виртуального пути (то есть, пути, фактически не существующего в файловой системе) для файлов, предоставляемых с помощью функции express.static, необходимо указать путь монтирования для статического каталога, как показано ниже:

app.use('/static', express.static('public'));

Теперь можно загрузить файлы, находящиеся в каталоге public, указанного в префиксе пути /static.

Тем не менее, путь, переданный в функцию express.static, указан относительно каталога, из которого запускается процесс node. В случае запуска приложения Express из другого каталога, безопаснее использовать абсолютный путь к каталогу для предоставления файлов:

app.use('/static', express.static(__dirname + '/public'));

87. Express.js. Возврат JSON.

https://www.geeksforgeeks.org/express-js-res-json-function/

https://expressjs.com/en/5x/api.html#res.json

res.json([body])

Отправляет ответ в формате JSON. Этот метод отправляет ответ (с правильным типом содержимого), который представляет собой параметр, преобразованный в строку JSON с помощью JSON.stringify().

Параметр может быть любого типа JSON, включая object, array, string, Boolean, number или null, и вы также можете использовать его для преобразования других значений в JSON.

```
res.json(null)
res.json({ user: 'tobi' })
res.status(500).json({ error: 'message' })
```

88. Express.js. Middleware.

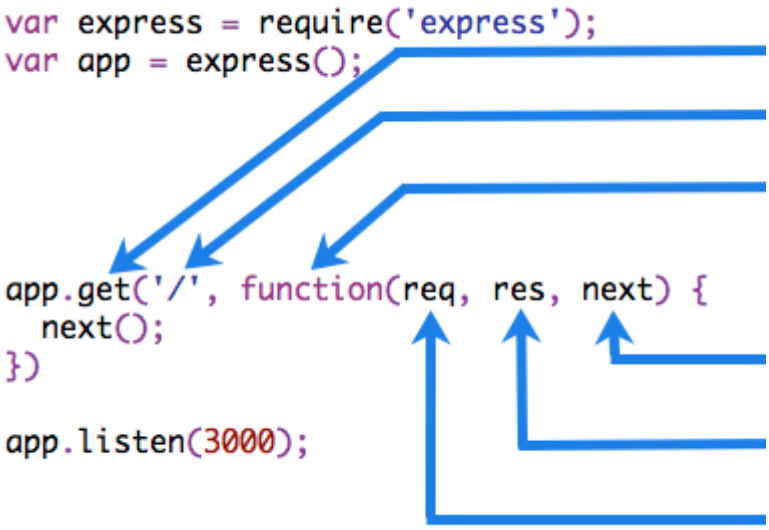
https://expressjs.com/ru/guide/writing-middleware.html

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, как правило, 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 следующего промежуточного обработчика 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.



Ниже представлены элементы вызова функции промежуточного обработчика:

- Метод HTTP, к которому применяется данный промежуточный обработчик.
- Путь (маршрут), к которому применяется данный промежуточный обработчик.
- Функция промежуточного обработчика.
- Аргумент обратного вызова для функции промежуточного обработчика, именуемый "next" согласно стандарту.
- Аргумент ответа HTTP, именуемый "res" согласно стандарту.
- Аргумент запроса HTTP, именуемый "req" согласно стандарту.

89. REST API. Основные понятия, назначение, использование.

Representational State Transfer — это архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Архитектурный стиль – это набор согласованных ограничений и принципов проектирования, позволяющий добиться определённых свойств системы.

6 принципов REST:

1. Клиент-серверная архитектура
2. Stateless
3. Кэширование
4. Единообразие интерфейса
5. Layered system
6. Code on demand

Методы HTTP-запроса

Метод, используемый в HTTP-запросе, указывает, какое действие вы хотите выполнить с этим запросом. Важные примеры:

- GET: получить подробную информацию о ресурсе
- POST: создать новый ресурс
- PUT: обновить существующий ресурс
- DELETE: Удалить ресурс

Код статуса ответа HTTP

Код состояния всегда присутствует в ответе HTTP. Типичные примеры:

- 200 — успех
- 404 — страница не найдена

Когда вы разрабатываете RESTful сервисы, вы должны сосредоточить свое внимание на ресурсах приложения. Способ, которым мы идентифицируем ресурс для предоставления, состоит в том, чтобы назначить ему URI — универсальный идентификатор ресурса. Например:

- Создать пользователя: POST /users
- Удалить пользователя: DELETE /users/1
- Получить всех пользователей: GET /users
- Получить одного пользователя: GET /users/1

90. Express.js. Создание RESTfull API.

Express - фреймворк web-приложений для Node.js. Express может являться backend'ом для программного стека MEAN, вместе с базой данных MongoDB и каркасом Vue.js, React или AngularJS для frontend'a.

