***Лекция.***

***Введение***

*JavaScript — это кросс-платформенный, объектно- ориентированный скриптовый язык, который позволяет веб-разработчику управлять тем, как ведет себя веб- страница. Это делает JavaScript принципиально отличным от HTML, который отвечает за структуру веб-документа, и СЅЅ, который формирует внешний вид веб-страниц.*

*Программы, написанные на языке JavaScript, называются скриптами или сценариями. Скрипты JavaScript могут быть вставлены в любое место HTML- документа, и как только браузер отображая код страницы доходит до скрипта - выполняются. Процесс выполнения скрипта называют интерпретацией.*

***Важно!***  *Если РНР скрипт обрабатывается на стороне сервера с помощью РНР интерпретатора, то JavaScript выполняется в браузере пользователя JavaScript интерпретатором.*

*Сегодня каждый браузер поддерживает JavaScript, тем самым делая его языком веба-разработки. Вот некоторые из задач, которые может решать программист с помощью JavaScript:*

*• Проверять правильность заполнения пользовательских HTML-форм;*

*менять стили HTML-элементов, скрывать, показывать элементы и т.п.; отображать всплывающие и диалоговые окна;*

***Тема 1. Введение в JavaScript 2 JS***

*• реагировать на действия посетителя, обрабатывать клики мыши, перемещения курсора и т.п.;*

*посылать запросы на сервер и загружать данные без перезагрузки страницы.*

***Подключение сценариев.***

*Чтобы программа (сценарий) JavaScript запустилась, ее нужно внедрить в НТМL-документ. Сценарии можно внедрить в HTML-документ различными стандартными способами:*

* *Поместить код между открывающим и закрывающим тегами script;*
* *Поместить скрипты во внешний файл (с расширением .js), а затем подключить его к документу HTML;*
* *Поместить код непосредственно в атрибут события HTML-элемента.*

***JavaScript в элементе script***

*Самый простой способ внедрения JavaScript в HTML-документ - использование тега script. Теги script могут размещаться в любом месте документа.*

*Таким образом, на одной веб-странице могут располагаться сразу несколько сценариев. Выполнение сценариев браузерами происходит по мере их загрузки. Браузер читает HTML-документ сверху вниз и, когда он встречает тег script, рассматривает текст программы как сценарий и выполняет его.*

***Важно!***  *Оставшийся HTML-контент страницы не загружается и не отображается, пока не будет выполнен весь код в элементе script.*

*<!-- ... -->*

*<header>*

*<h1 align = "center">Введение в JavaScript</h1>*

*</header>*

*<!--инструкции HTML-кода...-->*

*<script>alert("Добро пожаловать в JavaScript!");</script>*

*<!--инструкции HTML-кода...-->*

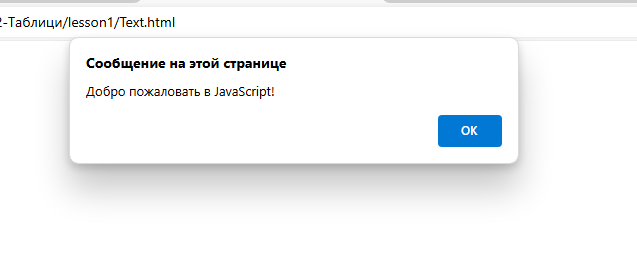
*<footer align = "center">*

*<h3>Добро пожаловать на pechora\_PRO</h3>*

*<a href="https://vk.com/pechora\_proger" target="\_blank">pechora\_PRO</a>*

*</footer>*

*<!-- ... -->*

**

*Однако считается плохим стилем программирования "разбрасывать" элементы script по всему документу. Гораздо правильнее приучить себя создавать сценарии либо в начале, либо в конце документа. Таким образом, так же, как отделяют СЅЅ от HTML, стараются отделить сценарии JavaScript от собственно верстки НТМL-документа.*

*example\_2.*

*Вынесение кода в тег HEAD*

*<!-- ...*

*<head>*

*<meta http-equiv="Content-Type" content="text/html;*

*charset=utf-8">*

*<!-- ... -->*

*<head>*

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

*<title>Курс JavaScript</title>*

*<script>*

*alert("Добро пожаловать в JavaScript!");*

*</script>*

*</head>*

*<!-- ... -->*

***Внешний JavaScript***

*Как правило, только простейшие скрипты помещаются в HTML документ. Если JavaScript-кода много - его выносят в отдельный файл, который, как правило, имеет расширение .js.*

*Чтобы включить в HTML-документ JavaScript-код из внешнего файла, нужно использовать атрибут src (source) тега script. Значением атрибута должен быть адрес файла, в котором содержится Ѕ-код.*

*\*

***Например****:*

*<script src= "/script/script.js"></script>*

***Для подключения нескольких скриптов используйте несколько тегов script:***

*<script src = "/script/banner.js"></script>*

*<script src = "/script/page.js"></script>*

*Одновременно подключить внешний файл и написать что-то внутри тега script нельзя. Придется создавать два разных тега: первый, с атрибутом src, второй - с командами, которые будут выполнены после выполнения внешнего файла.*

*<!-- так работать не будет -->*

*<script src = "/script/banner.js">*

*alert ("Добро пожаловать в JavaScript!");*

*</script>*

*<!-- а так будет -->*

*<!-- сначала выполнится внешний скрипт -->*

*<script src= "/script/banner.js"></script> <!-- потом встроенный -->*

*<script>*

*alert ("Добро пожаловать в JavaScript!");*

*</script>*

*Внешние скрипты практичны, когда один и тот же JS-файл используется более чем в одной веб-странице проекта. Браузер скачает файл один раз, и в дальнейшем будет брать его из своего кеша, благодаря чему скрипт, содержащий, к примеру, библиотеку функций, может использоваться на разных страницах без полной перезагрузки с сервера.*

***Важно!*** *У такой возможности браузера есть и оборотная сторона. Когда правите скрипты, не забывайте время от времени чистить кеш браузера (Ctrl+Shift+Del), либо выполняйте обновление страницы без учета.*

*Кроме этого, благодаря внешним скриптам, упрощается сопровождение кода, поскольку вносить изменения или исправлять ошибки приходится только в одном месте.*

***JavaScript в атрибутах событий HTML-элементов***

*Не всегда нужно, чтобы выполнение сценария начиналось сразу при загрузке страницы. Чаще всего требуется, чтобы программа запускалась в ответ на* ***определенные действия пользователя.***

***Каждый НТМL****-элемент имеет JS-события, которые срабатывают вопределенный момент.*

*Нужно добавить необходимое событие в HTML-элемент как атрибут, а в качестве значения этого атрибута указать исполняемый код (при дальнейшем изучении исполняемую функцию). Код, исполняемый в ответ на срабатывание события, называется* ***обработчиком события.***

*В следующем примере HTML-элемент* ***button*** *с атрибутом JavaScript события* ***onclick*** *обеспечивает реакцию скрипта на клик мыши. При нажатии кнопки генерируется событие* ***onclick*** *и выполняется связанный с событием ЈЅ-код.*

***<body>***

***<header>***

***<h1 align = "center">Введение в JavaScript</h1>***

***</header>***

***<button onclick="alert('Добро пожаловать в JavaScript!')">Нажми меня</button>***

***<footer align = "center">***

***<h3>Добро пожаловать </h3>***

***<a href="https://clck.ru/3D2GVT" target="\_blank">PRO</a>***

***</footer>***

***</body>***

***<!-- ... -->***

***JavaScript как обработчик события.***

***Важно!*** *Независимо от того, как JS-код включается в HTML-документ,элементы script интерпретируются браузером в том порядке, в котором они* ***расположены в HTML-документе****. Сначала интерпретируется код первого элемента script, затем браузер приступает ко второму элементу script и т. д.*

***Важно!***  *События тема отдельного урока. Здесь это понятие приведено для общего представления возможностей подключения сценариев.*

