Методичка

JavaScript.Основы

[Урок 1 Введение 3](#_Toc19986499)

[Типы данных 4](#_Toc19986500)

[Переменные 7](#_Toc19986501)

[Практические задания 9](#_Toc19986502)

[Домашнее задание 9](#_Toc19986503)

[Урок 2 Строки и условия 9](#_Toc19986504)

[Сравнение чисел 9](#_Toc19986505)

[Строки 10](#_Toc19986506)

[Получение отдельного символа строки 11](#_Toc19986507)

[Получение части строки 11](#_Toc19986508)

[Перевод строки в заглавный или строчный регистр 12](#_Toc19986509)

[Ввод и вывод данных 12](#_Toc19986510)

[Условные конструкции 13](#_Toc19986511)

[Конструкция if 13](#_Toc19986512)

[Конструкция if... else 13](#_Toc19986513)

[Цепочка if…else…else if 14](#_Toc19986514)

[Домашнее задание 14](#_Toc19986515)

[Урок 3 Массивы 15](#_Toc19986516)

[Доступ к элементам массива 16](#_Toc19986517)

[Создание и изменение элементов 16](#_Toc19986518)

[Разные типы данных в одном массиве 18](#_Toc19986519)

[Добавление элементов в массив 18](#_Toc19986520)

[Удаление элементов массива 19](#_Toc19986521)

[Объединение массивов 19](#_Toc19986522)

[Поиск индекса элемента в массиве 20](#_Toc19986523)

[Домашнее задание – Начать на занятии 20](#_Toc19986524)

[Урок 4 Массивы и циклы 21](#_Toc19986525)

[Превращаем массив в строку 21](#_Toc19986526)

[Доработка домашнего задания 21](#_Toc19986527)

[Циклы 22](#_Toc19986528)

[Цикл while 22](#_Toc19986529)

[Цикл for 23](#_Toc19986530)

[Циклы и массивы 23](#_Toc19986531)

[Практическая работа – Таблица умножения 24](#_Toc19986532)

[Домашнее задание 26](#_Toc19986533)

[Урок 5 Случайные числа 27](#_Toc19986534)

[Работа над домашним заданием 27](#_Toc19986535)

[Случайные числа 27](#_Toc19986536)

[Случайные индексы 29](#_Toc19986537)

[Программа случайного выбора вариантов 30](#_Toc19986538)

[Практическая работа 31](#_Toc19986539)

[Генератор случайных дразнилок 31](#_Toc19986540)

[Домашнее задание 31](#_Toc19986541)

# Урок 1 Введение

JavaScript – это язык программирования, с помощью которого можно управлять внешним видом странички, отслеживать движения мыши, нажатие на клавиши и другие действия пользователя, а также менять дизайн, цвет или положение любых объектов, в зависимости от этих самых действий.

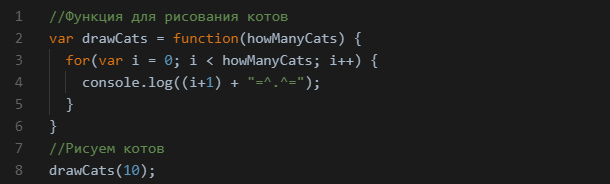
JavaScript используется почти на любом сайте. Он облегчает работу с почтой, комментариями, отправкой сообщений, прокручиванием страницы и ещё много с чем.

Чтобы убедиться насколько мощный язык программирования – JavaScript и посмотреть, что на нём можно сделать, давайте откроем браузер и перейдём по вот этой ссылке cubeslam.com – это игра, написанная полностью на JavaScript.

JavaScript далеко не единственный язык программирования. Всего насчитывают несколько сотен различных языков программирования, но почему тогда мы будем изучать именно его? На самом деле, ответ очень прост. Во-первых, JavaScript очень легко изучать, особенно если хотя бы чуть-чуть знаешь английский, а, во-вторых, чтобы писать и выполнять JavaScript-программы нам подойдёт любой браузер (Internet Explorer, Opera, Chrome, Yandex Browser или любой другой). Все программы, которые мы напишем с помощью JavaScript, мы легко сможем сразу же протестировать в своём браузере.

Сейчас мы с вами это и попробуем. Открывайте браузер, если у вас он не был открыт до этого. При зажатых клавишах CTRL+SHIFT нажмём на клавишу J, чтобы открыть секретный инструмент, называемый консолью. С помощью консоли мы можем выполнять небольшие программы на любой страничке.

Давайте решим простой пример: напишем в консоли *3 +4* и нажмём ENTER (Этой клавишей мы будем запускать наш код). Вы видите, что мы получили ответ *7* на следующей строке. Но JavaScript это нечто большее, чем какой-то калькулятор, вообще говоря, с помощью JavaScript можно написать свой калькулятор. Но для этого нам нужно **понять,** как работает JavaScript.

****Сначала давайте разберемся, почему JavaScript называют языком: во-первых, он, как и любой другой язык (Русский, Английский или Немецкий) имеет правила: орфография и пунктуация; а, во-вторых, его называют языком, потому что с помощью него мы будем общаться с компьютером. Компьютер – это просто машина, которая ничего не будет делать, пока ей правильно не скажут. Сейчас мы с вами научимся правильно общаться с компьютером на языке JavaScript, потому что на русском он, к сожалению, не понимает.

Чтобы написать несколько строчек в консоли, нужно в конце строчки нажать SHIFT+ENTER, вместо просто ENTER, тогда написанная строчка не будет выполняться, а под ней появится новая пустая строчка, в которой мы продолжим писать.

В этой маленькой программке встречается много разных символов, таких как скобки (), точки с запятой ; , фигурные скобки {}, знак плюс +, а также некоторые непонятные слова, например, *var* и *console.log*. Всё это и есть те самые правила языка JavaScript, которые называют синтаксисом.

Сразу пугаться не надо, вы всё это со временем запомните, если будете усердно заниматься программированием на JavaScript.

В первой строке нашей программы написано:

*//Функция для рисования котов*

Это называется *комментарий*. Программисты пишут комментарии в своих программах, чтобы не забыть то, что они задумали, и чтоб другим программистам легче было читать и понимать их код. В Javacript комментарии начинаются с двух символов наклонной. Всё что вы напишите после них в этой же строке, компьютер воспримет как комментарий и не будет выполнять его как обычный код. Комментарий никак не влияет на саму программу, поэтому советую вам оставлять комментарии в своих будущих программах, чтоб, когда вы откроете их спустя какое-то время, вы могли легко вспомнить, как работает код и что происходит в том или ином кусочке кода.

