**Практическое занятие №9 Введение в JS**

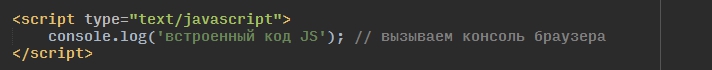
**Задание №1**

Проверьте запущен ли у Вас Open Server. Папка с материалами урока (js9.loc) должна быть размещена в директории /domains сервера. Заходим на <http://js9.loc> и начинаем урок.

**Задание №2**

Существует несколько способов подключения JS. В этом задании мы их рассмотрим.

***Встраивание (embedding)*** - код сценариев пишется непосредственно между тэгами <script>. Это самый простой способ. Код JS располагается непосредственно на самой html странице. Открываем файл index-2.html и размещаем тег <script>, как показано ниже:

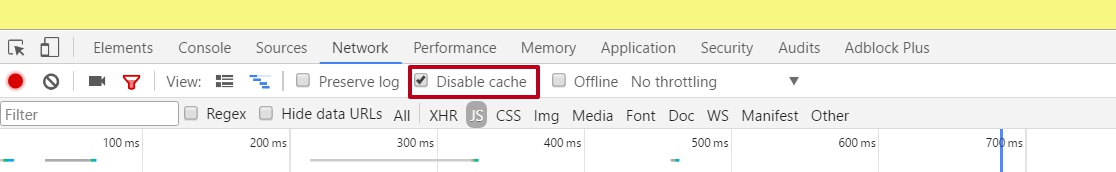


Этот код может быть размещён как в секции <head> так и <body>, на данный момент разницы для нас нет.

***console.log()*** - Выводит значения в консоль браузера. Подробнее об этом методе Вы узнаете на следующих занятиях.

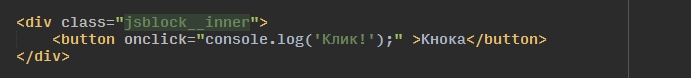
Как увидеть результат работы скрипта? Перезагрузить страницу. Открыть консоль браузера (DevTools, вкладка «Console»).

**ЭТО ВАЖНО!** Отключите кеширования в настройках DevTools (Вкладка Network->Disable cache)!



***Вложение (inline)*** - код сценариев JS пишется в значении атрибутов событий (изучается на следующих занятиях).

Для того что бы Вы имели начальное представление о вложениях мы напишем сейчас небольшой пример вложения и убедимся, что он работает. Создаем еще один блок «.jsblock\_\_inner» сразу после первого и в нем создайте кнопку с атрибутом onclick, как показано ниже:



Если Вы сделали все верно, то при клике на кнопку в консоли будет выводиться новое сообщение. Не забудьте перезагрузить страницу после написания кода.

***Связывание (linking)*** – с помощью атрибута src тега <script> указывается адрес внешнего файла. Этот вид внедрения сценариев предпочтителен и обладает преимуществами перед методом встраивания. У элемента <script> используется атрибут type=“text/javascript” для обозначения типа скрипта. Значение этого атрибута указывает браузеру на то, каким модулем обрабатывать этот контент (его можно встретить в таких элементах, как <style>, <object>, <embed>, <link>). Раньше использовался атрибут language, но в настоящее время использовать его не рекомендуется.

В папке задания создайте файл custom.js с одной строчкой JS кода:

http://dl3.joxi.net/drive/2017/05/26/0004/3207/285831/31/4e8a635cc6.jpg

и подключите его к странице в разделе <head>:

http://dl3.joxi.net/drive/2017/05/26/0004/3207/285831/31/3c93bec612.jpg

Перезагрузите страницу и в консоли должны быть сообщения, сгенерированные в custom.js.

В этом задании мы рассмотрели способы подключения JS к HTML странице. Идем дальше.

**Задание №3**

Создайте файл JS в папке урока и подключите его к странице урока (index-3.html). Создайте пять переменных, с произвольными именами. Присвойте им произвольные строковые значения. Затем выведите переменные в консоль браузера.

http://dl4.joxi.net/drive/2017/05/26/0004/3207/285831/31/f904af8499.jpg

ЭТО ВАЖНО! Не забывайте использовать разделитель инструкций «;»!

ЭТО ВАЖНО! Не забывайте про правила наименования переменных!