***Расположение тегов script***

*Как уже было сказано, браузер читает HTML-документ сверху вниз и, начинает отображать страницу, показывая часть документа до тега script. Встретив тег script, переключается в JavaScript-режим и выполняет сценарий. Закончив выполнение, возвращается обратно в HTML-режим и отображает оставшуюся*

*часть документа.*

*Если на странице используется много скриптов JavaScript, то могут возникнуть Длительные задержки при загрузке, в течение которых пользователь видит пустое окно браузера. Одним из способов решения проблемы считается указание 1Ѕ-сценариев после контента страницы, перед закрывающим* ***тегом body:***

*<!--<script src = "/script/banner.js"></script>*

*<script src="/script/page.js"></script>*

*Такое расположение сценариев позволяет браузеру загружать страницу быстрее, так как сначала загрузится контент страницы, а потом будет загружаться код сценария.*

*Для пользователей это предпочтительнее, потому что страница полностью визуализируется в браузере до обработки JavaScript-кода.*

***Синхронные, асинхронные и отложенные сценарии***

*Расположение JS-сценариев перед закрывающим тегом body в некотором мысле ускоряет загрузку и отображение HTML-документа. Но есть более современные способы управлять порядком выполнения сценариев.*

***Синхронное выполнение сценария***

*В современных сайтах скрипты обычно тяжелее, чем код HTML: они весят больше, дольше обрабатываются.*

*По умолчанию файлы JS-кода прерывают синтаксический анализ (парсинг) НТМL-документа до тех пор, пока скрипт не будет загружен (если он внешний) и выполнен, тем самым увеличивая промежуток времени до первой отрисовки страницы.*

***Важно!***  *JavaScript является блокирующим ресурсом для парсера. Это означает, что JavaScript блокирует разбор самого HTML-документа. Когда парсер доходит до тега script (не важно внутренний он или внешний), он останавливается, забирает файл (если он внешний) и запускает его.*

*Если каждая инструкция размещается на отдельной строке, то разделитель можно не писать. В таких случаях JavaScript интерпретирует переход на новую строчку как разделитель команд для автоматического ввода точек с запятой завершающих инструкции.*

*<script>*

*alert("Сообщение");*

*break*

*</script>*

*Одна инструкция может располагаться на нескольких строчках:*

*<script>*

*var user = "Name";*

*alert("Рады вас видеть,*

*user +*

*"!");*

*</script>*

*п*

*В этом случае JavaScript ждёт завершение выражения и поэтому автоматически не вставляет виртуальную точку с запятой между строчками.*

*Тем не менее, рекомендуется всегда добавлять точки с запятой - это позволит избежать побочных эффектов.*

*К сведению. Хотя точки с запятой в конце инструкций необязательны, рекомендуется всегда добавлять их. Сейчас это правило, которому следуют все большие проекты.*

*Чувствительность к регистру*

*Для написания JavaScript-программ используется набор символов Unicode, который включает в себя наборы ASCII и Latin-1 и поддерживается практически всеми языками и платформами.*

*Тема 1. Введение в JavaScript*

*B JavaScript все элементы, включая имена переменных, функций и операторов, чувствительны к регистру и должны всегда содержать одинаковые наборы прописных и строчных букв.*

*Например, ключевое слово while должно набираться как while, а не While или WHILE.*

*Комментарии*

*Комментарии позволяют выделить фрагмент программы, который не выполняется интерпретатором JavaScript, а служит лишь для пояснений содержания программы.*

*Комментарии в JS могут быть однострочными и многострочными.*

*Однострочные комментарии начинаются с двойного слэша (//). Текст считается комментарием до конца строки:*

*<script>*

*// объявляем переменную*

*var num = 5;*

*alert( num ); // выводим переменную*

*</script>*

*Многострочный комментарий начинается с слэша и звездочки (/\*), а заканчивается ими же в обратном порядке (\*/). Так можно закомментировать одну и более строк:*

*<script>*

*/\* многострочный комментарий*

*эта часть кода не выполнится*

*var num = 5;*

*alert( num );*

*Тема 1. Введение в JavaScript*

*18*

*/\* многострочный комментарий на одной строке \*/*

*alert("Привет, Мир!");*

*</script>*

*К сведению. Не пренебрегайте комментариями в своих кодах. Они пригодятся вам при отладке и сопровождении программ. Есть мнение, что комментарии могут составлять до 30% от общего объема текста программ.*

Идентификаторы

*Идентификатор-это последовательность букв, цифр, символов подчёркивания (-) и знаков доллара ($). Цифра не может быть первым символом идентификатора, т. к. тогда интерпретатору JavaScript труднее отличать идентификаторы от чисел.*

*Идентификаторы выступают в качестве имён переменных, функций, свойств объекта и т. д.*

*Для совместимости и простоты редактирования для составления идентификаторов обычно используются только символы ASCII и цифры.*

*Однако в ЕСМАScript идентификаторы могут содержать буквы и цифры из полного набора символов Unicode. Это позволяет программистам давать переменным имена на своих родных языках и использовать в них математические символы.*

*Примечание. Хотя это и не запрещено стандартом, использовать русские символы в именах идентификаторов не принято.*

*B JavaScript наиболее популярным стилем именования идентификаторов, состоящих из нескольких слов, является camelCase - "верблюжья" нотация.*

*Примечание. В JavaScript объединение нескольких слов для записи идентификаторов с применением дефисов запрещено. Они зарезервированы для математических вычитаний.*

Ключевые и зарезервированные слова

*Стандарт ЕСМАScript определяет набор ключевых слов (keywords), которые не могут использоваться в качестве идентификаторов. Зарезервированные слова имеют определенное значение в языке JavaScript, так как они являются частью синтаксиса языка. Использование зарезервированных слов приведет к ошибке компиляции при загрузке скрипта.*

*Вот всего лишь некоторые зарезервированные ключевые слова: break, case, catch, class, while, continue, function....*

ECMAScript

*Чтобы понять, что такое ECMAScript и чем он отличается от JavaScript, нужно немного углубиться в терминологию.*

*Есть такая организация - Ecma International. Она создает стандарты для разных технологий. Самый простой пример стандарта клавиатура QWERTY. Дизайн может быть любым, но расположение клавиш обычно QWERTY.*

*Еста International разработала ЕСМА-262. Это стандарт скриптового языка.*

*В стандарте описана спецификация скриптового языка общего назначения. Эта спецификация называется ECMAScript.*

*К сведению. В спецификации ЕCMAScript содержатся рекомендации, сведения и правила, которые должны соблюдаться, чтобы скриптовый язык считался совместимым с ECMAScript.*

*JavaScript - это скриптовый язык, который соответствует спецификации ECMAScript.*

*Таким образом:*

*• Спецификация ECMAScript расскажет, как создать свой скриптовый язык.*

*• Документация JavaScript объяснит, как пользоваться скриптовым языком.*

*В самостоятельных работах к уроку:*

*• Создание внутренних скриптов.*

*• Создание и подключение внешних скриптов*

*•Проверка работоспособности скрипта.*

*P.S.*

*Для отработки и закрепления учебного курса в расширенной платной версии предоставляется следующий раздаточный материал.*

Переменные и константы

* *Объявление переменной*
* *Имена переменных*
* *Константы*
* *Вывод в документ*
* *Консоль браузера*
* *Повторное объявление переменных*
* *Строгий режим*
* *Кеширование файлов*

*JavaScript-сценарию обычно нужно работать с информацией. Информацией веб-приложения может быть все, что угодно.*

*Например:*

* *права доступа текущего пользователя;*
* *внешний вид отображаемой страницы;*
* *выбираемые пользователем товары и корзина покупок;*
* *критерии фильтрации и сортировки выводимых данных;*
* *и многое другое...*

*Переменные используются для хранения этой информации и являются основным понятием любого языка программирования.*