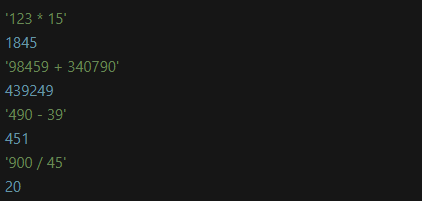
## Типы данных

Что такое данные? Данные – это информация, которая хранится в памяти компьютера. Это может быть ваше имя, фамилия, возраст, количество котов, которых нужно нарисовать, как в прошлой программе – всё это данные.

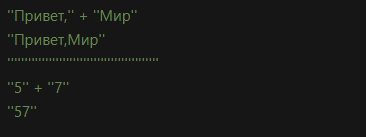
В JavaScript есть три основных типа данных: числа, строки и булевы значения. С числами всё просто, мы уже пробовали их складывать, умножать, тут всё понятно.

Любые текстовые данные записываются в строки, как мы делали с нашими котами: “=^.^=”. Точно также можно записать любой текст, обернув его предварительно в кавычки: “Какой-то текст”.

Булевы значения тоже очень простые, они могут хранить только два значения: *true*(“правда”) или *false(“ложь”).* С помощью булевых значений можно отвечать на вопросы, требующие ответа «да» или «нет».

**** Но нужно понимать, что с разными типами данных нужно обращаться по-разному. Например, перемножить два числа можно, а две строки – нет.Давайте попробуем решить несколько примерчиков с помощью JavaScript.

Как я уже говорил ранее – перемножить две строчки нам не удастся, точно так же как разделить строчку на строчку или вычесть из одной строчки другую. Но зато строчки можно складывать между собой.



Как можно увидеть из примера: операция сложения строк – соединяет две строчки между собой. Поэтому, если мы хоти написать предложение, нам стоит помнить, что между словами должны стоять пробелы. Что касается чисел, то, если обернуть их в кавычки, JavaScript сразу начнёт думать, что это строка и даже не будет пытаться производить какие-либо математические действия с ними.

Итого: что мы можем делать с числами – мы знаем, что можно делать со строками – тоже научились. Давайте теперь рассмотрим наши булевы значения – что можно сделать с ними?

Мы воспринимаем с вами булевы значения как *true*(“правда”) или *false(“ложь”),* но компьютер видит их по-другому. Для него, правда – это 1, а ложь – 0. Поэтому, если мы будем пытаться складывать, вычитать, умножать или делить их друг на друга, то на самом деле, компьютер будет проводить все эти операции с нулём и единицей (только не забывайте, что на нуль делить нельзя).

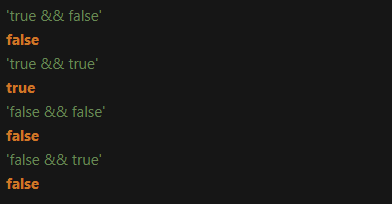
Зачем тогда нужны булевы значения, если компьютер ведёт себя с ними, как с обычными числами 0 и 1? На самом деле, они очень важны для принятия разных решений, и с ними можно проводить ещё кое-какие операции.

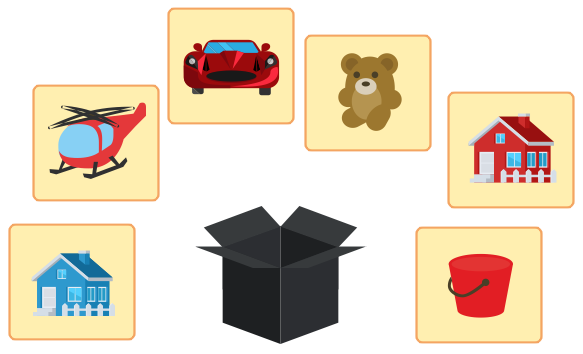
Первая операция называется ИЛИ и записывается как две рядом стоящие вертикальные черты || .



Поэксперементировав, мы с вами можем заметить, что результатом данной операции будет ложь (false), только тогда, когда мы спросим: «ложь ИЛИ ложь?», во всех остальных случаях у нас есть правда. А так как условие ИЛИ требует, чтоб была хотя бы одна правда, чтоб в итоге получить правду, во всех остальных случаях ответом является правда (true).

Вторая наша операция называется «И» и записывается && . Эта операция более строгая, и чтобы здесь получить правду, нужно чтоб оба высказывания были правдой. Во всех остальных наших случаях результатом будет ложь (false).

****

**Задача:**

Дед купил внуку игрушку. Он сказал, что у игрушки есть детали красного цвета, но нет окошек. Какую игрушку нужно положить в чёрную коробку?

*Построить с детьми таблицу типа:*

|  |  |  |  |
| --- | --- | --- | --- |
| **Название игрушки** | **Детали красного цвета** | **Нет окошек** | **Результат** |
| Синий дом | false | false | false |
| Вертолёт | true | false | false |

Оператор ! означает «не» — так его и называйте. Используйте этот оператор, чтобы превратить false в true или, наоборот, true в false. Это полезно для работы со значениями-противоположностями.

## Переменные

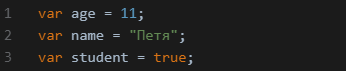
Значениям в JavaScript можно давать имена, используя переменные. Переменная похожа на ящик, в который можно положить только один предмет, будь то число, строка или булево значение. Но чтобы положить туда что-то ещё, с прежним содержимым придётся попрощаться.

Чтобы создать новую переменную, используйте ключевое слово **var**, после которого укажите имя переменной. Когда JavaScript встретит в вашем коде это ключевое слово – он сразу поймёт, что вы создали новую переменную, в которую хотите поместить какое-то значение. Давайте создадим переменную и дадим ей какое-нибудь имя.

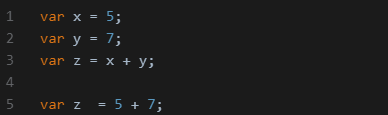
> **var** name;

> undefined

В ответ консоль нам выведет **undefined** что означет – «значение не определено». Но это не ошибка, так как мы на самом деле ещё не присвоили никакого значения нашей переменной. Давайте попробуем создать переменную и присвоить ей значение, то есть положить в нашу «переменную-коробку» то, что мы захотим.



Теперь, введя в консоль имя переменной, мы увидим её значение. Точно так же одной переменной можно присвоить значение другой, тем самым скопировав её. Переменной можно сразу присвоить значение какого-то выражения, или посчитать это выражение, используя другие переменные.