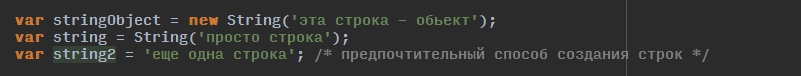
Убедитесь, что в консоли у Вас не ошибок и все данных выведены.

**Задание №4**

В этом задании мы узнаем, как создавать строковый тип данных в JS.

Создайте файл JS в папке урока и подключите его к странице урока.

Строки в JS это объекты определенного типа, создать строку можно с помощью вызова класса-конструктора или с помощью строкового литерала, выполните код как показано ниже:



Теперь проверим тип созданных нами переменных stringObject, string с помощью оператора typeof:



Теперь выведем переменные напрямую в консоль:



Какие выводы мы можем сделать? В переменной stringObject находится объект, а в string и string2 – строки. Объект имеет сложную структуру, строки – простую. Это связано с тем что объект содержит в себе данные и информацию как работать с этими данными (методы). Базовая структура объекта показана на рисунке ниже.

В строковом объекте находится сама строка, это его данные (левая желтая часть на рисунке), а также в нем находятся методы, которые могут работать с этой строкой (правая часть) с ними мы познакомимся подробнее в следующих заданиях (также более подробно будет рассмотрена объектная модель в JS).

**Задание №5**

В JS строку можно создать, используя двойные и одинарные кавычки. Это дело вкуса. Правило одно внутри двойных кавычек можно свободно использовать одинарные, и наоборот.

В файле задания у Вас уже есть закомментированные шесть строк, нужно их раскомментировать (убрать символы // в начале каждой строки) и поместить каждую строку в отдельную переменную, затем вывести переменные в консоль браузера. Обратите внимание, что некоторые строки содержат внутри кавычки.

**Задание №6**

Особое значение имеет символ “\” (обратная косая черта или обратный «слеш»). В совокупности с некоторыми другими символами он образует так называемые «управляющие последовательности». Эти последовательности могут обозначать непечатаемые символы и знаки. Далеко не все управляющие последовательности нужны для отображения на мониторе. Некоторые требуются для управления выводом на других типах устройств, например, на принтере.

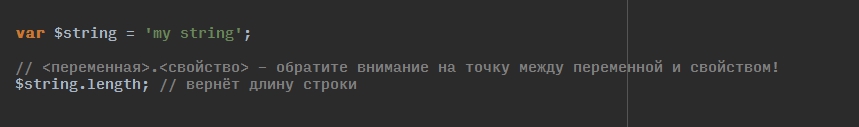
С помощью управляющих последовательностей можно внутри строки, ограниченной двойными кавычками, поставить двойную кавычку \”. Этот прием называется «экранированием».

Сделайте то же самое, что и в предыдущем задании, только используя экранирование. Например, если в строке встречается двойная кавычка, то строку мы обрамляем двойными кавычками, а внутри кавычки экранируем.



**Задание №7**

Свойство строки ***length***- возвращает длину строки, вызываются свойства как показано ниже:



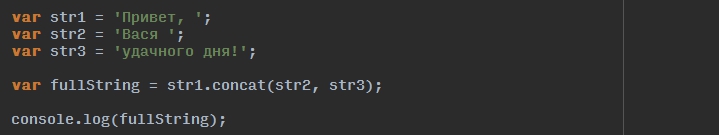
Скопируйте все свои переменные со строками из предыдущего задания и выведите в консоль длины их строк с помощью свойства ***length***.

ЭТО ВАЖНО! Как это возможно? Очень просто! Когда мы пытаемся вызвать метод или свойство у простой строки JavaScript «в тихую» создаёт объект и уже от имени этого объекта вызывает метод или свойство, после чего объект уничтожается.

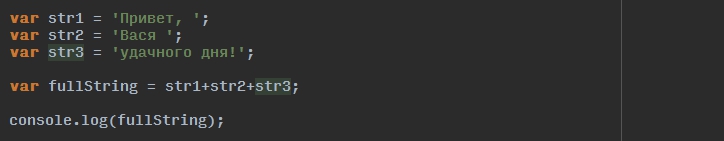
**Задание №8**

В этом задании мы проведём обзор основных методов типа данных String. Работаем по простой схеме: сначала идёт описание работы метода, и за тем демонстрационный пример. Вы набираете демо примеры и убеждаетесь в их работоспособности и так далее.