*Переменная- это именованная область памяти, в которой мы можем хранить какие-либо данные, например, строки или числа. Переменные похожи на условный контейнер в памяти компьютера. Этим контейнерам можно присваивать имена и впоследствии использовать эти имена для извлечения и изменения хранимых данных.*

*Прежде чем использовать переменную в JavaScript, ее необходимо объявить.*

# Объявление с помощью ключевого слова var

*Для объявления или, другими словами, создания переменной используется ключевое слово var (от variable - переменная):*

*var myLogin;*

*var age;*

*После объявления переменной ей можно присвоить некоторое значение, это называется инициализацией переменной:*

*var myLogin;*

*myLogin = "master"; // сохраним в переменной строку*

*// можно совместить объявление переменной и ее инициализацию*

*var age = 33; // сохраним в переменной число*

*Значение переменной сохраняется в соответствующей именованной области памяти и в дальнейшем доступ к этой области осуществляется при обращении по имени переменной. Если переменную не инициализировать, то ее начальное значение остается неопределенным undefined.*

*<script>*

*// объявляем переменную myLogin*

*var myLogin;*

*// инициализируем переменную текстовым значением*

*myLogin = "master";*

*// объявляем переменную age*

*var age;*

*// выводим значения переменных*

*alert (myLogin); // выведет содержимое переменной*

*alert(age); // выведет undefined*

# Особенности использования var:

*var не препятствует объявлению переменных с одинаковым именем; область видимости переменных var ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока; если блок кода находится внутри функции, то var становится локальной переменной в этой функции.*

Важно. *Более подробно переменные var будут рассмотрены при изучении управляющих конструкций. Сейчас достаточно понять сопровождающие урок примеры кода.*

Объявление с помощью ключевого слова let

*Ключевое слово let появилось в ECMAScript 6, его, упрощённо, можно назвать блочной версией var.*

*let msg = 'Hello';*

*Ключевое слово let - почти то же самое, что и var. Область видимости переменных, объявленных с помощью ключевого слова let, ограничивается блоком, оператором или выражением, в котором оно объявлено, а также вложенными блоками.*

*Примечание.*

*Тестирование различий между переменными let и var, будет проведено нами позже, пока эти различия не имеют для нас решающего значения.*

*example\_2. Объявление переменных с помощью let*

*<script>*

*// в простых примерах ничего не меняется*

*// объявляем и инициализируем переменную myLogin*

*let myLogin = "master";*

*// объявляем переменную аде*

*let age;*

*// выводим значения переменных*

*alert (myLogin); // выведет содержимое переменной*

*alert(age); // выведет undefined*

*</script>*

*Особенности использования let:*

*let не позволяет обратиться к переменной до её объявления;*

*• let препятствует объявлению в одной области видимости переменных (и/или функций) с одинаковым именем;*

*Тема 1. Введение в JavaScript*

*5*

*S*

*let является блочной или локальной (видна только внутри блока с областью видимости, ограниченной текущим блоком кода).*

*Важно. Более подробно переменные let будут рассмотрены при изучении управляющих конструкций. Сейчас достаточно понять сопровождающие урок примеры кода.*

*Имена переменных*

*Имена переменных могут начинаться с одного из следующих символов:*

* *буква в верхнем или нижнем регистре: a-z, A-Z*
* *символ нижнего подчеркивания:*

*Вслед за первым символом имени переменной можно использовать*

*последовательность букв, цифр и символа подчеркивания без пробелов и*

*знаков препинания.*

*Поскольку JavaScript чувствителен к регистру, буквы включают символы от А до 2 (верхний регистр) и символы от а до z (нижний регистр).*

*Примеры правильных имен переменных:*

*var myName;*

*var my\_adress;*

*var x;*

*var te1512\_7456;*

*Примеры неправильных имен переменных:*

*var 512group; // начало не может быть цифрой*

*· var my-adress; // дефис '-' не является разрешенным*

*символом*

*var my adress; // пробел не является разрешенным символом*

*При выборе имен переменных нельзя использовать ключевые слова, то есть слова, используемые в определениях конструкций языка. Например, нельзя выбирать слова var, return, class, const, true, false, function и ряд других.*

*Например, следующий сценарий выдаст синтаксическую ошибку:*

*var class = 15; // ошибка*

К сведению. *Имя переменной должно быть понятным и максимально чётко соответствовать хранимым в ней данным.*

*Существуют проверенные опытом правила именования переменных:*

*Имена, состоящие из одной буквы, равно как и имена, не несущие смысловой нагрузки, можно применять только в том случае, если они используются в небольшом фрагменте кода и их применение очевидно; используйте составные имена из нескольких слов, чтобы как можно точнее описать назначение переменных.*

*возможны два способа объединения слов для создания имени: использование "верблюжьей" нотации (camelCase) или запись слов через символ нижнего подчеркивания (snake\_case).*

*при выборе имен применяйте только английский язык, т.к. среди разработчиков всего мира принято использовать английский язык для имён переменных (впрочем, если вы планируете покорить мир своими программами..., ведь Стив Джобс тоже не стал бы тем, кем он стал, если бы действовал в шаблонах чужих правил;).*

# Константы

*Кроме переменных в JavaScript можно создавать константы. Константа - это переменная, которая не изменяется в процессе выполнения скрипта. Создать именованную константу можно используя ключевое слово const.*

*Для примера создадим константу, в которой будет хранится значение числа пи:*

*const pi= 3.14;*

*Как правило константы записываются большими буквами, чтобы их можно было отличить от обычных переменных:*

*const PI =3.14;*

*const AGE = 18;*

К сведению. *Имена констант пишутся буквами в верхнем регистре, через нижнее подчёркивание (если необходимо). В остальном синтаксис*

*идентификатора константы является таким же, как идентификатора обычной переменной.*

*Имена констант не должны совпадать с именами переменных, объявленных в той же области видимости, например, следующий сценарий выдаст синтаксическую ошибку.*

*const MY LOGIN = "master";*

*var MY LOGIN = "master"; // ошибка*

*Попытки изменить значение константы также приведут к ошибке.*

*const FLAG = 1;*

*FLAG = 2; // ошибка*

*alert(FLAG); // не сработает*

*Попытки изменить значение константы также приведут к ошибке.*

*const FLAG = 1;*

*FLAG = 2; // ошибка*

*alert (FLAG); // не сработает*

*Особенности использования констант:*

*Константа должна быть сразу инициализирована (а не только объявлена); константа не может быть переопределена.*

Вывод в документ

*С помощью средств JavaScript можно управлять элементами HTML страницы, например, выводить в них текст. Подробнее возможности манипулирования элементами страницы будем рассматривать в теме, посвященный работе с DOM.*

*Иногда, однако, в отладочных целях требуется вывести какой-нибудь текст в окно браузера. Это делается с помощью команды document.write. Вывод в браузер может выглядеть следующим образом:*

*// вывод простого текста*

*document.write ("Изучаем JavaScript!");*

*// вывод числового значения*

*document.write (12345);*

*// вывод текста из переменной*

*var email =*

*'master@mail.ru';*

*document.write (email);*

*Приведу еще несколько примеров вывода в браузер с помощью document.write.*

*<script>*

*// вывод текста, заключенного в HTML-теги*

*document.write('<b>text1</b>');*

*// или так*

*let str = 'text2';*

*document.write('<b>' + str + '</b>');*

*// вывод текста с переносом строки*

*document.write('text3 <br> text4');*

*// вывод большого объема текста с переносом строки*

*Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание*

*Консоль браузера*

*Код уязвим для ошибок. Программист не робот, ему свойственно ошибаться, свойственно делать ошибки в коде. Но по умолчанию в браузере ошибки не видны. Если в JS-код закралась ошибка, она не будет видна обычному посетителю. Обычной возможностью вывода document.write мы не увидим, что именно пошло не так, и не сможем это исправить.*

*Для удобства программистов в каждом браузере есть специальная панель разработчика, в которую встроены так называемые Инструменты разработки (Developer tools или сокращённо devtools).*

*С помощью специальных консольных команд вы можете выводить данные своих скриптов в консоль, это применяется для того, чтобы заниматься отладкой программ.*

*Для примера выведем что-нибудь в консоль. Запустите код сценария example\_4.ч*

*Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание*

*В приведенном скриншоте консоли мы можем увидеть сообщение об ошибке, отрисованное красным цветом. В данном случае сообщение говорит о том, что скрипт содержит неизвестную переменную "primary".*

*Справа присутствует ссылка на исходный код example\_4.html:18 с номером строки кода, в которой эта ошибка и произошла.*

*Под сообщением об ошибке находится синий символ >. Он обозначает командную строку, в ней вы можете редактировать и запускать JavaScript- команды.*

*Для запуска команды необходимо нажать Enter. Обычно при*

*нажатии Enter введённая строка кода сразу выполняется. Чтобы перенести строку, нажмите Shift+Enter. Таким образом, можно вводить многострочный 1Ѕ-код.*

*В примере ниже я дважды вывел значение определенной в сценарии переменной message, сначала в диалоговом окне alert, затем используя рассматриваемый далее консольный метод log.*

*console.log()*

*Выводит сообщение (логирует) в веб-консоль.*

*Синтаксис*

*console.log(obj1 [, obj2, ..., objN]);*

*console.log(msg I, objl,*

*....*

*objN]);*

*Параметры*

*• obj1, ..., objN - список объектов для вывода.*

*• msg - строка, предваряющая вывод.*

*example\_5. Тестируем console.log*

*<script>*

*// определим переменную*

*var msg = "Stop the World I Want to Get Off";*

*// определим массив*

*var arr = new Array("php", "sql", "javascript", "css");*

*// выведем данные в консоль*

*console.log (msg);*

*console.log(arr);*

*</script>*

*console.log()*

*Выводит сообщение (логирует) в веб-консоль.*

*Синтаксис*

*console.log(obj1 [, obj2, ..., objN]);*

*console.log(msg I, objl,*

*....*

*objN]);*

*Параметры*

*• obj1, ..., objN - список объектов для вывода.*

*• msg - строка, предваряющая вывод.*

*example\_5. Тестируем console.log*

*<script>*

*// определим переменную*

*-*

*var msg = "Stop the World I Want to Get Off";*

*// определим массив*

*var arr = new Array("php", "sql", "javascript", "css");*

*// выведем данные в консоль*

*console.log (msg);*

*console.log(arr);*

*</script>*

*Способ просмотреть в консоли список свойств указанного JavaScript объекта. Вывод представлен в виде иерархического списка с возможностью просмотра содержимого дочерних объектов.*

*Синтаксис*

*console.dir (obj);*

*Параметры:*

*• obj - JavaScript-объект свойства которого нужно вывести. Примечание:*

*Несколько заметок о разнице между console.dir() и console.log():*

*• console.dir может принимать только один аргумент, тогда как в console.log() мы можем передавать несколько аргументов. Если в console.dir() передается несколько аргументов, в консоль будет выведен только первый аргумент, а остальные аргументы будут проигнорированы;*

*• console.log выводит элемент как HTML-дерево;*

*• console.dir выводит элемент как JSON-объект;*

*• поведение методов может зависеть от браузера.*

*<script>*

*// получим ссылку на список*

*let web = document.querySelector("ul");*

*// выведем список в консоль*

*// смотрим разницу*

*console.log("Список:*

*console.dir (web);*

*</script>*

*console.info()*

*Выводит сообщения, содержащие некоторую информацию, в веб-консоль.*

*<script>*

*// получим ссылку на список*

*let web = document.querySelector("ul");*

*// выведем список в консоль*

*// смотрим разницу*

*console.log("Список:*

*console.dir (web);*

*</script>*

*п*

*web);*

*console.info()*

*Выводит сообщения, содержащие некоторую информацию, в веб-консоль.*

*Хотя различия между ключевыми словами var и let достаточно тонкие, и проявляются в написании сценариев, содержащих блоки кода, одно из них можно обнаружить уже сейчас.*

*Переменную, объявленную с помощью ключевого слова var, можно объявить повторно, такое переобъявление вполне допустимо.*

*Повторное объявление переменных var*

*<script>*

*// тестируем поведение переменных с ключевым словом var*

*// переменная с названием неплохого фильма*

*var myFilm "Зеленая миля";*

*// переменная с названием лучшего советского фильма*

*var myFilm "Тот самый Мюнхгаузен";*

*console.log (myFilm); // Тот самый Мюнхгаузен*

*</script>*

Строгий режим

*Строгий режим (strict mode) — это важная часть современного JavaScript. Именно этот режим позволяет разработчикам пользоваться более*

*ограниченным, чем стандартный, синтаксисом.*

*Семантика строгого режима отличается от традиционного нестрогого режима, который иногда называют "грязным" (sloppy mode).*

К сведению. *В "грязном" режиме синтаксические правила языка не так строги, а когда происходят некоторые ошибки, система никак не оповещает о ошибки могут быть проигнорированы, а код, в*

*них пользователя. То есть котором они допущены, сможет выполняться дальше.*

*Это способно привести к неожиданным результатам выполнения кода.*

*Строгий режим вносит в семантику Javascript некоторые изменения. Он не дает системе закрывать глаза на ошибки, выдавая соответствующие исключения.*

*Это приводит к остановке выполнения программ.*

*Строгий режим, кроме того, помогает в написании программ, в которых нет недочётов, мешающих JS-движкам оптимизировать код. В строгом режиме запрещено использование элементов синтаксиса, которые могут получить особый смысл в будущих версиях языка.*

*Особенности применения строгого режима*

*Строгий режим можно применять к отдельным функциям или к целому скрипту. Его нельзя применить только к отдельным инструкциям или к блокам кода, заключённым в фигурные скобки.*

*Примечание. Наиболее ярко различия режимов проявляются при работе с*

*блоками кодов, функциями, объектами. Сейчас важно понять, что такое строгий режим и как он влияет на код сценария.*

*Для того чтобы использовать строгий режим на уровне целого скрипта, в самое начало файла, до любых других команд, нужно поместить конструкцию "use strict" или 'use strict'.*

*<script>*

*"use strict";*

*Если в проекте имеются некоторые скрипты, в которых не используется строгий режим, и другие, в которых этот режим используется, тогда может случиться так, что эти скрипты окажутся объединены.*

*Это приведёт к тому, что код, который не предназначен для выполнения в строгом режиме, окажется в таком состоянии, когда система попытается выполнить его в строгом режиме. Возможно и обратное код, написанный для строгого режима, попадёт в нестрогий режим. Поэтому лучше всего не смешивать "строгие" и "нестрогие" скрипты.*

*Преобразование "тихих" ошибок в исключения*

*Строгий режим влияет и на синтаксис кода, и на то, как код ведёт себя во время выполнения программы. Ошибки в коде преобразуются в исключения. То, что в нестрогом режиме тихо даёт сбой, в строгом вызывает сообщение об ошибке*

*"Тихие" ошибки преобразуются в строгом режиме в исключения. В нестрогом режиме на такие ошибки система явным образом не реагирует. В строгом же режиме наличие таких ошибок приводит к неработоспособности кода.*

*Так, благодаря этому сложно совершить ошибку случайного объявления глобальной переменной, так как переменные и константы в строгом режиме нельзя объявлять без использования директив var, let или const. В результате создание переменных без этих директив приведёт к неработоспособности программы.*

Поясню на примере. *Следующий фрагмент кода не выдаст никаких предупреждений и создаст глобальную переменную myVariable.*

*<script>*

*myVariable = "It's my life";*

*</script>*

*Но такой код нельзя запустить в строгом режиме. Строгий режим защищает программиста от непреднамеренного создания глобальных переменных.*