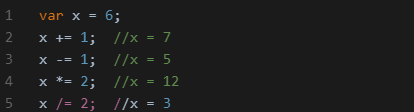
В обоих случаях значение переменной **z** будет равно 12. Давайте теперь каждый решит по задаче, например:

**Дано:** x = 9, у = 13, z = x\*6 + (y – 4) / 2 = ?

Если же вы не планируете решать сложный пример, а хотите просто увеличить или уменьшить значение какой-то существующей переменной, то это можно сделать так:



Всё это можно записать более коротко:



Вводя имена переменных, будьте внимательны и не допускайте опечаток. Даже если вы перепутаете строчные и заглавные буквы, интерпретатор JavaScript не поймёт, что вы от него хотите. Поэтому за названиями переменных нужно внимательно следить.

## Практические задания

1. Посчитать количество часов в неделе, месяце, году.
2. Выразить свой возраст в минутах, часах, днях.

## Домашнее задание

Задачки на логику.docx

Ответы:

1. Сначала перевезти на другой берег козу. Затем вернуться за капустой, привезти ее на берег к козе, поменять их местами (капусту оставить на берегу, а козу к себе в лодку), с козой в лодке отправиться на другой берег к волку. Высадить козу и взять на борт волка, отвезти волка к капусте на другой берег. Съездить и привезти козу.
2. День рождения у него 31 декабря, а фраза эта была сказана 1 января.
3. Сначала мама и папа – 2 минуты.

Папа возвращается - 3 минуты.

Дети идут к маме – 8 минут.

Мама возвращается к папе – 10 минут, и они вдвоем идут к детям – 12 минут.

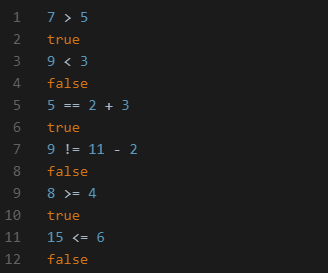
# Урок 2 Строки и условия

**Тест:** <http://igroflot.ru/logic/flash_game_3/big/>

## Сравнение чисел

На прошлом занятии мы с вами узнали, что такое булевы значения, но не всё попробовали с ними. В школе вы уже сравнивали числа и использовали различные операторы сравнения: больше, меньше, равно, больше или равно, меньше или равно.

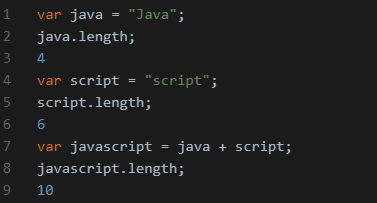
В JavaScript тоже часто используют операции сравнения, а так как ответить на пример-сравнение можно либо «да» либо «нет», то результатом операций сравнения в JavaScript будут булевы значения true или false.



## Строки

Как мы узнали в прошлом уроке – строки можно складывать. И результатом такого сложения получится новая строка, которая будет начинаться первой исходной строкой, а заканчиваться второй. Например, «Я люблю» + «JavaScript» = «Я люблюJavaScript». Так как во время сложения строк происходит их склеивание, то не стоит забывать про пробелы. «Я люблю » + «JavaScript» = «Я люблю JavaScript». Дополнительный пробел перед закрывающей кавычкой после слова «люблю» обеспечивает нам правильный вид нашего предложения.

 Но это не всё что можно делать со строками. Очень часто бывает нужным узнать длину строки, и, к нашему счастью, javacript умеет это делать. Чтобы узнать длину строки достаточно добавить к ее концу ***.length***

 Можно добавлять .length к концу как самой строки, так и переменной:

Здесь мы присвоили строковое значение "Java" переменной java,

а значение "Script" — переменной script. Затем мы добавили .length к концу каждой из переменных, узнав таким образом длины отдельных строк, а также длину составленной из них новой строки.

### Получение отдельного символа строки

Иногда требуется получить из строки одиночный символ. Например, вы можете зашифровать в наборе слов тайное послание, состоящее из вторых символов каждого слова. Тогда, чтобы узнать это послание, нужно получить все вторые символы и объединить их в новую строку. Чтобы получить символ, стоящий в определенной позиции строки, используйте квадратные скобки — []. Возьмите строку (или переменную, в которой хранится строка) и поставьте сразу после нее квадратные скобки, в которых указана позиция нужного символа. Например, чтобы получить первый символ строковой переменной myName, используйте запись myName[0]:



Обратите внимание — чтобы получить первый символ, мы указали в скобках позицию 0, а не 1. Дело в том, что JavaScript (как и многие другие языки программирования) ведет отсчет символов с нуля. Таким образом, для получения первого символа строки указывайте позицию 0, второго — 1 и т. д. Попробуем разгадать наш тайный шифр, где во-вторых буквах некоторого набора слов скрыто послание. Вот как это сделать:



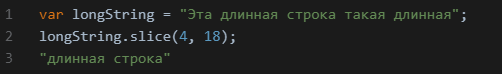
### Получение части строки

Чтобы получить часть, или «срез», строки, используйте slice. Например, представьте, что вам нужен отрывок из длинного описания фильма для анонса на вашем сайте. Чтобы воспользоваться slice, поставьте в конце строки (или переменной, содержащей строку) точку, а после нее слово

slice и круглые скобки. В скобках укажите позицию первого символа

той части строки, которую вы хотите получить, затем запятую, а затем

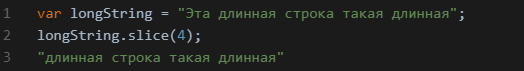
позицию последнего символа.

Например:

Первое число в скобках — позиция символа, с которого начинается срез, а второе число — позиция символа, который *следует за*последним

символом среза.

По сути, мы попросили JavaScript: «Вырежи из этой длинной строки часть, которая начинается с символа в позиции 4 и продолжается до позиции 18».

Если указать в скобках после slice только одно число, мы получим строку-срез, которая начинается с данной позиции и длится до конца строки:

### Перевод строки в заглавный или строчный регистр

Если нужно вывести какой-нибудь текст заглавными буквами, воспользуйтесь toUpperCase.

"Эй, как дела?".toUpperCase();

"ЭЙ, КАК ДЕЛА?"

Можно произвести и обратную операцию, использовав toLowerCase:

"эЙ, кАК деЛа?".toLowerCase();

"эй, как дела?"

**Задание:** Преобразовать строку «эЙ, кАК деЛа?» в «Эй, как дела?».

## Ввод и вывод данных

Консоль, которую мы с вами используем – это инструмент для разработчиков, и обычные пользователи, которые будут использовать наши программы, его не касаются. Для того, чтобы пользователь мог взаимодействовать с нашей программой, в JavaScript существуют диалоговые окна.