С помощью данного фреймворка возможна реализация ресурсов с обработчиками запросов RESTful API - GET, POST, PUT, DELETE.

Пример ресурса:

```
var express = require('express');
var app = express();
var itemRoute = express.Router();
itemRoute.param('itemId', function(req, res, next, id) {...});
itemRoute.route('/:itemId')
  .get(function(req, res, next) {...})
  .put(function(req, res, next) {...})
  .post(function(req, res, next) {...})
  .delete(function(req, res, next) {...})
app.use('/api/items', itemRoute);
app.listen(8080);
```

В данном примере мы описали все методы для маршрута /api/items, в котором также принимается параметр itemId.

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.

Плюсы	Минусы
1. Низкий порог вхождения, Express — это практически стандарт Node.js-приложения. 2. Полная кастомизация.	1. Все ресурсы необходимо создавать вручную и, в результате, появится много повторного кода или хуже — собственная библиотека. 2. Каждый ресурс требует тестирования или простой проверки на 500-ую ошибку. 3. Рефакторинг станет болезненным, так как будет необходимо править всё и везде. 4. Нету стандартного подхода, нужно искать свой.

91. Express.js. Использование роутера.

Router позволяет определить дочерние подмаршруты со своими обработчиками относительно некоторого главного маршрута.

Например пусть заданы следующие обработчики:

```
app.use("/products/create",function (request, response) {...});
app.use("/products/:id",function (request, response) {...});
app.use("/products/",function (request, response) {...});
app.listen(3000);
```

Три маршрута начинаются с "/products" и условно относятся к некоторому функционалу по работе с товарами (просмотр списка товаров, просмотр одного товара по id и добавление товара).

Объект Router позволяет связать подобный функционал в одно целое и упростить управление им.

Пример с использованием объекта Router:

```
// определяем Router
const productRouter = express.Router();
// определяем маршруты и их обработчики внутри роутера
productRouter.use("/create", function(request, response){...});
productRouter.use("/:id", function(request, response){...});
productRouter.use("/", function(request, response){...});
// сопоставляем роутер с конечной точкой "/products"
app.use("/products", productRouter);
app.listen(3000);
```

92. Express.js. Шаблонизатор handlebars.

В Node.js для генерации и отдачи HTML-страниц используются шаблонизаторы. Node.js шаблонизатор представляет собой специальный модуль, использующий более удобный синтаксис для формирования HTML на основе динамических данных и позволяющий разделять представление от контроллера.

Настройка Node.js шаблонизатора осуществляется заданием двух параметров:

- views - путь к директории, в которой находятся шаблоны;
- view engine - указание самого шаблонизатора.

Для задания этих параметров используется метод Express set().

```
app.set('views', './views')
app.set('view engine', 'handlebars')
```

Шаблонизаторов очень много, но наибольшее распространение получили Handlebars и Pug.

Установка handlebars:

```
npm install --save express-handlebars
```

С помощью метода engine() задается настройка Node.js handlebars, конкретно в примере указывается шаблон по умолчанию, в который будут подгружаться шаблоны страниц. Генерация и отдача представления осуществляется с помощью метода render(), который принимает два параметра:

- шаблон;
- данные для шаблона в виде объекта (если необходимо).

Шаблоны Node.js handlebars представляют собой обычные файлы HTML в формате handlebars, в которых с помощью специального синтаксиса выводятся передаваемые данные. Для отображения значения свойства переданного объекта используется запись {{{(название свойства)}}}.

В макете /views/layouts/main.handlebars запись {{{body}}} определяет место, куда при запросе определенной страницы будет вставлено соответствующее ей представление.