*<script>*

*'use strict';*

*myVariable = "It's my life";*

*</script>*

*> Uncaught ReferenceError: myVariable is not defined*

*example\_11. Применение режима use strict*

*<script>*

*'use strict'; // включаем строгий режим*

*// myVariable = "Test use strict";*

*var myVariable = "Test use strict";*

*console.log(myVariable);*

*</script>*

Кеширование файлов

*Веб-кэш располагается между веб-сервером и клиентом и следит за входящими запросами, сохраняя при этом копии ответов НТМL-страниц, скриптов, изображений и прочих файлов (контент), для собственных нужд. Затем, если поступает другой запрос с аналогичным URL-адресом, кэш может использовать сохраненный прежде ответ, вместо повторного запроса к серверу.*

*Существует две основные причины, по которым используется веб-кэш:*

*1. Уменьшение времени ожидания - так как данные по запросу берутся из кэша (который располагается "ближе" к клиенту), требуется меньше времени для получения и отображения контента на стороне клиента. Это делает веб более отзывчивым.*

*2. Снижение сетевого трафик - повторное использование контентаснижает объем данных, передаваемых клиенту.*

Важно. *Браузер может кэшировать подключенные JavaScript файлы. Это значит, что он сохраняет их у себя для повышения скорости загрузки сайта.*

*На практике это ведет к тому, что в какой-то момент при редактировании кода браузер будет применять предыдущую версию кода, а не вашу текущую. Для борьбы с таким поведением можно чистить кэш браузера.*

*Виды веб-кэшей:*

*• Кэш браузера (Browser cache).*

*Эта опция позволяет выделить область жесткого диска на вашем компьютере для хранения просмотренного ранее контента. Кэш браузера работает согласно довольно простым правилам. Он просто проверяет являются ли данные "свежими", обычно один раз за сессию (то есть, один раз в текущем сеансе браузера).*

*Этот кэш особенно полезен, когда пользователь нажимает кнопку "Назад" или кликает на ссылку, чтобы увидеть страницу, которую только что просматривал. Также, если вы используете одни и те же изображения навигации на вашем сайте, они будут выбираться из браузерного кэша почти мгновенно.*

*• Прокси-кэш (Ргоху сасһе)*

*Прокси-кэш работает по аналогичному принципу, но в гораздо большем масштабе. Прокси обслуживают сотни или тысячи пользователей; большие корпорации и интернет-провайдеры часто настраивают их на своих файрволах или используют как отдельные устройства (intermediaries).*

*Прокси-кэши являются своего рода общей кэш-памятью (shared cache): вместо обслуживания одного человека, они работают с большим числом пользователей и поэтому очень хороши в сокращении времени ожидания и сетевого трафика. В основном, из-за того, что популярный контент запрашивается много раз.*

# Кэш-шлюз (Gateway Cache)

*Также известные как "реверсивные прокси-кэши" (reverse prоxу сасһе) или "суррогаты" (surrogate cache) шлюзы тоже являются посредниками, но вместо того, чтобы использоваться системными администраторами для сохранения пропускной способности канала, они (шлюзы) обычно используются веб- мастерами для того, чтобы сделать их сайты более масштабируемыми, надежными и эффективными.*

*Сети доставки контента (content delivery networks, CDN) распространяют шлюзы по всему интернету (или некоторой его части) и отдают кэшированный контент заинтересованным веб-сайтам.*

Типы данных

*• Динамическая типизация*

*· Типы данных*

*• Примитивные типы данных*

*• Ссылочные типы данных*

# Динамическая типизация

*В процессе работы компьютерные программы манипулируют различными значениями, каждое из которых может быть обработано в языке программирования и относится к определённому типу данных.*

*В JavaScript типы данных можно разделить на две категории:*

* *простые (примитивные) типы;*
* *составные (ссылочные) типы.*

*JavaScript-это слабо типизированный или динамический язык программирования, который позволяет определять типы данных, осуществлять синтаксический анализ и компиляцию "на лету", на этапе выполнения программы. Это значит, что вам не нужно определять тип переменной заранее. Тип определится автоматически во время выполнения программы.*

*Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов.*

*example\_1. Слабая типизация JavaScript*

*<script>*

*// инициализируем переменную значениями разных типов*

*var value = "string"; // строка*

*value = 7; // число*

*<script>*

*// экспериментируем с типами данных*

*var value = "string"; // строка*

*console.log(typeof(value)); // string*

*value = 7; // простое число*

*console.log(typeof(value)); // number*

*value = {name: "Иван", login: "master"}; // объект*

*console.log(typeof(value)); // object*

*value = ["php", "javaScript", "python"]; // массив*

*console.log(typeof(value)); // object (обратить внимание)*

*value = true; // булев тип*

*console.log(typeof(value)); // boolean*

*value = null; // специальный тип null*

*console.log(typeof(value)); // object*

*</script>*

# К сведению.

*• Для значения массива оператор typeof возвращает значение object.*

*Позже мы рассмотрим, что массив является частным случаем объекта. Для значения null оператор typeof возвращает значение object несмотря на то, что согласно спецификации JavaScript значение null представляет отдельный тип.*

Интерполяция

*Использование косых кавычек позволяет применять такой прием как интерполяция встраивание данных в строку.*

*Например:*

*let user = "Eugene";*

*// интерполируем переменную в строку*

*let msg = 'Hello our dear ${user}!';*

*console.log (msg); // Hello our dear Eugene! Веб-разработка | Профессионалы | Демоэкзамен* [*https://vk.com/pechora\_pro*](https://vk.com/pechora_pro)

*Для встраивания значений выражений (значений переменных и констант) в строку перед выражением ставится знак доллара $, после которого в фигурных скобках указывается выражение. Так, в примере выше ${user} означает, что в этом месте строки надо встроить значение переменной user.*

*Подобным образом можно встраивать и больше количество данных:*

*example\_6. Пример интерполяции переменных*

*<script>*

*let login = "master";*

*let id = 3157;*

*let access = true;*

*let msg = Login: $(login} (id=${id}), allow full access: $(access};*

*// вывод в консоль сформированного сообщения*

*console.log (msg);*

*// Login: master (id=3157), allow full access: true*

*</script>*

*Разница между примитивными и составными типами*

*Разница между примитивными и составными типами данных проявляется при копировании их значений.*

*Вспомним. Переменная-это поименованная область памяти компьютера, Отводимая под хранение некоторого значения.*

*Когда переменной присваивается значение простого типа, то в переменную записывается само значение (например число).*

*Когда мы присваиваем переменную простого типа в другую переменную, она копирует значение. В результате каждая переменная будет иметь своё Веб-разработка | Профессионалы | Демоэкзамен* [*https://vk.com/pechora\_pro*](https://vk.com/pechora_pro)

*Важно. Когда переменной присваивается значение простого типа, то в переменную записывается само значение.*

*Когда переменной присваиваем значение составного (ссылочного) типа, то в переменную записывается ссылка на значение (ссылка на объект).*

*Веб-разработка | Профессионалы | Демоэкзамен https://vk.com/pechora\_pro*

*Тема 1. Введение в JavaScript 6*

*Когда мы присваиваем одну переменную (в значении которой ссылка на составное значение) другой переменной, то происходит копирование ссылки на составное значение. В результате обе переменные ссылаются на одно и то же составное значение и изменения в значении одной из переменных будут сказываться на другой переменной.*

*Важно. Когда переменной присваиваем значение составного типа, то в переменную записывается ссылка на значение (ссылка на объект).*

*Поэкспериментируем с переменной типа объект. Объект состоит из набора свойств в виде пар ключ: значение. Для обращения к свойствам объекта используется точечная нотация:*

*// инициализируем объект*

*var obj1 = {x:10};*

*// выводим свойство объекта в браузер*

*document.write( obj.x ); // 10*

*Поэкспериментируем с переменной типа массив. Массив состоит из набора значений. Для обращения к значениям используется индекс, обозначающий порядковый номер значения в массиве. Индекс указывается в квадратных скобках, отсчет индексов начинается с 0:*

*// инициализируем массив*

*var arr = ["php", "javaScript", "python"]*

*// выводим второй элемент ммассива*

*document.write(arr[1]); // javascript*

*example\_5. Присваивание переменных составного типа (массив)*

*<script>*

*// создадим массив*

*// массив -переменная ссылочного типа*

*var arr1 = ["php", "javaScript", "python"]; //исходный*

*массив*

*var arr2 = arr1; // присваиваем массив*

*// при этом значения исходного массива не копируется,*

*// обе переменные будут ссылаться на одну область памяти*

*document.write("<h2>Начальные значения второго элемента*

*массивов</h2>");*

*document.write("arr1[1]*

*document.write("<br>arr2[1]*

*+ arr1[1]); // javaScript*

*+ arr2[1]); // javaScript*

*document.write("<h2>Изменяем значение второго элемента*

*первого массива</h2>");*

*// изменим второй элемент первого массива*

*// второй массив должен измениться соответственно*

*arr1[1] = "JS";*

*document.write("<br>arr1[1]*

*document.write("<br>arr2[1]*

*</script>*

*+ arr1[1]); // JS*

*+ arr2[1]); // JS*

*<script>*

*// создадим объект*

*var obj1 = {x:10}; // объект - переменная ссылочного типа*

*var obj2 = obj1; // присваиваем одну переменную другой,*

*// при этом значение не копируется,*

*// обе переменные будут ссылаться на одну область памяти*

*document.write("<h2>Начальные значения свойств*

*объектов</h2>");*

*document.write("obj1.x – " + obj1.x); // 10*

*document.write("<br>obj2.x – " + obj2.x); // 10*

*document.write("<h2>Изменяем значение свойства объекта*

*obj1</h2>");*

*// изменим свойство объекта obj1*

*// свойство объекта obj2 должно измениться автоматически*

*obj1.x = 25;*

*document.write("obj1.x – " + obj1.x); // 25*

*document.write("<br>obj2.x – " + obj2.x); // 25*

*</script>*

Преобразование типов

* *Явное и неявное преобразование*
* *Преобразование к типу String*
* *Преобразование к типу Number*
* *Преобразование к типу Boolean*
* *Функции parseInt и parseFloat*
* *Операторы вместо функций*

***Введение***

*DOM (Document Object Model) была придумана создателями браузеров, и является основным инструментом работы и динамических изменений на странице. Консорциум World Wide Web Consortium (W3C) стандартизировал DOM как единую спецификацию.*

*Концепция DOM реализована с разной степенью поддержки во всех основных браузерах. С помощью объектной модели можно создавать документы, перемещаться по структуре документа, а также добавлять, изменять или удалять элементы и содержимое.*

***Важно.*** *DOM является основным инструментом построения и динамических изменений страницы HTML-документа.*

*Согласно DOM-модели, документ является иерархией. Каждый HTML-тег образует отдельный элемент-узел, каждый блок текста - текстовый узел, и т.п.*

*Другими словами, DOM - это представление документа в виде дерева тегов. Это дерево образуется за счет вложенной структуры тегов плюс текстовые блоки страницы, каждый из которых образует отдельный узел.*

*Перед тем, как перейти к изучению объектной модели документа необходимо вспомнить, каким образом формируется исходный код веб-страницы (HTML документа).*

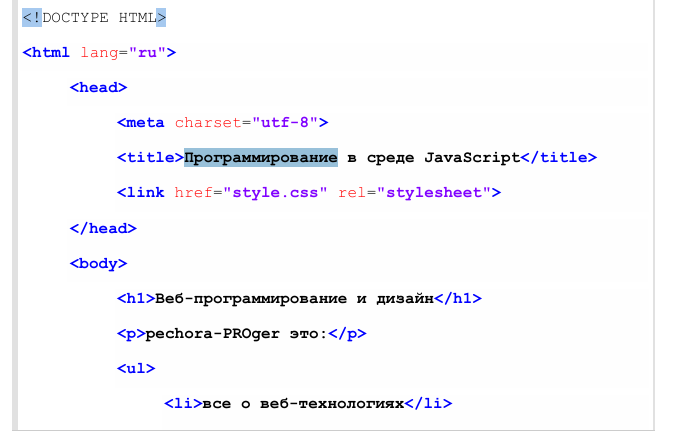
*Исходный код веб-страницы состоит:*

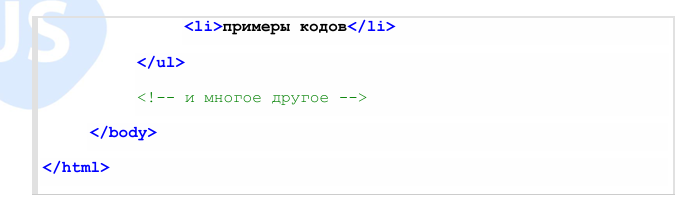
*• теги;*

*• атрибуты;*

*• текст;*

*• комментарии*

**

**

*В этом коде корневым элементом является элемент html.*

*• элемент html содержит вложенные элементы head и body;*

*• элемент head содержит вложенные meta, title, link; • элемент body содержит вложенные h1, p, ul, комментарий;*

*• элемент ul в свою очередь, содержит вложенные элементы li.*

*Теперь рассмотрим, как браузер на основании HTML-кода строит DOM-дерево.*

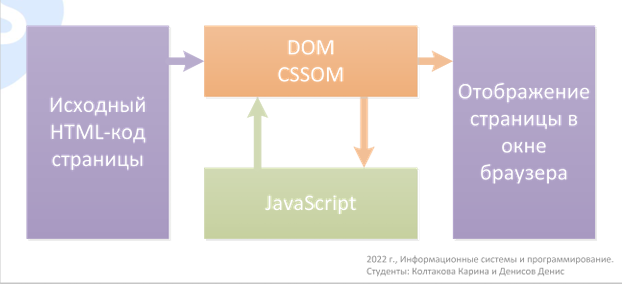
*• Механизм рендеринга браузера разделяет HTML, CSS и JS-файлы полученного кода запрошенной им веб-страницы.*

*• После разделения содержимого страницы, HTML и CSS-файлы анализируются по частям. В процессе анализа и разбора входных HTML и CSS данных браузер строит на их основе DOM и CSSOM-деревья (CSS Object Model). Деревья являются независимыми объектами, которые отражают различные аспекты документа: один описывает содержимое, а другой описывает правила стиля, которые необходимо применить к документу.*

*• Дерево визуализации создается путем объединения деревьев DOM и CSSOM, и содержит как информацию о порядке элементов, так и информацию об их стиле. На этом этапе каждому узлу присваиваются специальные координаты, чтобы гарантировать, что каждый элемент будет правильно позиционироваться на экране.*

*• После выполнения этого действия браузер приступает к отображению страницы путем обхода дерева визуализации, и каждый узел, упомянутый в дереве, отображается на экране. В этом процессе браузер использует созданное таким образом DOM-дерево, а не исходный HTML-код*

***Важно.*** *DOM – это объектная модель документа, которую браузер создаёт в памяти компьютера на основании HTML-кода, полученного им от сервера.*

**

*Другими словами, HTML-код – это исходный текст страницы, а DOM – это набор связанных объектов, созданных браузером при синтаксическом анализе (парсинге) её текста.*

***Критический путь рендеринга (CRP)***

*Загрузка веб-страницы или приложения начинается с запроса HTML. Сервер возвращает HTTP-ответ, состоящий из заголовков и тела ответа. Именно в теле ответа содержится HTML-документ.*

*Браузер начинает парсить загружаемый HTML, преобразуя полученные байты документа в DOM-дерево. Браузер создаёт новый запрос каждый раз, когда он находит ссылки на внешние ресурсы, будь то файлы стилей, скриптов или ссылки на изображения.*

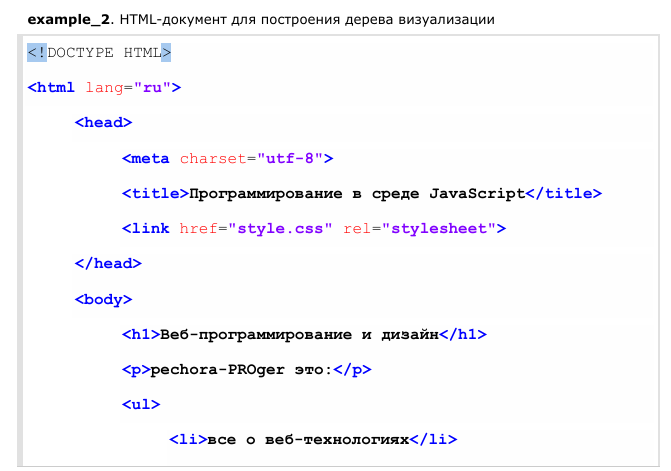
*Некоторые запросы являются блокирующими. Это означает, что пока такие запросы выполняются - другие запросы приостанавливаются. Браузер продолжает парсить HTML и создавать DOM до тех пор, пока процесс получения исходного кода HTML не подходит к концу. После завершения построения DOM, браузер конструирует CSS модель. Как только эти модели.*

***Дерево DOM***

*Объектная модель документов (DOM) является основным инструментом динамических изменений на странице. DOM дерево это объект, представляющий полностью разобранную HTML-страницу.*

*Элементы, вложенные в другие элементы, представлены в виде дочерних узлов образуя таким образом иерархию узлов. Каждый узел содержит всю необходимую информацию об HTML-элементе, соответствующем этому узлу. Чем больше количество узлов имеет приложение, тем дольше происходит формирование дерева DOM, а значит дольше происходит обработка критических этапов рендеринга. Связи между узлами в модели формируются на основании того, как HTML элементы расположены в коде относительно друг друга.*

***Другими словами****. DOM API – это набор огромного количества различных объектов, их свойств и методов, которые мы можем использовать для чтения и изменения DOM.*

**

*Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание*

*Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание*

***К сведению****. DOM — это представление документа в виде дерева тегов, доступное для изменения через JavaScript.*

*CSSOM-представление файла style.css, представленного в example\_3, приводится на рисунке 3.*

*Изображение выглядит как текст, снимок экрана, дисплей, Шрифт

Автоматически созданное описание*

*Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание*

***Исполнение JavaScript***

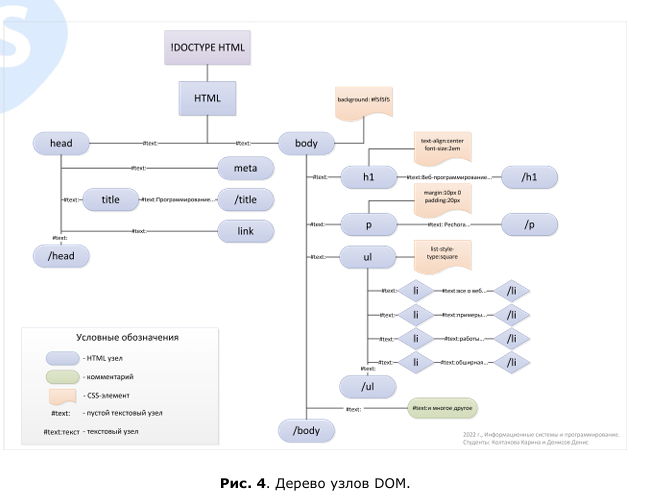
*Для работы с DOM в большинстве случаев используется JavaScript, т.к. на*

*сегодняшний день это самый популярный язык программирования скриптов,*

*исполняемых браузером.*

*JavaScript является блокирующим ресурсом для парсера. Это означает, что JavaScript блокирует разбор самого HTML-документа.*

*Когда парсер доходит до тега script (не важно внутренний он или внешний), он останавливается, забирает файл (если он внешний) и запускает его. Вот почему, если мы имеем JavaScript-файл, который ссылается на элементы документа, мы обязательно должны поместить его после их появления.*

**

*Существует 12 типов узлов.*

*Но на практике мы в основном работаем с 4 из них:*

*• document – "входная точка" в DOM;*

*• узлы-элементы – HTML-теги (основные строительные блоки);*

*• текстовые узлы – содержат текст (в том числе пустой текст);*

*• комментарии – иногда в них можно включить информацию, которая не будет показана, но доступна в DOM для чтения с помощью JS.*

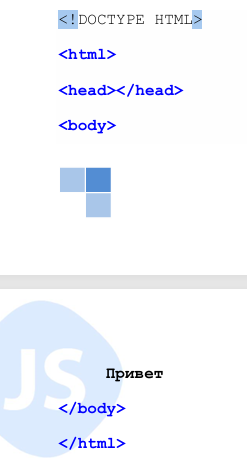
***Важно.*** *Дерево состоит из узлов, но только некоторые из них являются HTML-элементам.*

***Автоисправление***

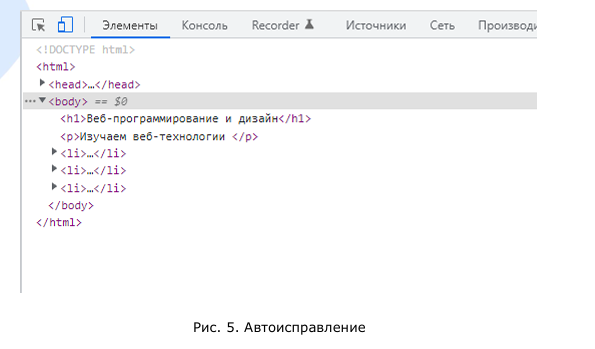
*Если браузер сталкивается с некорректно написанным HTML-кодом, он автоматически корректирует его при построении DOM.*

*Например, в начале документа всегда должен быть тег html. Даже если его нет в документе – он будет в дереве DOM, браузер его создаст. То же самое касается и тега body.*

*Например, если HTML-файл состоит из единственного слова "Привет", браузер обернёт его в теги html и body, добавит необходимый тег head, и документ будет выглядеть так:*

**

**

**

***Возможности, которые дает DOM***

*DOM API нужен, для реализации возможности, используя JavaScript изменять страницу на "лету", то есть, делать её динамической и интерактивной.*

*DOM API предоставляет разработчикам огромное количество методов, с помощью которых он может создавать сложные интерфейсы, формы, выполнять обработку действий пользователей, добавлять и удалять различные элементы на странице, изменять их содержимое, свойства (атрибуты), и многое другое.*

*Сейчас в вебе практически нет сайтов, в сценариях которых отсутствовала бы работа с DOM.*

*Итого:*

*• DOM-модель — это внутреннее представление HTML-страницы в виде дерева;*

*• все элементы страницы, включая теги, текст, комментарии, являются узлами DOM;*

*• у элементов DOM есть методы, которые позволяют взаимодействовать с этими элементами;*

*• DOM-модель используется не только в JavaScript, это известный способ представления XML-документов*

# ***Доступ к DOM***

***Методы выбора DOM элементов***

*Объектная модель документа (DOM) — это независимый от платформы и языка интерфейс, который позволяет программам и скриптам динамически обращаться и изменять содержимое, структуру и стили документа. DOM представляет документ в виде иерархичного дерева узлов (элементов).*

*Если коротко, то DOM-дерево, состоит из узлов, которые вместе образуют иерархию, аналогичную HTML. В терминах DOM каждый тег - это узел (с типом: узел элемента). Просто текст тоже представлен узлом (с типом: текстовый узел).*

*Вообще существует 12 типов узлов, но на практике чаще всего приходится работать с узлами следующих типов:*

*document - "входная точка" в DOM;*

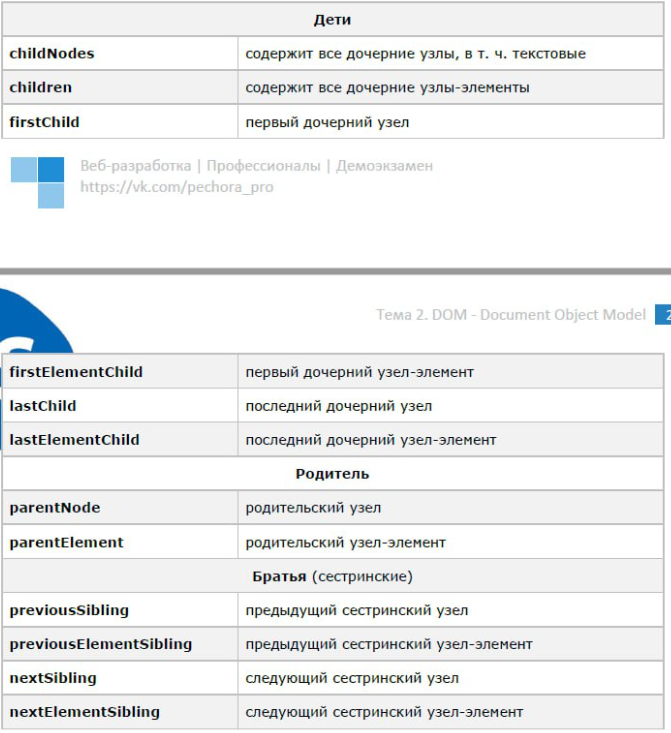
*узлы-элементы НТМL-теги (основные строительные блоки);*

*текстовые узлы - содержат текст (в том числе пустой текст);*

*комментарии иногда в них можно включить информацию, которая не будет показана, но доступна в DOM для чтения с помощью JЅ.*

*Дерево состоит из узлов, но только некоторые из узлов являются HTML-*

*элементами.*

**

*Конкретные узлы, чаще всего, описывают собой конкретные теги из HTML и*

*содержат их атрибуты внутри себя. У узлов есть тип, который определяет набор свойств и методов узла.*

*Важно. Свойства и методы узла определяются типом, к которому он*

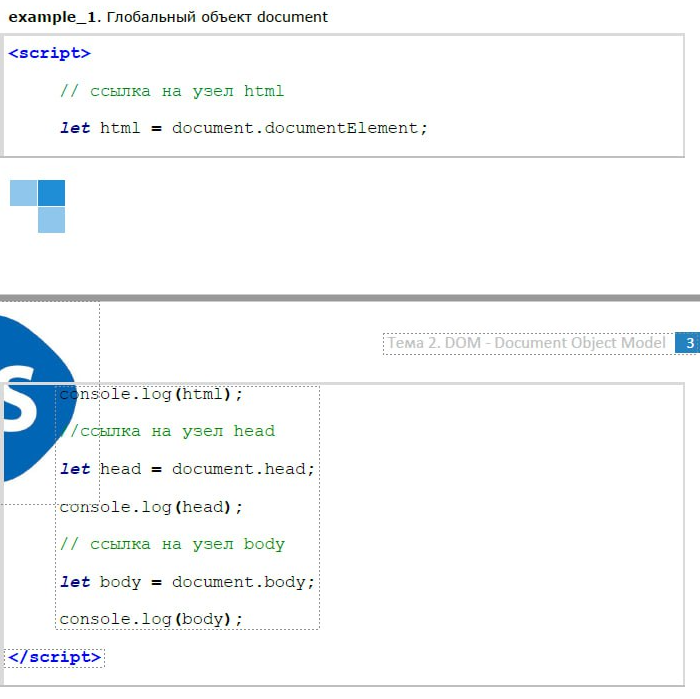
*принадлежит. Узлы-элементы (теги) имеют один набор, комментарии другой и т.д.*

*Методы для работы с DOM содержатся в глобальном объекте document. Document содержит все узлы DOM и функции для работы с ним:*

*• document.documentElement - ссылка на узел html*

*• document.body - ссылка на узел body*

*• document.head ссылка на узел hеаd*

**

***Важно.*** *Если скрипт подключен до body, то document.body будет равен null.*

*Это справедливо для любого другого тега если скрипт пытается получить доступ к элементу, который объявлен ниже по коду, результатом будет null.*

*Свойства, позволяющие получить первый и последний дочерние узлы (это может быть текст, комментарий и т.д.):*

*• firstChild*

*• lastChild*

***Свойства firstChild, lastchild***

***Изображение выглядит как текст, снимок экрана, число, Шрифт

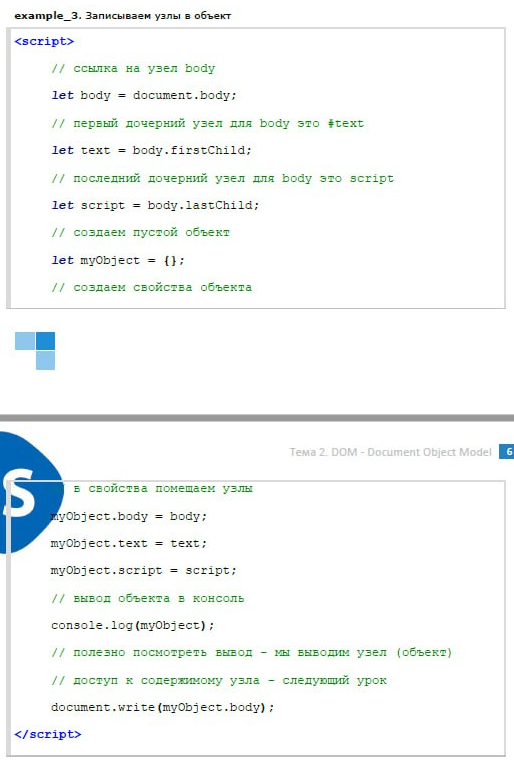
Автоматически созданное описание***

*Дополним код примера example\_2 созданием объекта. Объект в качестве значения свойства может хранить любой тип. Запишем полученные узлы дерева ДОМ в соответствующие свойства объекта.*

*Обратите внимание на инструкцию вывода в браузер:*

*document.write(myObject.body);*

*Узел выводим как объект дерева узлов DOM. С содержимым узлов будем работать в следующем уроке.*

**

*Свойства, позволяющие получить первый и последний дочерние html узлы:*

*• firstElementChild*

*• lastElementChild*

*Свойства firstElementChild, lastElementChild*

*Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание*

*Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание*

*Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание*

*В следующем примере запишем полученные узлы дерева DOM в элементы*

*массива.*

*Обратите внимание на инструкцию вывода в браузер:*

*document.write(myArray[0]);*

*В данной инструкции выводим узел как объект дерева узлов DOM.*

***Важно.*** *Будьте крайне внимательны при "вышагивании" по дереву DOM. Узел-элемент, пустой текстовый узел, узел-комментарий, все это необходимо учитывать. Именно по этой причине я не могу вставить поясняющие*

*комментарии в HTML-код, сильно изменится дерево узлов.*

*Свойства, позволяющие получить следующий и предыдущий html-узел (элемент):*

*nextElementSibling*

*previousElementSibling*

*Свойства nextElementSibling, previousElementSibling*

*Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание*

*Свойства, позволяющие получить дочерние узлы (непосредственно лежат в данном узле):*

*childNodes―дочерние узлы. К дочерним узлам также относятся текст,*

*комментарии, переносы строки и другие символы*

*• children*

*- дочерние html-узлы (элементы)*

*Свойства childNodes, children*

*Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание*

*Например строки:*

*let body = document.body;*

*let header = body.firstElementChild;*

*let div header.nextElementSibling;*

*Можно заменить строкой:*

*Let div ocument.body.firstElementChild.nextElementSibling;*