Первое диалоговое окно, с которым мы познакомимся – это alert()**.**

alert() – эта функция, которая вызывает на странице диалоговое окно с кнопкой ОК. С помощью этой функции вы можете сообщать пользователю какую-либо информацию.

 Второе диалоговое окно – это вопрос пользователю. prompt() задаёт пользователю вопрос и предоставляет текстовое поле, в которое тот может ввести свой ответ. В отличии от alert(), prompt() может сохранить ответ в переменную:

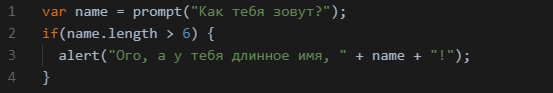
## Условные конструкции

В JavaScript есть два вида условных конструкций — это **if** и **if...else.** Оператор **if** выполняет фрагмент кода, если какое-то условие истинно (true). Например: *если* вы хорошо себя вели, получите конфетку.

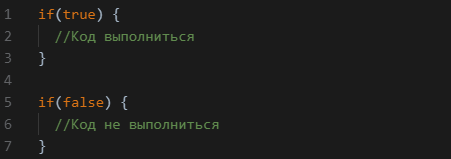
А оператор if... else выполняет один фрагмент кода, если условие дает true, и другой фрагмент в противном случае. Например: *если* вы хорошо себя вели, получите конфетку, *иначе* вас не отпустят гулять.

### Конструкция if

Самая простая из управляющих конструкций JavaScript — это if. Она

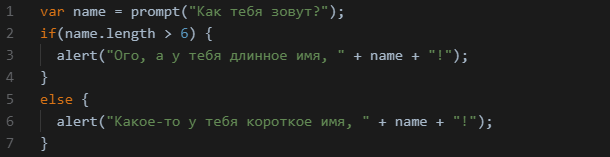
 используется, чтобы запускать код, если некое условие истинно (true).

То есть, для того чтобы кусок кода заключённый между фигурными скобками после if, сработал, нужно, чтоб в круглых скобках после if (в этих скобках записывается условие), было правдивое высказывание.

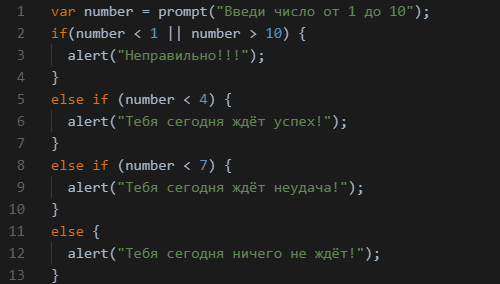


### Конструкция if... else

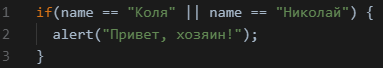
Как я уже говорил, оператор if запускает код своего тела, только если

условие дает true. Но если вы хотите, чтобы по условию false тоже что-то происходило, вам нужна конструкция if... else. Давайте дополним предыдущий пример:

### Цепочка if…else…else if

Чтобы создать цепочку if... else, начните с обычного оператора if и после закрывающей фигурной скобки его тела введите ключевые слова else if, а следом — еще одно условие и еще одно тело. После можно добавить еще else if, и так до тех пор, пока у вас не закончатся условия (которых может быть сколько угодно). Завершающая секция else будет выполнена, если ни одно из условий не дает true.

## Домашнее задание

Напишите программу, в которой вы будете спрашивать у пользователя имя. Если это ваше имя, то компьютер будет отвечать: «Привет, хозяин!». Если это имя папы, то «Привет, папа!» и так далее. Не забывайте, что имя может быть, как полное, так и сокращённое (Николай = Коля).

# Урок 3 ****Массивы****

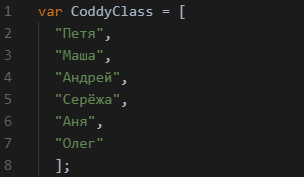
Мы уже изучили числа и строки — типы данных, которые можно хранить и использовать в своих программах. Но одни лишь числа и строки — это как-то скучновато; не столь уж многое можно сделать со строкой как таковой. С помощью *массивов* JavaScript позволяет создавать и группировать данные более любопытными способами. А по сути своей массив — всего лишь список, где хранятся другие значения. Например, если вашему другу интересно, какие три игры вам нравятся больше всего, вы можете создать массив и расположить там по порядку названия этих игр:

Теперь вместо того, чтобы показывать своему другу три отдельные строки, вы можете воспользоваться единственным массивом games.

Зачем нужны массивы? Предположим вы решили записать все игры, которые вы знаете. Вы можете создать для каждой игры новую переменную, но это очень неудобно и будет выглядеть громоздко, особенно если вы знаете очень много игр. Создать 1000 переменных будет не так уж легко. Вот для этого и нужны массивы.

Чтобы создать массив со значениями, нужно перечислить эти значения внутри квадратных скобок, разделяя их запятыми. Отдельные значения, хранящиеся в массиве, называют *элементами*. В нашем примере все элементы будут строковыми (это наши имена), поэтому запишем их в кавычках. Создадим список нашего класса. Для этого создадим массив и заполним его именами.

Длинный список сложно читать, когда он записан одной строкой, но, к счастью, это не единственный способ форматирования кода при создании массива. Вы можете поставить открывающую квадратную скобку на одной строке, каждый элемент писать с новой строки и последней строкой поставить закрывающую квадратную скобку:



Для JavaScript неважно, отформатируете вы код, расположив весь массив на одной строке или на нескольких строках по частям. Сколько бы ни стояло переносов, JavaScript увидит один и тот же массив — в нашем случае состоящий из шести строк.

## Доступ к элементам массива

Чтобы получить доступ к элементам массива, используйте квадратные скобки с *индексом* нужного вам элемента, например:

CoddyClass[0];

"Петя"

CoddyClass[3];

"Серёжа"

Индекс — это номер элемента, в котором хранится значение. Аналогично символам в строке, первому элементу массива соответствует индекс 0, второму — 1, третьему — 2 и т. д. Поэтому, запросив индекс 0 в массиве CoddyClass, мы получили "Петя" (это первый элемент), а запросив индекс 3 — "Серёжа" (четвертый элемент). Возможность доступа к отдельным элементам массива очень полезна. Например, если вы хотите показать кому-то ученика, сидящего за первой партой, ни к чему показывать весь массив. Вместо этого просто возьмите первый элемент.

## Создание и изменение элементов

Используя индекс в квадратных скобках, можно задавать или изменять значения элементов и даже добавлять новые элементы. Например, чтобы заменить содержимое первого элемента массива CoddeClass "Петя") на "Петя Иванов", можно написать:

А если вспомнить, что элементы нашего массива – строки, а строки мы можем склеивать между собой. Тогда добавить фамилию можно ещё одним, или даже двумя способами.