93. Express.js. Работа с базой данных.

Для того чтобы добавить функциональную возможность подключения базы данных к приложению Express, необходимо всего лишь загрузить драйвер Node.js для соответствующей базы данных. Для Express реализованы драйверы наиболее популярных СУБД:

- MySQL
- MongoDB
- PostgreSQL
- Redis
- SQLite

Пример установки:

$ npm install mongodb

У каждого драйвера реализован свой API для взаимодействия с БД. Рассмотрим подключение и простой запрос на примере MongoDB:

Пример

```
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017/animals', function(err, db) {
  if (err) {
    throw err;
  }
  db.collection('mammals').find().toArray(function(err, result) {
    if (err) {
      throw err;
    }
    console.log(result);
  });
});
```

Пример с PostgreSQL:

PostgreSQL

Модуль: pg-promise - **Установка**

```
$ npm install pg-promise
```

Пример

```
var pgp = require("pg-promise")(/*options*/);
var db = pgp("postgres://username:password@host:port/database");

db.one("SELECT $1 AS value", 123)
    .then(function (data) {
        console.log("DATA:", data.value);
    })
    .catch(function (error) {
        console.log("ERROR:", error);
    });
```

Можно заметить, что API действительно разный, однако логика схожа - сначала выполняется запрос, характерный для конкретной СУБД, который передается в виде строки, после этого определяется функция, в которой в переменной-параметре функции содержатся данные полученные из запроса, и с этими данными можно работать уже из кода js.

94. MongoDB. Общая характеристика, сравнение.

Определение

MongoDB это кросс-платформенная, документо-ориентированная база данных, которая обеспечаивает высокую производительность и лёгкую масштабируемость. В основе данной БД лежит концепция коллекций и документов.

База данных

База данных представленна в виде физического хранилища коллекций. Каждая БД имеет свой собственный набор файлов в файловой системе. Обычно, один MongoDB сервер имеет несколько БД.

Коллекция

Коллекция – это группа документов MongoDB. Является эквивалентом простой таблицы в реляционной базе данных. Коллекция помещена внутри одной БД. Документ в коллекции моет иметь различные поля. Чаще всего, все документы в коллекции созданы для одной, либо относящихся друг ко другу целей.

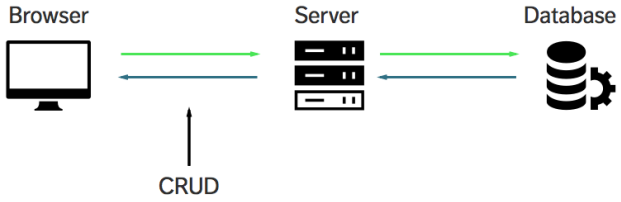
Документ

Документ – это набор пар “ключ – значение”. Документ имеет динамическую схему. Это означает, что документ в одной и той же коллекции не обязан иметь один одинаковый набор полей или структуру, а общие поля в коллекции могут иметь различные типы данных.

Ниже приведена небольшая сравнительная таблица реляционных БД и MongoDB:

Реляционная БД	MongoDB
База данных	База данных
Таблица	Коллекция
Ряд	Документ
Колонка	Поле
Объединенние таблиц	Встроенные документы (embedded)
Первичный ключ (primary key)	Первичный ключ (primary key). По умолчанию MongoDB генерирует Default key_id

95. Express.js. CRUD операции с MongoDB.



Шаблон по написанию обработчиков GET, POST, PUT, DELETE с использованием MongoDB.

```
MongoClient.connect(/*                                    ...                                    */)
  .then(client                                            =>                                     {
    //                                                                                           …
    const                db                             =                client.db('star-wars-quotes')
    const          someColletcion                       =          db.collection(‘some-collection’);
    app.use(/*                                            ...                                    */)
    app.get(/*                                            ...                                    */)
    app.post(/*                                           ...                                    */)
    app.listen(/*                                         ...                                    */)
  })
 .catch(console.error)
```

Закомментированный код в обработчиках имеет доступ к переменным, в которых мы определяем коллекции. Мы можем описать метод POST для создания какой-нибудь записи.

POST - CREATE (создание записи в коллекции)

```
app.post('/quotes',                (req,                res)                =>                {
  someCollection.insertOne(req.body)
    .then(result                                          =>                                     {
      console.log(result)
    })
    .catch(error                                  =>                          console.error(error))
})
```

Мы также можем описать другие методы, которые будут иметь доступ к данной коллекции:

GET - READ (поиск записей в БД)

```
app.get('/',                (req,                res)                =>                {
  const                cursor                =                db.collection('quotes').find()
  console.log(cursor)
  //                                                                                           …
});
```

PUT - UPDATE (изменение записей в БД)

```
app.put('/quotes',                (req,                res)                =>                {
```

```
        someCollection.findOneAndUpdate(
  query,
  update,
  options
)
  .then(result => {/* ... */})
  .catch(error => console.error(error));
  /* ... */
});
```

DELETE - DELETE (удаление записи в БД)

```
app.delete('/quotes', (req, res) => {
  // Handle delete event here
someCollection.deleteOne(
  query,
  options
)
  .then(result => {/* ... */})
  .catch(error => console.error(error));
});
```

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

Понятие

ORM (Object-Relational Mapping, объектно-реляционное отображение) — технология программирования, суть которой заключается в создании «виртуальной объектной базы данных».