**.concat( string1 [, string2] )**- Объединяет исходную строку и строки-аргументы. На практике используют оператор сложения “+”;



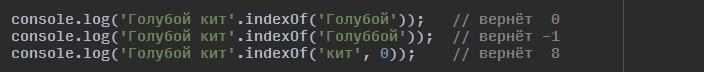
Тоже самое только используя оператор сложения строк «+» (конкатенацию).



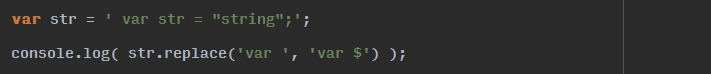
**.charAt(n)** – возвращает символ строки по указанной позиции.

http://dl3.joxi.net/drive/2017/05/26/0004/3207/285831/31/abcd74e53b.jpg

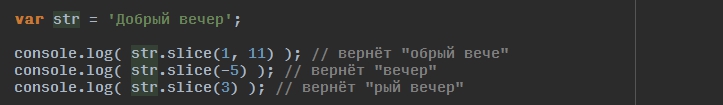
**.indexOf(text[, start])** - возвращает позицию первого вхождения указанного текста в строке (>=0) или -1, если ничего не найдено. Ищет с начала строки. Если стартовая позиция не указана, поиск идет с первого символа.



**.replace(text1, text2)** - возвращает строку, в которой произведена замена первого аргумента на второй.



**.slice(begin[, end])** – вырезает часть строки между позициями begin и end. Отрицательное значение будет считаться от конца строки.



**.split([separator[, limit]]) –** разбивает строку по разделителю. Возвращает массив строк.

http://dl3.joxi.net/drive/2017/05/26/0004/3207/285831/31/01be62153b.jpg

**.toLowerCase() и .** **toUpperCase() –** преобразует строку в нижний или верхний регистр.



**Задание №9**

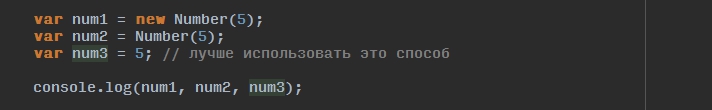
Пользуясь методами и свойствами из предыдущих уроков сделайте первую букву в строке «ann» заглавной.