С помощью индексов также можно добавлять в массив элементы. Например, можно добавить меня в конец нашего массива. Но для того, чтобы это сделать, нам нужно знать какой сейчас последний элемент. Когда наш массив небольшой, мы можем посчитать или вывести его в консоли и посмотреть. Но что делать, если наш массив насчитывает около тысячи элементов или больше?

Чтобы не считать все элементы вручную, можно воспользоваться методом, который мы уже использовали – это метод вычисления длины строки – length. Так как строка – это набор различных символов, а массив – это тоже набор, только вместо символов здесь могут быть числа, строки, другие массивы и всё это вперемешку. Поэтому точно так же как мы узнавали длину строки, мы можем узнать длину массива. Для этого напишем название массива и в конце припишем .length.

 Теперь мы узнали, что в нашем массиве 6 элементов, но давайте вспомним, как нумеруются элементы в массиве. Да, они начинают нумероваться с нуля, а это значит, что последний элемент имеет номер 5. И если мы хотим добавить в конец ещё один элемент, то нам нужно создать элемент под номером 6:

 Теперь длина нашего массива – 7, а последний элемент имеет индекс 6. Иногда приходиться часто добавлять элементы, и чтоб не помнить, какой должен быть номер следующего элемента, можно сделать проще. Если внимательно посмотреть на связь длины массива и на номер последнего элемента (длина = 6 – последний индекс = 5; длина = 7 – последний индекс = 6), можно заметить, что индекс последнего элемента всегда на единицу меньше, чем длина массива. Это как раз из-за того, что мы начинаем нумеровать массив с 0. Таким образом индекс последнего элемента мы можем выразить как длина массива минус один. А так как в квадратных скобках массива мы можем писать не только числа, но и переменные и даже целые выражения, то, чтобы посмотреть последнего человека из нашего массива, мы можем написать так:

 Тогда, чтобы добавить ещё один элемент в конец массива, нужно написать:

## Разные типы данных в одном массиве

Не обязательно, чтобы все элементы массива были одного типа. Например,

давайте создадим массив, в который запишем возраст каждого из нас.

 А теперь добавим этот массив в тот, который мы создали ранее:

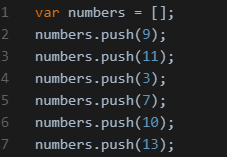
Теперь, чтобы проверить, что у нас получилось, давайте введём в консоли название нашего первого массива, нажмём ENTER и посмотрим, как выглядит наш массив. Да, теперь в последней ячейке нашего массива ещё один массив.

Чтобы обратиться к элементам массива, вложенного в другой массив, нужно использовать вторую пару квадратных скобок. Например, если команда CoddyClass[8]; вернет весь вложенный массив, то CoddyClass[8][0]; — лишь первый элемент этого вложенного массива (10).

## Добавление элементов в массив

Чтобы добавить элемент к концу массива, можно воспользоваться методом push. Введите .push после имени массива, а после в круглых скобках укажите элемент, который нужно добавить:

Push — буквально «протолкнуть, добавить»

Этим методом можно создавать массив с нуля. Давайте создадим пустой массив numbers, и, с помощью метода push(), добавим в него какие-нибудь числа:

А теперь можно засунуть этот массив в конец массива CoddyClass, тоже используя метод push(). С помощью этого метода, гораздо проще добавлять элементы в конец массива. Но теперь представьте, что нам нужно добавить элемент в начало. Сделать это, не используя никаких методов, очень сложно, потому что нужно будет пододвинуть каждый последующий элемент на единичку. Но, так как у нас есть методы для работы с массивами, мы это легко сделаем.

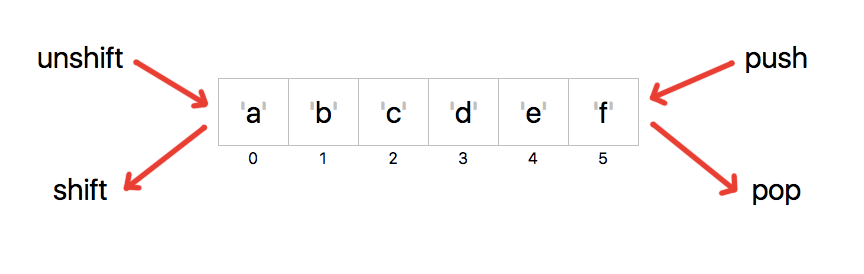
Чтобы добавить элемент в начало массива, используйте метод .unshift(элемент). Давайте добавим ещё числа в наш массив, который теперь находиться внутри массива CoddyClass. Задача не из лёгких. (*Пусть дети попробуют справиться самостоятельно*)

Помимо того, что метод unshift() добавляет в начало массива элемент, он также как и метод push() изменяет длину массива при каждом добавлении.

## Удаление элементов массива

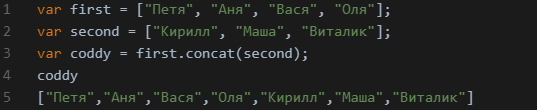
Убрать из массива последний элемент можно, добавив к его имени .pop(). Метод pop делает сразу два дела: удаляет последний элемент из массива и возвращает этот элемент в виде значения.

Таким образом мы удалили массив наших чисел из массива Coddyclass и сохранили его в numb.

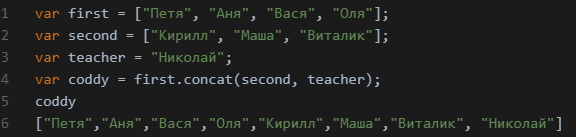
 Чтобы удалить из массива первый элемент, вернув его значение, используйте .shift(). Работает он точно так же как и pop(), только удаляет первый элемент.

## Объединение массивов

Чтобы «склеить» два массива, создав таким образом новый массив, используйте команду firstArray.**concat**(secondArray). Метод concat создает массив, в котором элементы из firstArray будут расположены перед элементами из secondArray.

 Давайте создадим два новых массива: массив сидящих на первом ряду и на втором. А потом соединим их в новый массив coddy.

С помощью concat можно объединить больше чем два массива. Для этого укажите дополнительные массивы в скобках, разделив их запятыми:



Метод concat удобен, когда нужно объединить несколько массивов в один. Скажем, пусть у вас есть список любимых книг и у вашего друга тоже есть свой список, и вы решили выяснить, можно ли купить все эти книги в магазине по соседству. Тогда будет проще, если у вас будет не два списка, а один. Просто объедините ваш список со списком вашего друга методом concat, и пожалуйста — у вас один список на двоих!

## Поиск индекса элемента в массиве

Чтобы выяснить, какой у определенного элемента индекс в массиве, используйте .indexOf("элемент"). Попробуйте, посмотрите каждый свой индекс.

Если элемента, индекс которого вы запрашиваете, в массиве нет, JavaScript вернет значение −1.

## Домашнее задание – Начать на занятии

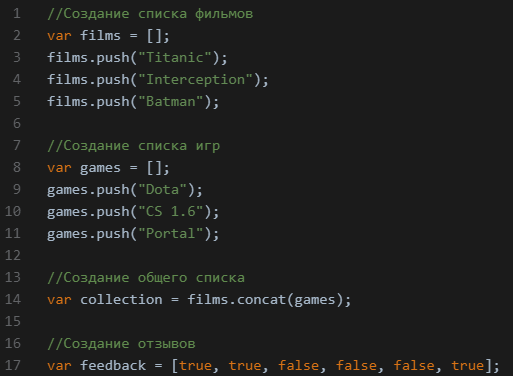
Написать программу, которая будет содержать список ваших любимых фильмов и игр.

1. Создать два массива: фильмы и игры. Заполнить их с помощью метода push().
2. Объединить два массива.
3. Создать массив с отзывами YesOrNo, длина которого будет равна количеству фильмов и игр вместе взятых.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| YesOrNo | true | false | false | true | false | true | true | true |

*Пример массива с отзывами*

1. True и false в этом массиве означают стоит смотреть/играть этот фильм//игру. Номер ячейки совпадает с номером фильма/игры в исходном массиве.



# Урок 4 ****Массивы и циклы****

## Превращаем массив в строку

Воспользовавшись методом .join(), можно соединить все элементы массива в одну большую строку.

Метод join возвращает строку, в которой через запятую перечислены все элементы массива. Но что если мы не хотим использовать в качестве разделителя запятую?

Нам поможет метод .join ("разделитель"), который делает все то же самое, но вместо запятых ставит между элементами выбранный разделитель. Давайте попробуем три разных разделителя: дефис с пробелами по сторонам, звездочку \* и союз «и» с пробелами по сторонам. Обратите внимание: разделитель нужно записывать в кавычках — ведь это строка.

## Доработка домашнего задания

1. При запуске программы пользователю выводится список всех фильмов и игр (с помощью .join()), а после вопрос: «Какой фильм хочешь посмотреть или в какую игру хочешь поиграть?».
2. Пользователь вводит название, если игра/фильм хорошая (в соответствующей ячейки стоит true) вывести сообщение: «Да, это годная штука».
3. Иначе, вывести: «Нет, не советую».

*Пример программы в файле* **feedback.js**

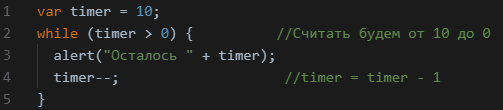
## Циклы

Во время программирования часто возникают ситуации, когда необходимо повторить некоторое действие несколько раз. Если, уже пройденные нами, условные конструкции позволяют запускать часть кода, если условие даёт *true*. То, то с чем мы сегодня познакомимся, циклы – позволяют выполнять кусок кода многократно, до тех пор, пока, заданное нами условие, не перестанет выполняться. Например, мыть руки, пока они не станут чистыми. Здесь «мыть руки» – это часть кода, которую мы хотим выполнить несколько раз, а «не чистые руки» – условие. Пока это условие верно, наш код будет выполняться.

### Цикл while

Самый простой из циклов — цикл while. Этот цикл снова и снова выполняет код своего тела, до тех пор, пока заданное условие не перестанет давать true. Используя цикл **while**, мы имеем в виду следующее: «Продолжай делать это, пока условие дает **true**. Но если оно даст **false**, остановись».

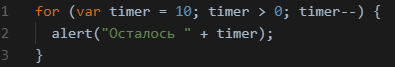
Давайте попробуем сделать обратный отсчёт с помощью цикла while.

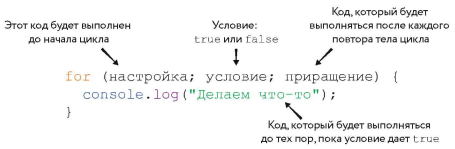


Давайте разберём, что же мы делаем. Сначала мы создаём таймер, который будет равен 10. Дальше мы в цикле проверяем условие: таймер больше чем 0. Да, так как таймер равен 10. Дальше мы выводим сообщение склеивая строчку «Осталось »(не забывая пробел) и значение нашего таймера. После этого мы уменьшаем значение нашего таймера на 1, теперь timer = 9. А теперь, главное отличие условных конструкций от циклов, главное отличие if от while: после того как наше условие оказалось верным, и нужная нам часть кода выполнилась, while не продолжает работу программы, как сделал бы if, вместо этого, он ещё раз проверяет условие. И если оно будет верным и в этот раз, то он снова выполнит этот код. И так до тех пор, пока наше условие не станет ложным.

Главное в работе с циклами не забывать, что если условие в скобках никогда не сможет стать false, то вы получите бесконечный цикл, остановить который можно будет только перезагрузив страницу.

### Цикл for

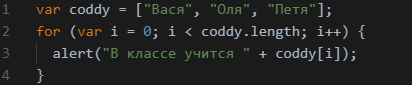
В данном примере переменную timer называют счётчиком, так как она отсчитывает количество раз, которые проработает цикл. Для более удобной работы с циклами, в которых нужен счётчик, придумали ещё один цикл – for.

 Вот так выглядит точно такой же скрипт, только написанный с помощью цикла for.

Циклы for применяют, когда нужно сделать что-то определённое количество раз, или, когда в процессе выполнение понадобиться счётчик. Циклы while используют, когда нужно что-то выполнить до тех пор, пока не выполниться какое-то условие.

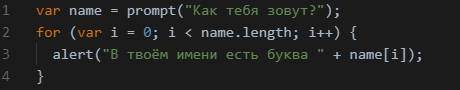
## Циклы и массивы

Очень часто цикл for используют для перебора всех элементов массива

или всех символов строки. Например, давайте создадим ещё раз массив нашего класса и напечатаем всех:

В этом цикле i сначала равняется 0, а затем возрастает до значения

coddy.length - 1, то есть 2. Числа 0, 1 и 2 — индексы элементов в массиве coddy. Это значит, что при каждом повторе цикла i принимает значение очередного индекса, а coddy [i] соответствует очередному животному из массива coddy. Когда в i число 0, coddy [i] выдаст нам строку "Вася". Когда в i число 1, coddy [i] выдаст "Оля" и т. д.