Назначение и ипользование

С помощью этой технологии разработчики могут использовать язык программирования, с которым им удобно работать с базой данных, вместо написания запросов SQL. Это может ускорить разработку.

ORM также позволяет мигрировать между различными БД практически без доработки кода: например, MySQL -> MariaDB, MySQL -> PostgreSQL.

Обычно в ORM встроены механизмы для предостващения SQL-инъекций (экранирование символа ' и т. д.)

ORM используют именно с реляционыыми БД на стороне бекенда т.к. бекенд-программисту удобно получать с запроса сразу готовые объекты-классы сущностей, а не писать вручную конвертеры с результатов SQL-запросов

Примеры использования

Пример с использованием ORM Sequelize

1. Ставим зависимости

```
npm install sequelize sqlite3
# или
yarn add sequelize sqlite3
```

2. Объявляем модели

```
import { Sequelize, Model, DataTypes } from 'sequelize';
const sequelize = new Sequelize('sqlite::memory:');
const User = sequelize.define('User', {
  username: DataTypes.STRING,
  birthday: DataTypes.DATE,
});
```

3. Сохранение данных и запрос

```
const jane = await User.create({
  username: 'janedoe',
  birthday: new Date(1980, 6, 20),
```

```
});
const users = await User.findAll();
```

97. Библиотека Sequelize. Общая характеристика, назначение, установка, подключение.

Sequelize - это ORM (объектно-реляционное отображение или преобразование) для работы с такими СУБД , 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:

```
npm install sequelize sqlite3
# или
yarn add sequelize sqlite3
```

Подключение к БД:

Для подключения к базе данных прежде всего необходимо создать объект Sequelize.

Для создания объекта sequelize используется функция Sequelize, которая принимает ряд параметров. Первый параметр - имя базы данных. Второй параметр - логин к бд, третий параметр - пароль. Это обязательные параметры.

Кроме того, с помощью четвертого параметра мы можем задать ряд дополнительных опций конфигурации. Четвертый параметр представляет объект, который имеет множество свойств. В данном случае используются только два. Первое свойство - dialect указывает на диалект языка SQL, который используется в запросах - в данном случае "mysql". Второе свойство host представляет адрес, по которому запущен сервер. По умолчанию host имеет значение "localhost", поэтому для подключения к локальной базе данных это свойство в принципе можно не указывать.

По умолчанию после того, как установки соединения, оно остается открытым. Для его закрытия следует вызвать метод sequelize.close().

98. Библиотека 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 в Sequelize — это абстракция, которая представляет собой таблицу базы данных. Она передает системе название, количество столбцов и тип содержащейся в них информации. Каждая модель имеет собственное имя. Чаще всего оно совпадает с названием таблицы. Например, если таблица называется Accounts («учетные записи», множественное число), то имя модели будет Account («учетная запись», единственное число).

Для определения моделей в Sequelize существует два способа:

1. Вызов sequelize.define(modelName, attributes, options) с передачей функции имени модели, атрибутов и опций.
2. Расширение класса Model и вызов init(attributes, options) с передачей атрибутов и опций.

Рассмотрим создание модели на примере. Предположим, что у нас есть таблица Accounts, и на ее основе мы хотим создать модель Account, в которой будут содержаться данные учетных записей пользователей — userLogin и userPassword. Для обращения к модели будет использоваться метод sequelize.models.Account:

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:');
const Account = sequelize.define('Account', {
// определение атрибутов модели
userLogin: {
type: DataTypes.STRING,
allowNull: false
},
userPassword: {
type: DataTypes.STRING
// значение allowNull равно true по умолчанию
}
}, {
// другие опции модели
});
// `sequelize.define` возвращает созданную модель
console.log(Account === sequelize.models.Account); // true
```

99. Библиотека Sequelize. Выполнение запросов к базе данных.

Подключение к БД

Соединение Sequelize должно передавать параметры и может быть настроено для открытия пула потоков, чтения и записи библиотеки и других операций. Простой способ написания такой: new Sequelize («имя таблицы», «имя пользователя», «пароль», конфигурация)

Все параметры редко используются при обычном использовании. Вот часто используемый шаблон. Вам нужно только изменить используемые значения.

```
const sequelize = new Sequelize('database', 'username', 'password', {
 host: 'localhost', // Адрес базы данных, по умолчанию этот компьютер
 port:'3306',
 dialect: 'mysql',
 pool: { // Настройки пула подключений
  max: 5, // Максимальное количество подключений
  min: 0, // Минимальное количество подключений
  idle: 10000
 },
});
```

Выполнение запросов к БД.

Запрос findAll - ищет по всем объектам модели, работает аналогично **SELECT * FROM ….**

```
let list = await model.findAll({
    where:{
        id:{$gt:10},// id больше 10
        name:"test" // имя равно test
    },
    order:[
        "id", // Сортировать по id
        ["id","desc"]// Обратный порядок по id
    ],
    limit:10,// Возвращаем число
    offset:20,// Начальная позиция, пропускаем количество
    attributes:["attr1","attr2"], // Возвращаемое поле
});
```

Запрос findById (id, opts) - первичным ключом данных по умолчанию здесь является id, и данные напрямую запрашиваются через id при запросе. При создании новой базы данных рекомендуется добавить идентификатор в качестве уникального первичного ключа.

Работает аналогично **SELECT * FROM … WHERE id=12**

```
let model = await model.findById(12);
//select a,b,c from model where id=12;
```

Запрос CREATE - создает запись в БД по созданному объекту. Запись, аналогичная **INSERT INTO .. (name, token) VALUES (“test”, “adwadfv2324”)**

```
let model= {
        name:"test",
        token:"adwadfv2324"
}
 await model.create(model);
//insert into model (name,token) values("test","adwadfv2324");
```

Запрос DESTROY - удаляет запись в БД по определенному паттерну. Запись, аналогичная **DELETE FROM … WHERE name LIKE “ЗАВОД%”;**

```
User.destroy(`name` LIKE "ЗАВОД%"').success(
function()
   {
     // Мы только что удалили все записи, в которых name начинался на “ЗАВОД”
   }
)
```

100. Библиотека Sequelize. Геттеры, сеттеры и виртуальные атрибуты.

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:

Пример:

```
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.

Сеттер — это функция set(), определенная для колонки. Она принимает значение для установки:

Пример:

```
const User = sequelize.define('user', {
  username: DataTypes.STRING,
  password: {
    type: DataTypes.STRING,
    set(value) {
      // Перед записью в БД пароли следует "хэшировать" с помощью криптографической функции
      this.setDataValue('password', hash(value))
    },
  },
})
```

Сеттер вызывается автоматически при создании экземпляра. В сеттере можно использовать значения других полей

Геттеры и сеттеры можно использовать совместно. Допустим, что у нас имеется модель Post с полем content неограниченной длины, и в целях экономии памяти мы решили хранить в БД содержимое поста в сжатом виде.

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

Пример:

```
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!')
    },
  },
})
```

Примерные практические задания к экзамену

1. Даны картинки. По нажатию на любую картинку увеличьте ее в 2 раза.

2. Даны N инпутов с классом .num и кнопка. По нажатию на кнопку получите числа, стоящие в этих инпутах и запишите их сумму в абзац с id=”result”.

3. Дана таблица с числами. По нажатию на кнопку найдите ячейку, в которой хранится максимальное число, и сделайте ее фон красным.

4. Дана таблица с числами. По нажатию на ячейку она активируется и становится красного цвета. Активировать можно много ячеек. Под таблицей кнопка. По нажатию по этой кнопке в абзац ниже выведите сумму активированных ячеек. Реализуйте кнопку ‘сбросить активированные ячейки’.

5. Реализуйте раскрывающийся список. По умолчанию есть список стран (ul), по нажатию на страну внутри li со страной появляется список городов.

6. В инпут через запятую вводятся страны. По нажатию на кнопку сделайте так, чтобы эти страны записались в ul под инпутом (каждая страна отдельный li).

7. Дан ряд ссылок. Сделайте так, чтобы по нажатию на ссылку она становилась с красным фоном. По нажатию на другую ссылку выделение первой ссылки снимается и выделяется та, на которую мы нажали. В абзац ниже пишите текст активной ссылки.