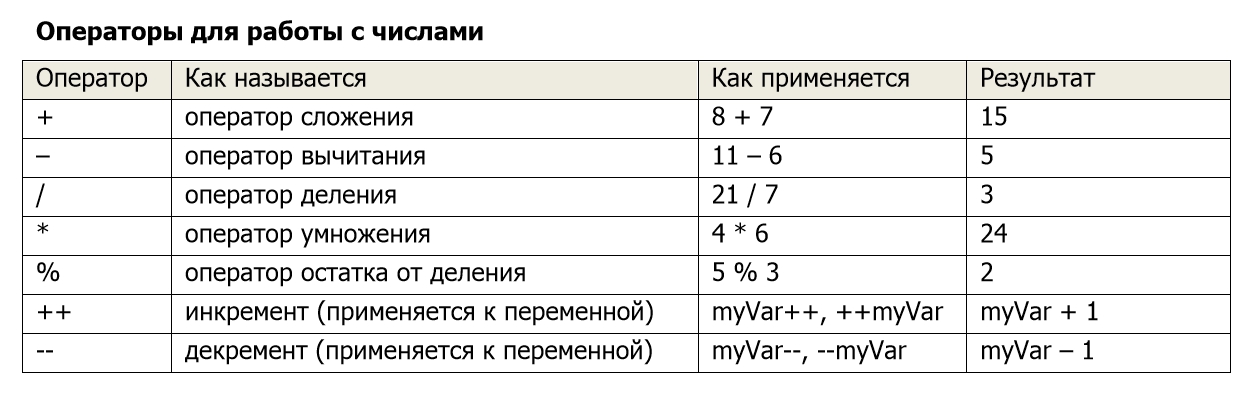
**Задание №10**

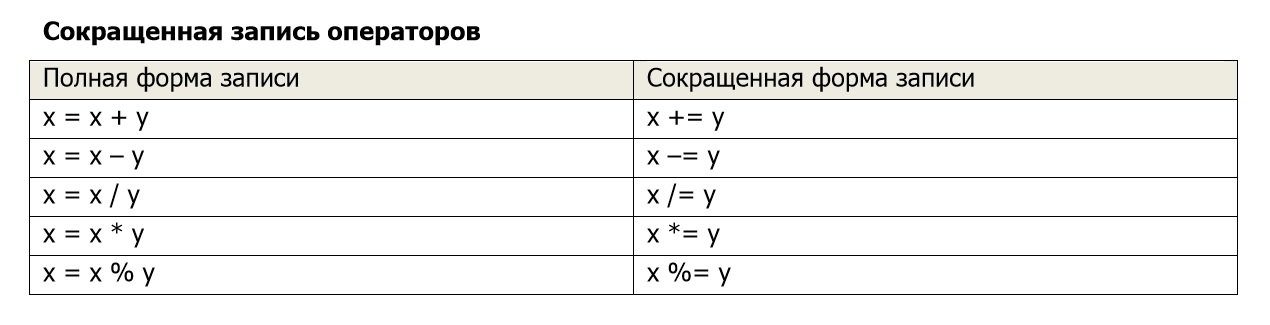
Из строки в переменной «str» нужно вырезать спам. Спамом считается строка «XXX» в любом регистре.

**Задание №11**

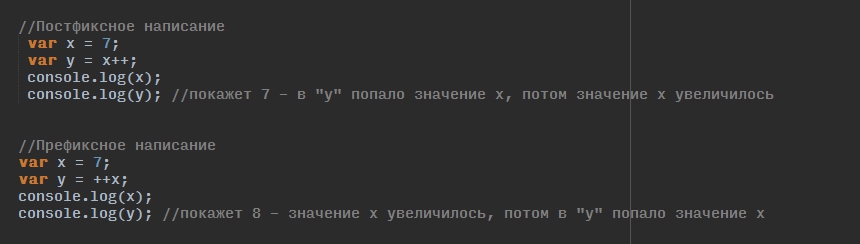
Аналогично строкам, числа в JS можно создавать различными способами. Создайте числа различными способами и выведите получившиеся переменные в консоль браузера:







Все приведённые в таблице выше операторы должны быть Вам знакомы со школы. Интерес представляют операторы инкремента (++) и декремента (--). В зависимости от места указания инкремента и декремента в инструкции, результат выполнения именно этой инструкции может быть различным. Разберем пример:



Из примеров видно, что по отношению к операнду инкремент и декремент работают одинаково независимо от формы записи. Разница видна в результатах операции присвоения.

Знак “–” (минус), записанный перед числом, делает его отрицательным (смена знака). Знак “+” (минус), записанный перед именем или значением переменной, пытается превратить значение переменной в число.

**Задание №12**

В этом задании мы проведём обзор основных методов типа данных Number. Работаем по простой схеме: сначала идёт описание работы метода, и за тем демонстрационный пример. Вы набираете демо примеры и убеждаетесь в их работоспособности и так далее.

**.toFixed( [n] )** - форматирует число, используя запись с фиксированной запятой, где n - необязательный параметр. Количество цифр после десятичной запятой; может быть значением между 0 и 20 включительно, хотя реализации могут поддерживать и больший диапазон значений. Если аргумент опущен, он считается равным 0.

http://dl3.joxi.net/drive/2017/05/29/0004/3207/285831/31/1400565c81.jpg

**.toPrecision( [точность] )** - возвращает строку, представляющую объект Number с указанной точностью.

http://dl3.joxi.net/drive/2017/05/29/0004/3207/285831/31/d2d6c70c62.jpg

**Задание №13**

Используя JS сделайте расчет сколько секунд в одной неделе и выведите результат в консоль.

**Задание №14**

Используя JS сделайте расчет сколько часов прошло, начиная с года Вашего рождения (без учета високосных годов, то есть принять год равным 365 дней).

