# Занятие 3 - Массивы и хэши. Циклы

## Массивы

Продолжаем знакомиться со встроенными в JavaScript типами данных. Еще один тип, очень часто используемый программистами - это массив. Массив позволяет хранить много данных в одной переменной. Можно представить себе его, как волшебный чемодан, поделенный на отсеки. В каждом отсеке лежит свое значение (например, число или строка). Каждый отсек пронумерован. Если мы хотим добавить в массив новое значение, он увеличивается, и в нем появляется новый отсек с числом.

В JavaScript массив создается таким образом:

const ***numbers*** = [4, 8, 15, 16, 23, 42];

const ***drinks*** = ['beer', 'milk', 'whiskey', 'cola'];

В первом примере мы создали массив чисел, во втором - массив строк. Массив может содержать любые значения, и даже значения разных типов в одном массиве, а также массивы внутри массивов.

const ***mixedArray*** = [1, true, 'I am a string', [0, 0]];

Однако, такими "смешанными" массивами редко удается пользоваться. Как правило, массивы создаются для хранения однотипных данных.

Для того, чтобы получить доступ к элементу массива, мы будем пользоваться **индексом** элемента. Это - **целое число**, обозначающее номер элемента в массиве (вроде бирки с номером для определенной ячейки чемодана). Индексация в массивах всегда начинается с нуля, поэтому первый элемент будет иметь индекс 0, второй элемент - индекс 1, и так далее.

Обращаться к элементам так:

> ***console***.log(drinks[0]);

< "beer";

Мы получили первый элемент массива, обратившись к нему и добавив индекс в квадратных скобках.

> ***console***.log(drinks[2]);

< "whiskey"

Третий элемент массива можно получить по индексу 2.

Добавить элемент в массив можно с помощью метода push:

drinks.push('pina colada');

Обратите внимание на неизвестный вам ранее синтаксис. Вы уже привыкли вызывать функции alert или prompt. Здесь - мы как бы вызываем функцию, привязанную к какому-то конкретному объекту. Объект - это наша переменная drinks, которая является **объектом типа Array** (массив). Такие функции мы будем называть **методами**. У всех массивов есть определенный набор **методов**, с помощью которых можно производить определенные действия. Также, методы есть у большинства объектов в JavaScript, но об этом мы поговорим позже.

Итак, с помощью вызова метода push у массива array, можно добавить к массиву значение, которое мы передаем в скобках. Это значение автоматически попадает в конец массива и ему назначается новый индекс. При этом мы говорим, что размер массива вырос на единицу.

Обратите внимание, что у нас не получается ошибки, несмотря на то, что мы объявили наш массив через ключевое слово **const**. Дело в том, что тот же самый массив остается на месте, он продолжает быть тем же значением этой переменной, просто у него меняются какие-то внутренние свойства (могут быть добавлены или изменены или даже удалены элементы). А вот если бы мы попробовали переназначить переменную drinks на другой массив, тогда мы получили бы ошибку:

drinks = ['milk', 'coffee', 'water']; // нельзя переопределить **const**

Узнать размер массива можно с помощью обращения к **свойству** массива length:

const ***len*** = drinks.length;

***console***.log(***len***);

< 5

Мы добавили к массиву элемент и его размер вырос на единицу.

Еще одна новая для вас концепция - **свойство**. Это нечто вроде дополнительной информации, которая всегда хранится в объекте, но к которой мы можем получить доступ, зная ее название. Например, мы теперь знаем, что у всех массивов можно узнать размер, обратившись к свойству length. Также, например, у строк, есть такое же свойство length, которое показывает длину этой строки:

const ***message*** = 'Hello!';

***console***.log(***message***.length);

< 6

И действительно, строка состоит из 6 символов, значит ее длина равна 6.

## Объекты (хэши)

Еще одним популярным типом для хранения данных являются объекты, или еще их называют хэшами. Объект позволяет хранить множество различных данных в себе, но в отличие от массива, хранение в нем обеспечивается не системой нумерации индексами, а с помощью строковых значений - **ключей**.

const ***john*** = {

firstName: 'John',

lastName: 'Doe',

age: 28

};

Мы создали объект из трех пар ключ-значение:

* Ключ 'firstName', имеет строковое значение 'John'
* Ключ 'lastName', имеет строковое значение 'Doe'
* Ключ 'age', имеет целочисленное значение 28

И присвоили его переменной john.