Как мы уже знаем, к отдельным символам строки можно обращаться тем же способом, что и к элементам массива, — с помощью квадратных скобок. В следующем примере мы используем цикл for для вывода символов имени:

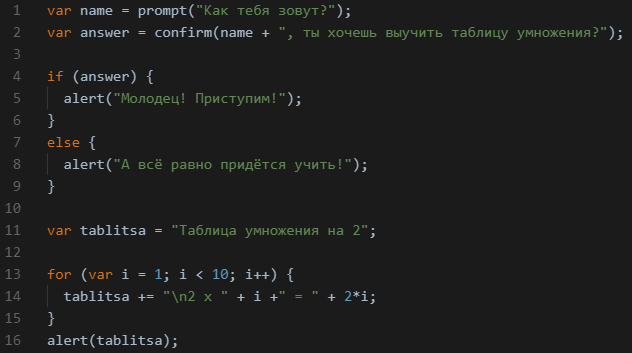
## Практическая работа – Таблица умножения

*Дети должны максимально самостоятельно выполнять задание*

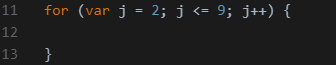
Для того, чтобы потренироваться работать с циклами и массивами, а также вспомнить ввод и вывод сообщений, давайте напишем такую программу:

1. Спрашиваем у пользователя его имя.
2. Задаём «безвариантный» вопрос: «Хочешь выучить таблицу умножения?»

Чтобы задать вопрос, на который можно ответить «Да» или «Нет», нужно воспользоваться функцией confirm(). Она работает точно также, как и prompt, только не выводит поле для ответа, а имеет только две кнопки. При нажатии на кнопку «ОК» возвращает true, иначе false.

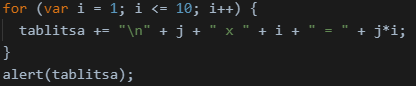
1. Если нажимает «Да» - пишем «Хорошо», иначе – «Всё равно надо!»
2. Создаём строку, в которую циклом заносим примеры умножения на 2 (каждый пример в новой строке с помощью “\n” – перевода на новую строчку)
3. Выводим полученную строчку.

Ну а что, если мы хотим выучить всю таблицу умножения? Как нам это сделать? Оказывается, мы можем вставлять цикл в цикл, и делать так можно хоть до бесконечности.

Для того чтобы сделать таблицу умножения от 2 до 9, нам нужно создать ещё один цикл for, в котором счётчик будет изменятся от 2 до 9.

А строчки, которые раньше находились с 11 по 16, нужно вставить между фигурными скобками нашего цикла.

Остались небольшие изменения и наша программа готова. Теперь мы не можем создавать строчку tablitsa = “Таблица умножения на 2”, так как у нас она сначала будет на 2, потом на 3 и так далее. Теперь мы должны создавать её с использованием нашего счётчика j.

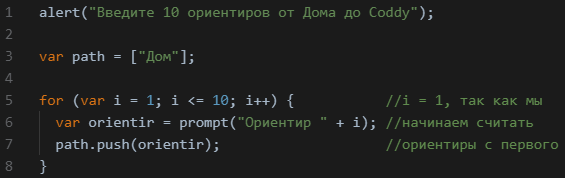
Также изменения претерпит и строчка 15 в нашей программе. В ней мы к строке tablitsa прибавляем пример, но в начале примера пишем цифру 2, так ещё и результатом является 2\*i. Это выражение нужно исправить таким образом, чтоб оно не зависело от цифр, а только от переменных.

*Полный код скрипта в файле* **multTable.js**

Следующее задание, которое мы с вами сейчас начнём, останется вам в виде домашнего задания. Нам с вами предстоит написать Навигатор. С помощью этого навигатора можно будет легко добраться от вашего дома до Coddy и обратно. Схема работы нашей программы такова:

1. Пользователь поочерёдно вводит 10 ориентиров от Дома до Coddy.
2. Программа выводит путь от Дома до Coddy разделяя словом «потом»
3. Точно так же выводит обратный путь, который нужно перевернуть, используя метод pop (из этого массива) и push (в другой массив).

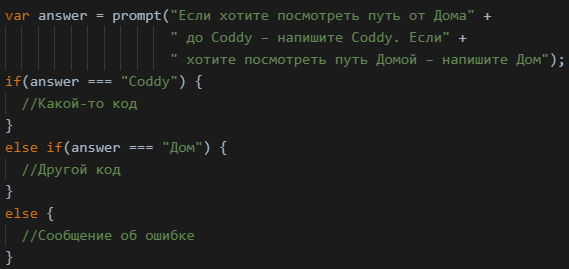
Для начала нам нужно вывести сообщение с информацией для пользователя:

После этого мы должны создать массив, в котором будем хранить наш путь, занести первый элемент – «Дом», а потом в цикле спрашивать у пользователя все ориентиры. *(Самостоятельное задание для детей)*

 Следующем шагом нужно добавить самый последний ориентир – Coddy.

 Осталось последнее – вывести путь от Дома до Coddy, используя метод join() и разделитель «потом»:

## Домашнее задание

1. Доработать программу таким образом, чтоб после того, как будут известны все ориентиры, пользователю задавали вопрос: «Если хотите посмотреть путь от Дома до Coddy – напишите Coddy. Если хотите посмотреть путь Домой – напишите Дом».
2. Соответственно, в зависимости от нажатия, выводить тот или иной путь.
3. Если введено что-то другое – выводить сообщение об ошибке.

*Для примера можно показать кусок, отвечающий за эту проверку*

*prompt в несколько строк, так как не помещался на картинку*

*Пример программы в файле* **navigator.js**

# Урок 5 ****Случайные числа****

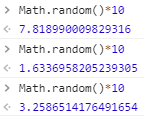
## Работа над домашним заданием

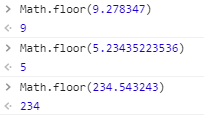
В 17-й строке расположено наше условие. Если ответ на наш вопрос – «Дом», то нам необходимо создать второй массив, который будет являться перевёрнутым первым. В 18-й строке мы создаём этот массив, а в 19-22 заполняем его элементами из первого массива. Мы забираем последний элемент из нашего изначального массива в переменную lastElement, а после добавляем этот элемент в конец второго массива.

## Случайные числа

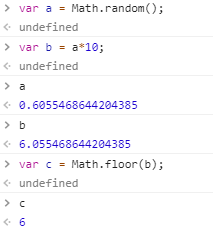
Многие из вас слышали о генераторах случайных чисел, но же такое случайное число и как его получить? На самом деле получить случайное число не так уж и легко. Человек не может сказать случайное число, потому что его выбор всегда с чем-то связан, на чём-то основан. Но JavaScript не человек, поэтому для него это не такая уж и сложная задача. Выбор случайного числа построен на разрешении некоторой математической функции. Для работы со всем, что связано с математикой у JavaScript есть библиотека Math. Чтобы получить необходимую функцию из этой библиотеки, нужно просто написать название – **Math** и через точку дописать название необходимого нам инструмента.

Первый метод из библиотеки Math, который мы рассмотрим – это метод для получения случайных чисел – **Math.random()**. Давайте разберёмся, как он работает, и посмотрим в консоли, что он возвращает при вызове. Если вызвать его несколько раз, сразу можно заметить, что он возвращает случайные числа от 0 до 1.

 Важно помнить, что Math.random() возвращает числа от 0 до 1, но не может вернуть 0 и 1. Но что, если нам нужно случайное число от 0 до 10, как быть в этом случае? На самом деле, сделать это очень легко, для этого нам нужно умножить полученное с помощью Math.random() значение на то число, до которого мы хотим выдавать случайные числа (в нашем случае 10).

 Научившись генерировать случайные числа до любого числа, мы всё равно не достигли того чего хотели, потому что наши числа не целые, и нам это не нравится. Но JavaScript и тут спешит нам на помощь. Оказывается, он умеет округлять числа с помощью метода **Math.floor()**. Этот метод отбрасывает все цифры после запятой, оставляя целое число, стоящее перед запятой.

 А поскольку мы можем объединять методы для получения необходимого нам результата, то давайте объединим два изученных метода из библиотеки Math.

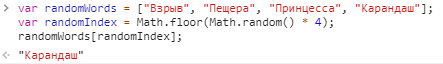
Давайте разберём этот пример. Сначала мы генерируем случайное число от 0 до 1 с помощью метода Math.random(). Потом мы умножаем полученное значение на 10, чтобы получить число от 0 до 10. И последнее наше действие – это округление полученного числа. Попробуйте сделать то же самое, но только поэтапно. Но для этого потребуется использовать переменные.

 Ещё один маленький нюанс, который стоит знать при работе со случайными числами, связан с возможностью сгенерировать случайные числа в некотором диапазоне. Например, вам нужно выбрать случайное число между 10 и 20. Так как изученная нами конструкция выдаёт числа от нуля, то для решения поставленной задачи, нужно всего лишь прибавить к полученному результату недостающие число.

В первом случае мы можем получить числа меньше 10, а во втором будут числа только из диапазона 10 – 20.

## Случайные индексы

Давайте используем оба метода, чтобы получить случайный индекс. Нужно лишь умножить *Math.random()* на длину массива и затем округлить полученное число методом *Math.floor()*. Например, если в массиве четыре элемента, это можно сделать так:

При каждом запуске этот код будет возвращать случайное число от 0 до 3 (включая 0 и 3). Поскольку *Math.random()* всегда возвращает значение меньше 1, *Math.random() \* 4* никогда не вернет 4 или большее число. Используя это случайное число как индекс, можно получить случайный элемент массива:

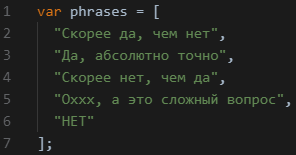
С помощью *Math.floor(Math.random() \* 4);* мы получили случайное число от 0 до 3. Сохранив это число в переменной *randomIndex*, мы использовали его как индекс для получения строки из массива *randomWords*.

В сущности, можно сделать этот код короче, избавившись от переменной *randomIndex*:

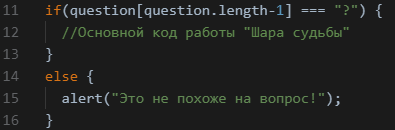
## Программа случайного выбора вариантов

Давайте попробуем применить наши знание на практике. Мы с вами создадим «Шар судьбы», который будет помогать отвечать на вопросы.

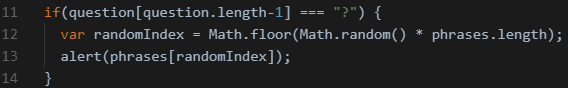
1. Спрашиваем у пользователя: «Спросите у шара, что пожелаете».
2. Проверяем правильно ли задан вопрос.
3. Выводим случайный ответ из, заранее подготовленного, массива ответов.

Первым делом создаём массив ответов.

 Когда ответы готовы, нужно вывести сообщение для пользователя. Так как сообщение вопросительное, то нам нужно воспользоваться функцией prompt().

Следующий шаг – это проверка правильности полученного вопроса. Для этого нам нужно проверить, стоит ли вопросительный знак в конце предложения. И если он там отсутствует, сообщить пользователю, что он неправильно задал вопрос.

После отсеивания предложений не похожих на вопросительные, мы должны реализовать основную часть – работу «Шара судьбы».

 Для этого нам нужно создать сгенерировать случайный индекс, с помощью которого мы выберем случайный ответ из заготовленного массива. А после выведем полученный ответ пользователю.

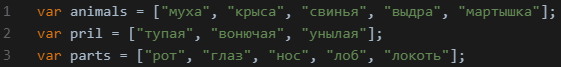
## Практическая работа

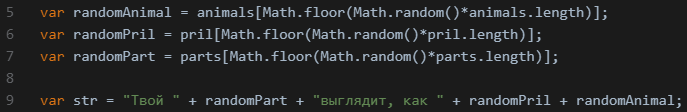
Доработать скрипт таким образом, чтоб можно было задавать вопрос шару судьбы несколько раз. Для этого нужно воспользоваться вопросом confirm() и циклом while.

*Полный скрипт в файле* **destiny.js**

## Генератор случайных дразнилок

Можно усовершенствовать код выбора вариантов, создав программу, которая при каждом запуске генерирует случайную дразнилку!

Создадим новый файл, в котором реализуем дразнилку. Для этого нам понадобиться три массива: массив животных, массив прилагательных и массив частей тела.

А после с помощью Math.random() и случайных индексов выберем из каждого массива по слову и соединим их в предложение.

А теперь самостоятельно сделайте вывод предложения.

## Домашнее задание

1. Добавить в дразнилку больше слов.
2. Спрашивать имя пользователя в начале и добавлять его к предложению.
3. Усовершенствуйте генератор дразнилок, чтобы он создавал дразнилки такого типа: «У тебя [часть тела] еще более [прилагательное], чем [часть тела животного] у [животное]». Подсказка: нужно будет создать еще один массив.