**Задание №15**

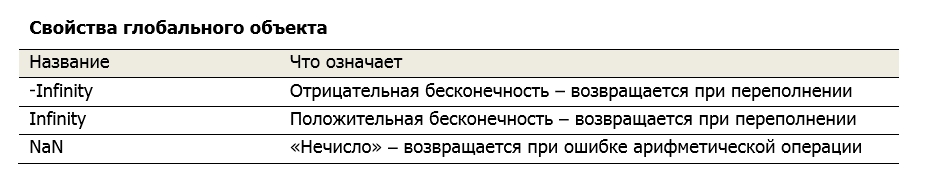
Дано значение температуры в градусах Фаренгейта TF = 100, найдите значение температуры в градусах Цельсия с помощью JS, если они связаны следующим соотношением:

http://dl3.joxi.net/drive/2017/05/29/0004/3207/285831/31/db0824c3aa.jpg

Выведите получившиеся значение в консоль.

**Задание №16**

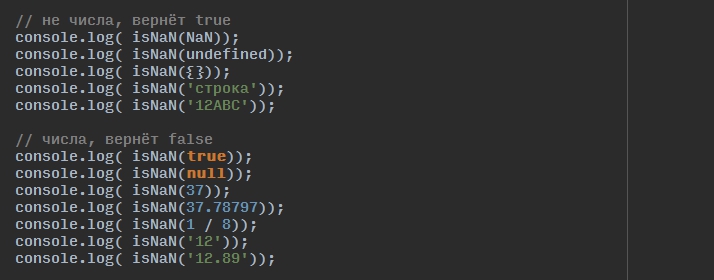
Глобальный объект JS обладает следующими свойствами для работы с числами:



Уникальность NaN – это значение не равно ничему, даже самому себе. Для проверки «нечисла» используется функция глобального объекта isNaN().

В этом задании мы проведём обзор основных методов глобального объекта для работы с числами. Работаем по простой схеме: сначала идёт описание работы метода, и за тем демонстрационный пример. Вы набираете демо примеры и убеждаетесь в их работоспособности и так далее.

**isNaN()** - определяет является ли литерал или переменная не числовым значением. По сути это проверка значения на «не число».



**parseFloat(string)** -если строка начинается с цифр, извлекает вещественное число и возвращает его, иначе возвращает NaN.

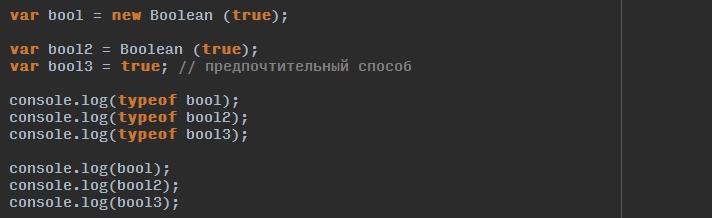
http://dl3.joxi.net/drive/2017/05/29/0004/3207/285831/31/be166ffb93.jpg

**parseInt(string)** – тоже самое что parseFloat, только возвращает целые числа.

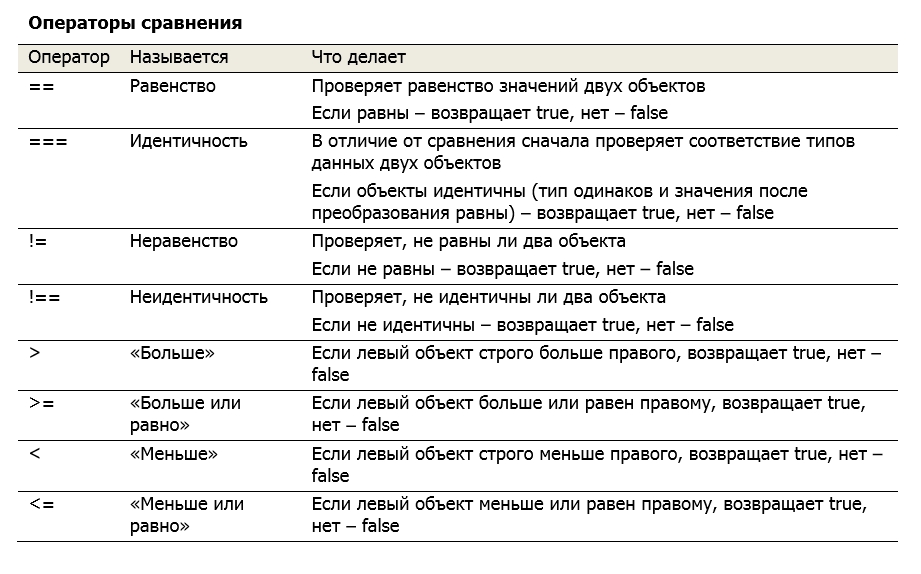
http://dl3.joxi.net/drive/2017/05/29/0004/3207/285831/31/b293260bba.jpg

**Задание №17**

Экземпляры логического типа данных Boolean имеют два возможных значения – true и false. Этот тип данных используется там, где есть проверка на соответствие условию. JavaScript легко преобразовывает типы данных из одного в другой, что следует учитывать в таких операциях.