Теперь мы можем получать данные, обращаясь к каждому ключу как к свойству, как мы рассматривали ранее. Также можно сказать, что мы создали **объект** с тремя **свойствами**.

const ***message*** = john.firstName + ' ' + john.lastName + ' is ' + john.age + ' years old';

***console***.log(***message***);

< 'John Doe is 28 years old'

В хэшах как раз часто хранятся разнотипные данные. Например, в хэш john можно добавить массив его любимых сериалов:

john.series = ['Criminal Minds', 'Castle', 'The Mentalist'];

Как видите, добавлять пару ключ-значение в хэш можно с помощью оператора присваивания - мы указываем название нового ключа и его новое значение. Также можно переназначать существующие свойства, как это работает с отдельными переменными:

john.age = 29; // Сегодня у Джона день рождения

Увидеть, что содержит в себе массив или хэш можно с помощью метода console.log:



В консоли можно "раскрыть" объект, если в нем вложены массивы или другие хэши, можно просмотреть всю структуру с помощью консоли.

Удалить свойство из объекта можно с помощью ключевого слова delete:

delete john.series;

Также как с массивами, обратите внимание, что несмотря на то, что объект был создан через ключевое слово const, также как с массивом, мы можем без проблем добавлять, изменять и даже удалять свойства этого объекта. Потому что опять же, сам объект продолжает быть значением переменной, а его внутренняя структура может быть подвержена любым изменениям.

## Циклы

Работать с массивами и хэшами удобнее всего, перебирая их значения с помощью циклов. Однако, давайте сначала рассмотрим понятие цикла поподробнее.

Цикл - это организация повторяющихся действий для достижения нужного результата. Мы уже рассматривали примеры циклов на предыдущих занятиях и в домашних заданиях. Например, алгоритм точения карандаша:

1. Вставить карандаш в точилку
2. Пока карандаш не будет заточен:
3. Провернуть карандаш вокруг своей оси на 360 градусов
4. Вынуть карандаш из точилки

Здесь простой цикл, с повторением одного действия, пока нужный результат не будет получен.

Или, например, раздача карт при игре в "дурака":

1. Повторить шесть раз:
2. Для каждого игрока в группе
3. Отдать одну карту из колоды

У нас получается "вложенный" цикл - нам надо раздать 6 карт, по одной карте каждому игроку. Поэтому получается, что мы 6 раз проходим по каждому игроку и раздаем по карте.

В любом случае у нас есть некие действия, которые происходят повторно - их мы будем называть **телом цикла**. Каждый раз, когда тело цикла выполняется, мы будем называть одной **итерацией** цикла.

### Do .. while цикл

В JavaScript можно организовать циклы наподобие первого алгоритма, когда мы выполняем действия, пока не достигнем нужного нам результата. Например:

let ***water*** = 0;

do {

***console***.log('Pouring 1 liter of water into the bucket');

***water***++;

} while (***water*** < 10);

***console***.log(***water***);

Итак, наш цикл начинается с ключевого слова do (делать). После него идет блок кода, ограниченный фигурными скобками.

В блоке кода мы делаем две операции:

* Выводим на экран сообщение (что якобы наливаем 1 литр воды в ведро)
* Прибавляем к переменной water единицу.
* Специальная запись water++ означает то же самое, что и  
  water = water + 1;
* Собственно поэтому мы и объявили переменную water через ключевое слово let, потому что ее значение переопределяется на новое.

После блока кода идет ключевое слово while, что означает "пока". И потом идет условие, на которое цикл проверяет всякий раз после итерации цикла. Если это условие верно, то запускается следующая итерация цикла. Если не верно, цикл прерывается.

В данном случае условие гласит: water < 10. То есть во всех случаях, пока переменная water будет менее 10, цикл будет продолжаться. Это и есть как раз тот результат, который нам необходимо достичь - в данном случае выражается с помощью условия. Можно прочитать как "выполнять действия, пока количество воды меньше десяти".

Самое главное - в теле цикла увеличивать переменную water, иначе наш цикл станет бесконечным.

Также следует отметить, что цикл do .. while выполнит тело цикла как минимум один раз:

let ***water*** = 10;

do {

***water***++;

} while (***water*** < 10);

***console***.log(***water***);

В этом примере количество воды отобразится как 11, поскольку тело цикла выполняется, и только потом проверяется условие.

Зачем нужен такой цикл? С его помощью можно организовать, например, такое условие проверки ввода пользователя:

let ***yourName***;

do {

***yourName*** = prompt("Who are you?");

} while (***yourName*** === '');

***console***.log(***yourName***);

Здесь сначала выполняется тело цикла, которое запросит у пользователя имя. Далее, происходит проверка условия yourName === ''. Если мы не введем ничего в предложенное prompt-ом поле, то в переменную вернется пустая строка. Так что, на данном этапе это условие будет верным и цикл запустит следующую итерацию. Если мы что-то введем в поле, то условие будет ложным и произойдет выход из цикла. Так мы можем быть уверенными в том, что пользователь что-то введет. Конечно, это не самый лучший способ ввода данных на веб-сайте, но для образовательных целей он нам подходит.

### While цикл

Следующий тип цикла очень похож на этот, но с одной разницей - while стоит в начале, а не в конце, и проверка условия соответственно тоже произойдет перед первой итерацией. В случае ложного условия этот цикл может не отработать ни разу.

let ***number*** = 1;

while (***number*** < 6) {

***console***.log('This is number', ***number***);

***number***++;

}

Данный код выведет пять сообщений, с числами от 1 до 5. После 5-ой итерации условие не сработает, т.к. 6 < 6 - это false.

### For цикл

While-циклы применяются часто в сочетании со специальными счетчиками, как в предыдущих примерах (water, number) и как правило, часто нужно просто повторить нужные действия определенное количество раз.

Для этой цели также существует специальный цикл, называемый for-циклом. Выглядит он следующим образом:

for (let ***i*** = 0; ***i*** < 5; ***i***++) {

***console***.log("The current iteration is", ***i*** + 1);

}

Здесь в скобках указываются следующие значения, разделенные точками с запятой:

* let i = 0 - объявляется переменная i, с начальным значением 0. Первый параметр в цикле for называется "**инициализацией**" - это действие производится только один раз в самом начале, перед проверкой на условие.
* указывается условие i < 5. Второй параметр цикла называется "**условием продолжения**". Это условие проверяется каждый раз перед началом выполнения тела цикла (кода в фигурных скобках). Если условие является правдой, то тело цикла выполняется.
* i++ - оператор "инкремента" - как мы уже знаем, является краткой записью выражения i = i + 1. Третий параметр цикла называется "**послесловием**" (afterthought) - выполняется всякий раз после завершения тела цикла.

Эти три значения обеспечивают выполнение цикла до достижения переменной i значения 5, при достижении которого "условие продолжения" перестает быть равным true, и цикл прекращается.

## Работа с массивом в цикле

Циклы очень часто используется для перебора элементов массива. Рассмотрим, как вывести список напитков из массива в консоль:

const ***drinks*** = ['beer', 'milk', 'whiskey', 'cola'];

for (let ***i*** = 0; ***i*** < ***drinks***.length; ***i***++) {

const ***drink*** = ***drinks***[***i***];

***console***.log("Today's drink is:", ***drink***);

}

Итак, сначала мы объявляем переменную-счетчик i, равную нулю. В данном случае можно сказать, что i - это у нас "индекс". Мы ставим его на нулевую позицию, с которой начинаются индексы массива. Далее, в условии продолжения мы сравниваем этот счетчик с длиной массива. Так получается, что длина массива всегда будет на единицу больше, чем последний индекс элемента в этом же массива (мы уже знаем почему, потому что индексация в массивах начинается с нуля):

const ***drinks*** = ['beer', 'milk', 'whiskey', 'cola']

// 0 1 2 3

***console***.log(***drinks***.length); // 4

Поэтому, в нашем случае условие i < drinks.length при увеличении каждый раз индекса на единицу обеспечивает нам прохождение в цикле ровно столько раз, какова длина массива. Когда i станет равным этой величине, условие станет ложным и цикл прервется.

Теперь в теле цикла мы:

* Создаем новую переменную drink из элемента массива drinks[i]. Обратите внимание, что в этом случае, мы вместо числа-индекса в квадратные скобки передаем переменную i. На каждой итерации цикла она будет равна новому индексу массива, т.к. каждый раз увеличивается на единицу.
* Печатаем сообщение, содержащее текущее значение переменной drink.

Также мы можем, например, заполнить массив, в цикле спрашивая у пользователя нужные значения:

const ***grades*** = [];

for (let ***i*** = 0; ***i*** < 3; ***i***++) {

const ***grade*** = parseInt(prompt('Enter grade' + ***i***));

***grades***.push(***grade***);

}

***console***.log(***grades***);

## Операторы краткой записи

Рассмотрим еще несколько операторов, которые существуют в JavaScript. Вы уже познакомились с оператором инкремента ++:

let ***i*** = 245;

***i***++;

***console***.log(***i***);

< 246

Он увеличивает значение переменной на единицу. Также есть оператор декремента, который делает противоположное - уменьшает значение переменной на единицу:

let ***i*** = 245;

***i***--;

***console***.log(***i***);

< 244

Также, существуют операторы, похожие на операторы инкремента и декремента, тем, что они сокращают количество кода, и служат как сокращенная запись определенных кусков кода, которые часто встречаются. Например, часто встречается такая запись:

let ***message*** = '';

***message*** = ***message*** + 'Hello';

***message*** = ***message*** + ' world!';

***console***.log(***message***);

< "Hello world!"

Получается, что мы с помощью оператора присваивания (=) назначаем переменной message новое значение, состоящее из предыдущего значения этой переменной, плюс некоторая новая строка, которая конкатенируется с предыдущей. Для таких случаев существует краткая запись:

let ***message*** = '';

***message*** += 'Hello';

***message*** += ' world!';

***console***.log(***message***);

< "Hello world!"

Этот оператор присваивания с добавлением (+=) берет предыдущее значение переменной и прибавляет к нему новое значение. Можно из этого примера увидеть, насколько уменьшается и становится читаемее код. Такой оператор работает не только с конкатенацией строк, но также и с математическими операциями:

let ***number*** = 100;

***number*** += 66;

***console***.log(***number***);

< 166

Также существуют похожие операторы для математических операций, которые производят базовые математические действия. Вот сводная таблица таких кратких операторов и какие действия они могут производить:

|  |  |  |
| --- | --- | --- |
| i++ | i = i + 1 | Часто используется в циклах как послесловие для постоянного увеличения значения на единицу |
| j-- | j = j - 1 | Также используется в циклах (например, при переборе массива с конца к началу) |
| a += b | a = a + b | Может быть использовано для конкатенации строк, в случае если a и b - переменные строковых типов, а также для математического сложения |
| a -= b | a = a - b | Используется для математического вычитания. |
| a \*= b | a = a \* b | Используется для математического произведения. |
| a /= b | a = a / b | Используется для математического деления. |

Также, рассмотрим несколько математических операций, например, получение случайного числа, что бывает полезно в решении задач или создании игр:

> Math.random();

< 0.13215622209922206

Метод Math.random() возвращает случайное дробное число от 0 до 0.9(9). Что же делать, если мы хотим получить случайное целое число, например от 1 до 6, чтобы сделать, например, бросок кубика d6? Для этого применяют следующую технику:

* Умножают получившееся число на максимальную границу (в нашем случае на 6).
* Прибавляют к числу единицу. Дело в том, что при умножении достаточно малого числа на 6, мы можем получить число меньше единицы, и так как Math.random() не возвращает число 1, мы никогда не получим ровно 6, так что наш текущий интервал будет равен от 0 до 5.
* Отсекаем дробную часть получившегося числа. Для этого воспользуемся еще одним методом Math.floor, который как раз делает это.

Полная формула получения случайного целого числа от 1 до 6 теперь у нас равна:

const diceThrow = Math.floor(Math.random() \* 6 + 1);

Еще несколько некоторых популярных математических операций:

|  |  |  |
| --- | --- | --- |
| Math.random() | > Math.random()  < 0.08804465516907567  > Math.random()  < 0.34859479734797239 | Возвращает случайное дробное число (float) в пределах [0, 1). |
| Math.floor(a) | > Math.floor(1.6)  < 1  > Math.floor(4.99999)  < 4 | Округляет "вниз" до ближайшего целого. Эффективно отсекает дробную часть числа. |
| Math.ceil(b) | > Math.ceil(3.9)  < 4  > Math.ceil(1.000001)  < 2 | Округляет "вверх" до ближайшего целого. |
| Math.round(c) | > Math.round(1.5)  < 2  > Math.round(1.49)  < 1 | Округляет по математическим правилам. |
| Math.pow(a, b) | > Math.pow(2, 2)  < 4  > Math.pow(3, 3)  < 27 | Возводит число a в степень b |

Если необходимы другие математические операции, здесь находится полный список всех методов объекта Math:

<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Math>

## Перенос строки

Иногда, у вас будут попадаться строки, которые будут достаточно длинными. Их, конечно, можно вместить на одну строку, однако в этом случае окно вашего текстового редактора придется двигать, либо включать перенос строки, что не всегда бывает читаемо. Чтобы перенести строку на несколько строк, можно воспользоваться такой техникой:

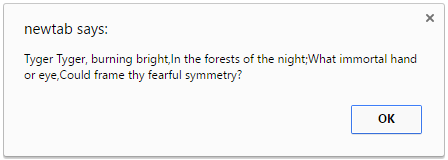
const ***secretMessage*** = 'Tyger Tyger, burning bright,' +

'In the forests of the night;' +

'What immortal hand or eye,' +

'Could frame thy fearful symmetry?';

Вот таким образом, мы переносим длинные строки с помощью конкатенации. Знак конкатенации остается в конце строки, затем начинается новая строка. В конце всего сообщения мы ставим точку с запятой - завершаем мысль. Однако что будет, если мы распечатаем это сообщение на экран?



К сожалению, все сообщение распечаталось в одну строку, без учета переноса. Дело в том, что приведенный выше пример - это обычная конкатенация, и если бы мы написали весь код в строку, то получили бы то же самое.

В таблицах кодировки символов существуют специальные "контрольные" символы, которые позволяют нам обеспечивать некоторое форматирование текста. С одним таким символом вы уже знакомы - это пробел, который позволяет нам отделять в тексте слова друг от друга. Он настолько часто используется, что для него, очевидно, есть специальная кнопка на клавиатуре - пробел. Однако, как мы переносим строки? Нажатием кнопки "ввод" - Enter на клавиатуре. Дело в том, что в текстовом редакторе эта кнопка всего лишь вводит специальный символ переноса строки при нажатии на эту кнопку. Мы знаем, что если мы нажмем Enter посреди строки JavaScript, редактор просто разобьет эту строку на две и начнет вторую часть с новой строки. То есть, редактор делает не совсем то, что надо - кнопка Enter позволяет нам редактировать код, а не то, что мы хотим вывести на экран. Для того, чтобы вывести на экран строку с переносом строки, мы должны ввести специальный символ в текст. Так как у этого символа нет визуального отображения (мы видим лишь как текст начинает быть с новой строки), в JavaScript есть специальные комбинации символов, которые обозначают подобные символы:

const message = "Hello,\nWorld!";

alert(message);



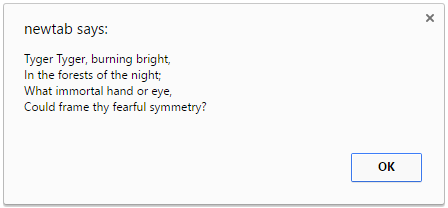
Как можно понять, нужная нам комбинация - это "\n". Буква n в данном случае означает newline - новая строка. Итак, если мы в нашу строку вставим комбинацию символов \n, то оно будет означать перенос строки на новую строку. И мы можем отредактировать текст стихотворения выше, чтобы оно выглядело как следует:

const ***secretMessage*** = 'Tyger Tyger, burning bright,\n' +

'In the forests of the night;\n' +

'What immortal hand or eye,\n' +

'Could frame thy fearful symmetry?';



## Вложенные циклы и условия

Как вы уже поняли, в теле цикла может находиться любой код, любое его количество. Точно также, в теле цикла могут располагаться какие-либо условия, или даже другие циклы. Посмотрим пример "реальной" программы, которая выведет на экран таблицу умножения.

let ***string*** = '';

for (let ***i*** = 1; ***i*** <= 9; ***i***++) {

for (let ***j*** = 1; ***j*** <= 9; ***j***++) {

***string*** += (***i*** \* ***j***) + ' ';

}

***string*** += "\n";

}

alert(***string***);

Итак, разберем данную программу по порядку. У нас имеется два цикла, при этом один из них "вложен" в другой. Если мы воспользуемся той же техникой, что и ранее, и попробуем "отслеживать" все итерации по очереди, мы увидим, что во время первой итерации первого цикла (будем называть его i-цикл), выполнятся все 9 итераций вложенного j-цикла.

В теле j-цикла у нас происходит добавление к переменной string нового значения произведения переменных i и j. В первой итерации i-цикла, значение переменной i всегда равно 1.

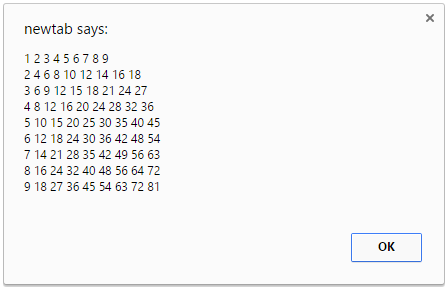
Так как итерация не считается оконченной, пока не выполнится весь код тела цикла, мы входим в j-цикл. Таким образом, на первой итерации i-цикла, j-цикл произведет все 9 итераций.

На всех итерациях j цикла, переменная j будет увеличиваться от 1 до 9, и произведение 1 \* j будет давать нам таблицу умножения на 1.

Затем, после окончания j-цикла, мы прибавляем к строке символ переноса на новую строку, чтобы новая таблица (умножение на 2) начиналась с новой строки.

Во второй итерации i-цикла переменная i будет равна 2, на всех девяти итерациях j-цикла будет собираться таблица умножения из произведения 2 на числа от 1 до 9, и так далее.

В результате мы получим следующее на экран:



## Вложенные условия

Также как и с циклами мы можем вкладывать условия в условия. Посмотрим на такой пример. Программа, которая имитирует разговор клиента с баристой у кофейного стенда:

const drinkName = prompt('What do you want to drink? Hint: coffee');

if (drinkName === 'coffee') {

const ***variant*** = prompt('How to make your coffee? black/with cream');

switch (***variant***) {

case 'black':

alert('One black coffee coming right up!');

break;

case 'with cream':

alert('Coffee with cream is a nice choice');

break;

default:

alert('We don't have that variety');

}

} else {

alert('Sorry, we serve only coffee.');

}

Выше приведен пример того, что в if можно вкладывать другие конструкции, в этом случае switch.

Таким образом, можно сказать, что в блоки кода, ограниченные фигурными скобками, можно вкладывать любой другой код и конструкции.

## Бесконечные циклы. Оператор break.

let ***number*** = 0;

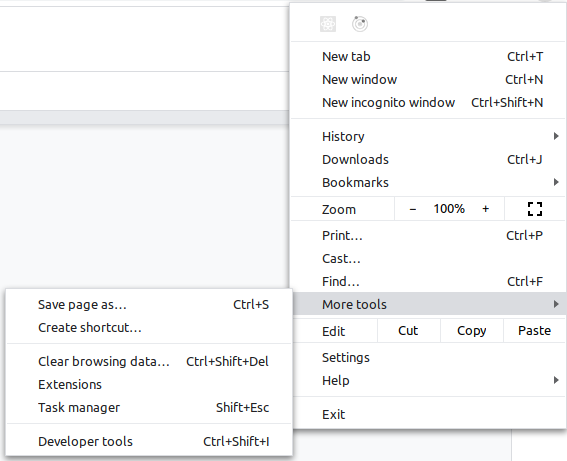
while (***number*** >= 0) {

***console***.log('Number is', ***number***);

***number***++;

}

В данном случае условие цикла таково, что условие продолжения будет выполняться всегда, т.к. мы увеличиваем number, и цикл будет выполняться бесконечно. Такие циклы называются "бесконечными". Если мы запустим такую программу, то наша вкладка браузера "зависнет", поскольку поток выполнения JavaScript для данной вкладки будет вечно заблокирован этим бесконечным циклом. Желательно не запускать подобную программу вообще, но если уж это произошло, в этом случае необходимо "убить" процесс вкладки через менеджер процессов. Например, в Google Chrome - это комбинация клавиш Shift + Esc, либо зайти в меню (три точки) - More Tools - Task Manager



Затем остановить процесс, отъедающий максимальное количество ресурсов процессора (CPU)

Однако, иногда бывает необходимо создать "псевдо"-бесконечный цикл - то есть такой, который надо остановить в нужный момент. Один из таких примеров мы уже видели:

let ***yourName***;

do {

***yourName*** = prompt("Who are you?");

} while (***yourName*** === '');

***console***.log(***yourName***);

Не всегда конечно было бы удобно создавать именно цикл do..while, поэтому есть возможность "останавливать" любые циклы с помощью ключевого слова break. Перепишем этот цикл таким образом:

let ***yourName***;

while (true) {

***yourName*** = prompt('Who are you');

if (***yourName*** !== '' || ***yourName*** !== null) {

break;

}

}

В этом примере вы видите пример "классического" бесконечного цикла - заданного через while (true) - таким образом мы сразу явно даем понять, что хотим повторять блок кода в цикле до какого-то момента внутри.

Если внутри выполняется нужное нам условие (введенные данные не равны пустой строке и пользователь не нажал "Отмена"), то выполняется оператор break, который выходит из цикла while и программа продолжает выполняться дальше за телом цикла (или останавливается, если там ничего дальше нет).

А если условие не выполняется, то цикл продолжается и опять запрашивает у пользователя данные.