Операторы сравнения возвращают своим результатом значение логического типа.



Наберите и протестируйте несколько примеров операторов сравнения:



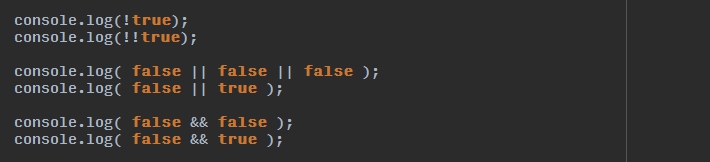
**Логические операторы**

**! -** логическое «НЕ». Возвращает обратное значение (инвертирует его).

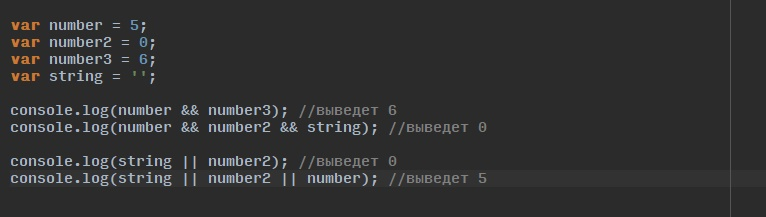
**&& -** логическое «И». Пытается преобразовать значение каждого операнда в логическое значение. Прекращает работу, когда значение операнда преобразуется в false. Возвращает значение последнего операнда или того, на котором прекратилось выполнение оператора. Перебирает все операнды, пока не встретится значение, которое нельзя преобразовать в true.

**|| -** логическое «ИЛИ». Пытается превратить значение каждого операнда в логическое значение. Возвращает значение первого операнда, значение которого удалось преобразовать в true или значение последнего операнда. Перебирает все операнды, пока не встретится значение, которое нельзя преобразовать в true.

Наберите и протестируйте несколько примеров логических операторов (с булевым типом):



Несколько примеров с не булевыми типами:



**Задание №18**

Преобразования значений в JavaScript производятся очень часто.

Следует понимать, что при таких фоновых преобразованиях значение, хранящееся в переменной, не меняется – создается временное значение с типом, который нужен для спешного выполнения операции. Например, сравниваются строка и число – эти значения приводятся к одному типу данных, сравнение возвращает логическое значение, преобразованные временные значения удаляются. Все преобразования зависят от контекста – операторов, которые выполняют над ними действия.

Ниже представлена таблица преобразования в различные контексты:

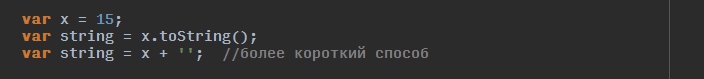


Существует несколько интересный методов преобразования типов в JS. Например, из любого типа данных можно получить логический тип данных с помощью двойного отрицания:



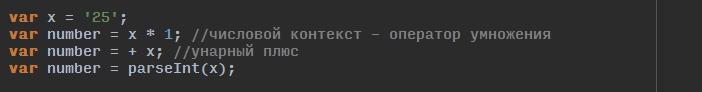
Выполните преобразование значений переменных от b1 до b8 в bool тип и выведите значения в консоль.

Получить строку из любого объекта можно двумя способами:



Выполните преобразование значений переменных от s1 до s8 в строковый тип и выведите значения в консоль.

Получение числа из строкового представления в 10-й системе счисления:



Выполните преобразование значений переменных от n1 до n8 в числовой тип и выведите значения в консоль.

**Задание №19**

Очень часто JS автоматически производит нужные преобразования типов, рассмотрим и протестируем несколько примеров. Все строки ниже будут преобразованы в числа на "лету" (метод isNaN вернёт false) проверьте это:

