

# *JavaScript с нуля за 100 часов*

*как ваш первый язык программирования*

Встреча вторая.

Типы данных. Линейный алгоритм

Академия ТОП  
Баринова Вера Олеговна  
системный программист и преподаватель  
стаж с 2005 года

[Эй, я не дочитал,  
верните меня обратно](#)

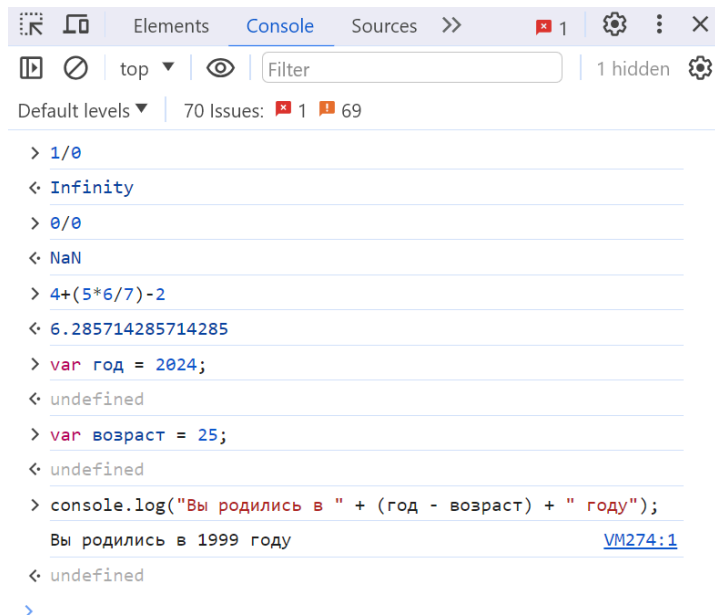
<b>Калькуляторы повсюду.....</b>	<b>3</b>
<b>Операторы и типы данных.....</b>	<b>3</b>
<b>Числа. Number, BigInt.....</b>	<b>4</b>
<b>Логический тип boolean.....</b>	<b>6</b>
<b>Строки.....</b>	<b>7</b>
Форматирующие строки.....	9
Escape-последовательности.....	9
<b>Преобразование типов.....</b>	<b>10</b>
<b>Линейный алгоритм.....</b>	<b>11</b>
<b>undefined, NaN и null.....</b>	<b>14</b>
<b>Подведём итоги.....</b>	<b>16</b>
Приоритет операций в JS.....	16
<b>Словарь урока.....</b>	<b>18</b>

Презентация "[быстрый старт](#)"

Презентация "[типы данных](#)"

## Калькуляторы повсюду

Откройте консоль. Введите пару арифметических примеров, используя действия сложения, вычитания, умножения и деления. Поделите на ноль. Поделите ноль на ноль.



```
> 1/0
< Infinity
> 0/0
< NaN
> 4+(5*6/7)-2
< 6.285714285714285
> var год = 2024;
< undefined
> var возраст = 25;
< undefined
> console.log("Вы родились в " + (год - возраст) + " году");
Вы родились в 1999 году VM274:1
< undefined
>
```

Создайте несколько переменных, но теперь вместо строк присвойте им числа. Попробуйте прибавить к переменной переменную. Попробуйте прибавить число к строке. Попробуйте вычесть из строки строку. Или вычесть из `i` 17.

Во-первых, вы только что обзавелись ещё одним калькулятором, который всегда под рукой, когда вы чувствуете неуверенность в своих подсчётах в уме. Во-вторых, вы встретились с ошибками, которые поджидают начинающего программиста: ошибками, приводящими к аварийной остановке программы.

Вычитание из отсутствующей переменной `i` приводит к ошибке, которая пишется в консоль красным цветом и прерывает работу программы. Применение оператора сложения к строке и числу, (если строка не состоит целиком из цифр!) приводит к неожиданному результату: строки склеиваются. Чтобы разобраться в том, как поведет себя тот или иной оператор, познакомимся с типами данных и применимыми к ним действиями.

## Операторы и типы данных

В отличие от математики в программировании все просто: вместо строгого определения "что такое оператор", можно составить полный список. Все, что не в списке - не оператор.

Для начала поделим все операторы на две большие группы: модифицирующие (меняющие значения переменных) и все остальные. К модифицирующим относится уже знакомый вам оператор присваивания `=`. К немодифицирующим относятся, например, арифметические операторы.

Посмотрите на пример. Обратите внимание на записи рядом со строчками. Они называются "комментарии". Эта часть кода не анализируется интерпретатором (ну, почти) и нужна для программиста, чтоб не забыть, что делает строчка, или чтобы поделиться этим с другим программистом. В JS комментарии бывают до конца строки от `//` или внутри `/**/` между звёздочками. Символы `//` или `/*` нельзя разделять пробелом!

JavaScript

```
var рост = 170; // создана переменная, записано значение 170
рост + 30;      // Вычислен результат - сумма роста и 30, А НЕ ПРИБАВЛЕНО 30!
console.log(рост); // вывод результата в консоль
```

Как видите, переменная не изменилась от действия арифметического оператора. Чтобы переменная изменилась, потребуется либо явно записать в неё новое значение с помощью оператора присваивания, либо воспользоваться арифметическим присваиванием. Арифметические операторы представляют собой два подряд идущих знака - знак арифметической операции (+-\* и т.д.) и знак равенства. Важно! Между ними нельзя ставить пробел.

**+=** это арифметическое присваивание - прибавить к переменной то, что стоит справа и записать в переменную результат сложения.

JavaScript

```
рост = рост + 10;
console.log(рост);
рост += 10;
console.log(рост);
```

Слева пример. В первой строке вычисляется сумма роста и десяти и записывается (присваивается) обратно в рост. В третьей строке делается РОВНО то же самое, но запись получается короче. Вы можете пользоваться тем вариантом, который вам понятнее, они абсолютно эквивалентны.

Создайте новую папку для сегодняшней практики, новую веб-страницу и новый скрипт.

**Запишите решение задачи:** "Клиент заказал два билета до Дублина и обратно по одинаковой цене 150 долларов. Затем применил промо-код на скидку 20%. На какую сумму выставляем ему счёт?" Пользоваться можно операторами присваивания, сложением, умножением, вычитанием и делением, создавать сколько угодно переменных. Решение во вложении. Обратите внимание на разные способы решения.

## Числа. Number, BigInt

В этих примерах вы познакомились с числовым типом данных number. В JS все такие числа являются вещественными, но если после запятой стоят нули, вывод будет похож на целое число. Другие примеры записи чисел типа number можно увидеть в программе:

JavaScript

```
var g = 9.8; // м/с в квадрате
var G = 6.67e-11; // Научная запись числа. -11 - степень десятки!
var c = 0xC0DE; // 16-ричная запись числа
var permissions = 0o755; // 8-ричная запись числа
var маска = 0b11001100; // двоичная запись числа
```

У нас нет возможности указывать единицы в коде! В языке не реализована возможность понимания физических единиц (метры, килограммы...), но вы можете сделать это сами с помощью классов. Все числа изначально безликие, и JavaScript все равно, каких коров с какими яблоками вы будете складывать. Задание: откройте консоль. Создайте все эти переменные и выведите их на печать (в консоль). Если к моменту чтения вами учебника ничего не изменилось, JavaScript автоматически преобразовал все числа, кроме G, к привычному нам виду в десятичной системе счисления.

Над числами определены арифметические операторы, т.е. действия математики. Помимо привычных сложения, умножения, вычитания и деления, определен оператор получения остатка от деления - знак процент %.

Например, остаток от деления числа 20 на число 7 равен 6, потому что 14 поделится нацело и ещё 6 останется. Остаток не бывает отрицательным и всегда меньше делимого.

Забавно, что он работает не только с целыми, но и с дробными числами. Впрочем, в JavaScript все числа (number) вещественные.

```
console.log(5.6 % 3.2);
```

Правда, как и все числа с плавающей запятой в компьютере, результат получается неточным. Вернее, кажется, что "не всегда точным", но в действительности, всегда неточным. Вспомните, как выполняли действия на калькуляторе, и получали ответ, у которого много девяток в дробной части. Здесь можно говорить только о том, что два числа отличаются не более, чем на заранее определенную величину.

Числа типа number хранятся в ограниченном количестве байт (на момент написания учебника 64) и в связи с этим представлены в виде мантииссы (53 бита) и экспоненты, поэтому существует максимальное целое число, которое может быть представлено в этом виде. Затем происходит потеря точности. Если вы не знакомы с двоичным представлением чисел в компьютере и не имеете времени их изучить, просто убеждайтесь, что ваше число меньше, чем `Number.MAX_SAFE_INTEGER`, либо переходите к другому типу - `BigInt`.

Тип `BigInt` не ограничивает вас ничем, кроме памяти компьютера, однако не может быть использован для вещественных чисел. Но даже маленькие числа можно записывать с помощью `BigInt`, однако вычисления с ними медленнее, чем с обычным `Number`. Пример:

JavaScript

```
var len = 125n; // просто число 125. Буква n показывает, что тип BigInt
var volume = 1024n*1024n*1024n*1024n*1024n*1024n; // 1 Exabyte (Экзбайт)
```

125, конечно, можно записать и с помощью обычного числа, а вот объемы в экзбайтах уже в Number "не умещаются". Всё, что больше Петабайта, требуется хранить в этом специальном типе. Кстати, простое взаимодействие между простыми и большими числами запрещено, как вы думаете, почему?

Помимо обычного и арифметического присваивания модифицирующими являются операторы инкремента и декремента. Они не только уменьшают или увеличивают значение переменной на единицу, но и возвращают результат. *Очень забавно выглядит увеличение на единицу нецелого числа 😊* Пример приведите сами.

```
> var день = 15; // создали переменную
< undefined // создание переменной не возвращает значения
> ++день; // префиксный инкремент
< 16 // возвращает новое значение и меняет переменную
> console.log(день);
16 // печать печатает в консоль
< undefined // печать не возвращает значения
> день++; // постфиксный инкремент
< 16 // возвращает старое значение, но переменную меняет
> console.log(день);
17 // напечатали, убедились, что день изменился
< undefined // печать не возвращает значения
>
```

Операторы бывают записаны не только значками, но и словами. Изучая массивы, вы познакомитесь с оператором delete, а в части про дату и время - new.

Существует оператор, показывающий тип данных - typeof, однако, область его применимости сильно ограничена - он покажет только число, большое число, строку, истину/ложь и несколько особых типов. Для более сложных типов потребуется другой подход, о котором вы узнаете позднее.

## Логический тип boolean

Назван в честь Джорджа Буля, много сил посвятившего математической логике. Логический тип содержит всего два значения - истина (true) и ложь (false), однако именно на этом типе данных построены все алгоритмы сложнее, чем прямая последовательность инструкций, не допускающая вариантов поведения в различных условиях. Все ситуации "а если" и "пока не" решаются именно с помощью логики, для чего требуется не только хранить фиксированные истину и ложь, но и обращаться с операторами, вырабатывающими логические значения, т.е. отвечающими на вопрос "да" или "нет". Для сравнения чисел и строк существуют операторы >, <, >=, <= и сравнение на равенство, который, в силу того, что одинарный знак равенства уже используется для присваивания, записывается с помощью == и ===.

> 3 > 8	> 3 <= 8	> "6" == 6	> "6" === 6
< false	< true	< true	< false

Перед нами 4 результата вычислений, представленных в виде логического типа данных. Обратите внимание, что для строки 6 и числа 6 два разных сравнения на равенство дают разный результат. Это очень важный момент: JS может автоматически преобразовывать некоторые типы к другим в особых случаях, но если запросить строгое равенство с учётом типа, то т.к. строка и число - разные типы, несмотря на совпадающие значения, мы получаем результат "ложь", "эти два значения не равны с учётом типа".

## Строки

Вы уже немного знакомы со строками: выводили в консоль текстовые сообщения, обрамляя текст кавычками.

```
JavaScript
var city = "Москва";
var lat = 55;
var lon = 37;
console.log(city + ' расположена на ' + lat + ` северной широты и ${lon}
восточной долготы.`);
```

Сложение строк не является математической операцией, поэтому оператор "+" применительно к строкам называют конкатенацией строк. [О вычитании - позже](#).

В JavaScript можно прибавлять к строке значения любого типа, они будут автоматически преобразованы в строку, как для печати. Это удобно, но может приводить к ошибкам, если вы не собирались этого делать.

Рассказываю историю с работы. Писала я программу, где получала дату с веб-страницы и рисовала график на эту дату. Программа работала хорошо, пока не потребовалось рисовать график не за текущий месяц, а за следующий. Визуально графики были настолько похожи, что я не заметила, что почему-то вместо марта (в феврале) график был нарисован за декабрь. Программа брала месяц в строковом виде (в JS месяцы нумеруются с нуля), для февраля это '1'. Прибавляла к нему единицу, получая '11' и рисовала график. В марте (2-ой месяц) программа сломалась, потому что 21-го месяца не существует, я заметила и исправила ошибку.

Строка "знает" свою длину. Это очень удобное и полезное знание получают так:

```
JavaScript
var city = "Константинополь";
console.log(city.length);
```

Осторожно! Для некоторых кодировок длина строки может отличаться от количества символов, хотя изначально была задумана именно так. Это известная ошибка, но для неё пока существуют только временные решения. Для вычисления количества символов можно [воспользоваться преобразованием в массив](#).

Строки, как мы уже писали, можно сравнивать. Если вы догадались, что равенство строк - это полное посимвольное совпадение строк одинаковой длины, то легко догадаетесь, что сравнение на *неравенство* (т.е. выяснение, какая строка "больше" или "меньше") выполняется... НЕТ, НЕ ПО ДЛИНЕ! А по тому, какая строка стояла бы раньше в словаре. Помните, как в школе вызывали к доске? Все выписаны в журнале по алфавиту, и если ваша фамилия, вроде моей, начинается на букву Б, вероятность, что вас вызовут, значительно выше, чем если она начинается на Щ.

Однако, тут есть тонкость. Поясним на примере. Пусть надо сравнить две строки: "Маша" и "Паша". Они одинаковой длины, но М в алфавите стоит раньше П и, по введенному правилу очевидно, Маша меньше Паши, потому что стоит в списке раньше.

Теперь пусть надо сравнить Пашу и кашу:

JavaScript

```
console.log("каша" > "Паша"); // истина!
```

Почему же Паша оказался меньше каши? Дело в том, что на самом деле, сравнение происходит не по алфавиту, а по номеру символа в таблице кодировок. Если вы вдруг забыли или никогда не знали, буквы в компьютере существуют только при отображении текста человеку на экран, принтер и т.д. Так же, как звуки, цвета и прочее. В компьютере обычно существует только два значения "условный ноль" и "условная единица" (о троичной, квантовой и прочих архитектурах вы можете почитать самостоятельно). "Включено" или "Выключено", "Заряжено" или "Разряжено". Из двух цифр удастся сформировать двоичную систему счисления, изучение которой также не входит в данный курс. Двоичная система счисления позволяет использовать целые и вещественные числа, которые, в свою очередь, могут обозначать буквы, цвета и звуки, если каждому цвету, букве или звуку поставить в соответствие номер. Иными словами, компьютер "внутри себя видит" числа и числовую пометку: отобразить как буквы. Все буквы, знаки препинания, арифметические знаки и т.д. - просто пронумерованы! Конечно, единственным и неповторимым образом (закадровый смех). Т.е. я хочу сказать, что таблиц кодировок (нумераций символов) придумано достаточно много и вам ещё предстоит получить массу удовольствия, встречаясь с перекодированием "зюк" ("зюки", "крокозябры" - бессмысленный на первый взгляд набор символов, свидетельствующий о том, что для отображения текста была выбрана не та кодировка, которая использовалась при сохранении текста). Так вот, исторически заглавные буквы появились в компьютерах раньше, поэтому в таблицах кодировок их номера МЕНЬШЕ! Кроме того, буква ё вообще стоит в кодировках "где попало", и её номер НЕ оказался между е и ж.

На сладкое: (копировать!!! не перепечатывать!!!) чему и почему равно 'с' > 'с'?



## Форматирующие строки

Теперь посмотрим ещё раз внимательно на пример. "Москва" написано в двойных кавычках, ' расположена на ' в одинарных и после lat прибавляется строка, записанная в ОБРАТНЫХ кавычках, которые расположены на букве Ё на клавиатуре. На Macintosh раскладка несколько отличается.

JavaScript

```
var city = "Москва";  
var lat = 55; // градусов  
var lon = 37; // градусов  
console.log(city + ' расположена на ' + lat + ` северной широты и ${lon}  
восточной долготы.`);
```


Такая строка называется и является шаблоном (форматирующей строкой), т.е. в неё могут быть добавлены специальные символы, осуществляющие подстановку значений, которые неизвестны на этапе написания программы и будут получены в процессе её работы (run-time). Помимо переменных между \${ и } могут располагаться целые выражения, однако, не рекомендуется этим злоупотреблять, чтобы не усложнять читаемость кода. Лучше вычислить сложное выражение заранее. Кроме того, такая строка может содержать символ перевода строки в явном виде (выглядит, как новое слово на новой строке), не отображающийся на печати, т.е. состоять из нескольких строк текста. Как выглядит символ перевода строки, когда строку нельзя "разрывать" с помощью нажатия enter, мы узнаем чуть позже.

Недостатки: нельзя "легко" определить строку заранее, а значения подставить потом. Т.е. шаблон не может использоваться многократно. Для этого приходится прибегать к специальным приёмам. Первый, довольно простой - использовать метод замены. Второй будет продемонстрирован на уроке по функциям.

Что такое метод? Вы уже встречались с функциями вывода в консоль console.log() и console.error(), где console - это объект консоли, а log или error - функции, выполняющие вывод или подкрашенный красным вывод. Это были *методы объекта консоль*. То есть метод - это функция или последовательность функций, являющаяся собственностью объекта.

## Escape-последовательности

Одинарные и двойные кавычки в JS не отличаются. Вы начинаете строку и заканчиваете одними и теми же кавычками. Если строка должна содержать кавычки или апостроф, просто используйте в начале и в конце кавычки другого типа. А что делать, если требуется в качестве символов использовать оба вида кавычек? Использовать шаблонную строку? А если все три типа? А если в шаблонную строку потребуется вставить знак доллара или фигурные скобки?

Во всех языках программирования эта проблема решается с помощью ввода в строку специальных символов. Обратная косая черта (back slash ) - начало специального

символа. Для ввода в строку собственно обратной косой черты требуется её удвоить (привет, пути в windows-подобных системах!), кавычки "забэкслэшиваются" (экранируются) и тоже становятся обычными символами. Также полезно знать, что для перевода строки используется `\n`, однако, поскольку преимущественно ваши тексты будут на странице, где перенос строки выполняется иначе, то этот символ вам будет редко нужен для клиентской части сайта. Если Вам потребуется какой-либо символ, который невозможно ввести с клавиатуры, можно использовать последовательность типа `'\u2698'`, где после буквы `u` следует 16-ричный код вашего символа. В данном случае появится цветочек ♡. Полный список можно получить, выведя циклом все символы подряд с помощью функции `String.fromCharCode(i)`. Мы [выполним это упражнение](#) в соответствующем уроке.

## Преобразование типов

Вы уже видели, как одни типы могут превращаться в другие в результате выполнения действий над ними. На вершине человеческой мысли тут, как всегда, строки. Складываем (конкатенируем) строки между собой или с числами - получаем строку. Вычитаем... ПОЛУЧАЕМ NUMBER! Причем, если строки можно преобразовать к числу, например, `"7" - "2"`, то получается 5, а если нельзя, то NaN! (Not a Number, который является специальным значением типа `number`). Как вам такое?

Кроме автоматического (неявного) преобразования, в JS существуют различные способы явного (принудительного) преобразования типов.

## Преобразование строки в числа

Компьютер хранит числа и строки схожим образом, однако существует некоторое отличие. Так или иначе в большинстве компьютеров (о троичной логике вы можете почитать самостоятельно) данные хранятся в двоичном виде - в виде последовательностей единиц и нулей. Как же получается, что человек видит картинки, слышит музыку, читает буквы? Для отображения информации в привычном человеку виде у компьютера существуют динамики, всего лишь чаще или реже, сильнее или слабее быстро-быстро толкающие воздух (звуковые волны) и программно-аппаратная система из видеокарты и монитора, знающая, как надо "зажечь" красную, синюю и зелёную компоненты каждого пикселя монитора, чтобы на нём "нарисовалась" буква или появилась картинка, собранная из цветных пикселей, как из мозаики. О кодировке цвета вы уже говорили на курсе CSS и ещё раз вспомните в другой раз, а сейчас напомним, что каждой букве, цифре и знакам препинания соответствует код (число!) в таблице кодировок. При этом числа могут храниться в виде строк `'65535'` и в "исходном виде" - числами. В первом случае для записи числа потребуется 5 байт по одному на каждую цифру, во втором - всего два байта, поскольку указанное число представляет собой  $2^{16} - 1$ . В некоторых случаях преобразование произойдет автоматически (вычитание, например), а в некоторых - нет! При сложении строки останутся строками. Для преобразования их в число потребуется одна из двух функций:

JavaScript

```
let days_in_a_year_str = '365';
let days_in_a_year = parseInt(days_in_a_year_str);
let g_constant_str = '9.78';
let g_constant = parseFloat(g_constant_str);
```

Упражнение: выведите в консоль типы данных всех переменных. Выполните сложение и убедитесь, что `days_in_a_year_str` и `g_constant_str` “склеиваются” при сложении, а не складываются математически.

## Преобразование числа в строку

Помимо привычной нам арифметики, в JS доступны действия с отдельными битами числа с помощью операций битовой (двоичной) арифметики, а для логических выражений используются логические “не” (!), “и” (&&) и “или” (||). Чтобы увидеть число в виде двоичной строки, потребуется явное преобразование `number.toString` с указанием системы счисления. Можно не указывать или преобразовать неявно, или указать десятичную - будет десятичное.

### Битовые операции

JavaScript

```
var маска = 0b11001100;
маска.toString(2);
'11001100'
(~маска).toString(2);
'-11001101'
(маска | 0b110011).toString(2);
'11111111'
(маска & 0b110011).toString(2);
'0'
(маска & 0b110111).toString(2);
'100'
(маска ^ 0b110111).toString(2);
'11111011'
(маска >> 1).toString(2);
'1100110'
(маска << 1).toString(2);
'110011000'
(маска >>> 1).toString(2);
'1100110'
```

### Логические операции

JavaScript

```
// Отрицание

! true; // false
! false; // true
! (5 > 6); // true
// утверждение в скобках ложно,
его отрицание истинно. Заметим,
что отрицание для > это <=

// Логическое И
(1 < 3) && (3 < 4); // true
// оба утверждения истинны -
истина

// Логическое ИЛИ
var x = 5;
(x < -1) || (x > 1); // true
```

```
(-5 >>> 2).toString(2);  
'111111111111111111111111111110'
```

// Хотя бы одно, в данном случае -  
правое утверждение истинно,  
результат логического или истинный

Отметим, что изучение двоичной арифметики не входит в данный курс и оставляется на усмотрение читателя. Данный раздел программирования упростит глубокое понимание многих аспектов изучаемого предмета, однако не является обязательным и может быть опущен для экономии времени либо отложен на более поздний срок.

## Линейный алгоритм

Для того, чтобы написать программу, сначала нужно очень хорошо понять, как бы вы решали задачу, потом, как бы вы её объяснили человеку, который НЕ УМЕЕТ ДУМАТЬ САМ. Т.е. вообще. Ваш исполнитель умеет выполнять несколько (возможно, очень много!) различных действий различной степени примитивности, обладает абсолютной памятью (оставим пока в стороне случай, когда у компьютера закончилась память или место на диске), предельно честен, терпелив и послушен до педантизма, но не сообразит сам, что если сперва надо было напечатать 1, потом 2, а потом 3, то "и так далее до 10 подряд" надо напечатать 4, 5, 6, 7, 8, 9 и 10, если это не заложено в нём изначально или вами.

Кроме того, нельзя скатываться в решение части задачи за исполнителя. Старайтесь передавать только то, что известно изначально и способы вычисления, но, например, если известно, что времени прошло 2 дня, то так и писать, а не 48 часов, чтобы можно было при необходимости легко поменять на 3 марсианских дня, указав новую константу вместо 24 и новое количество дней вместо 3, а умножением пусть занимается исполнитель. Позже в целях оптимизации вы откажетесь от этого ограничения, но поначалу для повышения навыка написания алгоритмов лучше писать максимально подробно.

Кроме того, используйте только то, что уже было вами изучено. На начальных этапах крайне вредно находить в интернете решение, "которое работает", потому что в нём будут непонятные места и возникнет ощущение, что книга описывает язык непоследовательно, требуя дополнительных знаний и многочасовых поисков. А на самом деле, все задачи первых уроков, пока не оговорено обратное, могут быть решены СТРОГО с использованием того, что описано в книге даже без перехода по ссылкам дальше. Вам велено вывести в консоль все числа от 1 до 10, а цикл ещё не проходили? Печально, значит, злой преп хотел, чтоб вы написали 10 строк:

```
JavaScript
console.log(1);
console.log(2);
```

```
// Я их писать всё же не буду, потому что злой преп это я
console.log(10);
```

...а вовсе не стащили из интернета

```
JavaScript
for(let i = 0; i < 10;) console.log(++i);
```

Потому что ко второму фрагменту много вопросов. Почему let, а не var? Почему, если дописать ещё один console.log в конец этой строки, то он напечатает от 1 до 11, а не будет честно выводить все числа по 2 раза? И чем дальше, тем более сложные кусочки кода будут решать относительно простые задачи. Но только понимание основ позволит вам играть этим, как кубиками лего, не превращая программирование в алхимию.

Попробуем? На данном этапе вам доступны следующие кубики: создание переменных типа числа, большие числа, строки, логика. Также доступны некоторые действия над этими типами. Поменяйте значения двух переменных местами.

```
JavaScript
var миска = "Суп";
var тарелка = "Каша";
[k, m] = [m, k] // решение, которое ищется в интернете
```

Но никаких квадратных скобок пока не было! Что делать?

Сначала понять, как бы это сделали мы в реальности. Есть две тарелки, в одной суп, во второй - каша. Если мы не повар-жонглер 80-го уровня, то нам потребуется третья емкость...

```
JavaScript
var плошка = миска; // перелили суп во временную емкость
// миска = null; // можно ещё вымыть по дороге :)
миска = тарелка; // перелили кашу
тарелка = плошка; // перелили суп, куда планировали
```

Забавный момент, кстати, о “вымыть по дороге”: мы сейчас получим две переменные с одним значением - плошку и миску. Они указывают на один и тот же объект! Мы ведь не перелили, как бы сделали это в реальности, а скопировали ссылку. Теперь, думая, что плошка пустая, то есть равна 0, мы используем ее где-то в другом месте программы... А там суп! Для второго “переливания” это не имеет значения, потому что при присваивании миска = тарелка, суп из миски будет безжалостно выплеснут. Хорошо, что он одновременно ещё и в плошке 😊 Так что сравнение с супом довольно поверхностное, скорее можно представлять себе это как лазание по скале. Когда вы пристёгнуты одним карабином - плохо, но допустимо, когда отцепите оба - можно сорваться вниз (потерять

связь с объектом), а двумя и более - один всегда можно отцепить, чтобы использовать его для прикрепления к другой части скалы.

А теперь поменяйте местами две переменных типа BigInt, не используя третью переменную и читерское решение из интернета.

Сначала алгоритм: в одну прибавим вторую. Теперь из неё вычтем вторую и запишем во вторую. Во второй оказалось СТАРОЕ значение первой! Теперь всего лишь надо из первой вычесть новую вторую (старую первую) и записать в первую. Запишите это решение с помощью js.

JavaScript

```
var a = 5n;  
var b = 6n;  
a = a + b;    // сейчас в а лежит что? Правильно, 11. В b по-прежнему 6  
b = a - b;    // (5+6)-6 будет 5. ХОБА! В b "переехало значение из старого а!"  
a = a - b;    // (5+6)-5 = 6 (ведь на предыдущем шаге b стало равно пяти...)
```

Но и это ещё не вершина! Представьте себе, что есть операция, которая для двух целых чисел, каждое из которых 0 или 1, говорит, разные у нас значения или одинаковые? (закадровый смех) Зря смеетесь, я не о сравнении на равенство, а о битовом исключаящем ИЛИ. (XOR), пишется "галочкой, крышечкой" ^

Данная операция равна 1, когда числа разные, и 0, когда одинаковые. Самостоятельно убедитесь сперва в том, что для любой пары (0,0;0,1;1,0;1,1) замена будет выполнена верно, как теоретически, рассуждая об операции в уме, так и с помощью консоли или небольшого скрипта, а затем - для двух целых чисел.

Преимущество в том, что эта операция не изменяет общее занимаемое числом количество бит, тогда как сложение может привести к переполнению. Недостаток в том, в JavaScript битовые операции работают с целой частью числа, отбрасывая дробную.

JavaScript

```
var g = 9.8;  
console.log(g ^ 0); // исключаящее или отбросит дробную часть числа.  
// Осторожно! Это не округление и даже не "округление вниз", потому что для отрицательных чисел тоже будет отброшена дробная часть, число будет округлено вверх!
```

Многие считают, что алгоритм обмена с помощью третьей переменной значительно проще для придумывания и понимания. Да, да, я намекаю на то, что если вы поняли все три и сейчас проворчали "да, ладно, что там такого", то вы необыкновенно круты, не говоря уже о том, если вы их сейчас или когда-то придумали сами.

Может быть, тогда вы скажете, как, не используя знак > и строковые операции, сообщить, что число больше нуля? Или что одно число больше другого?

Вернемся к более простым случаям.

2024 год начался с понедельника. Запишите алгоритм нахождения дня недели для любого числа января.

Правильно! Вычисляем остаток от деления на 7. Если получился ноль, значит воскресенье. Если брать американскую или еврейскую неделю, которая начинается с воскресенья, то все почти хорошо. Но если мы хотим, чтоб номер дня недели начинался с единицы или с понедельника, придется подправить формулу. Подробнее о представлении даты и времени мы поговорим [на соответствующем уроке](#). А если бы год начинался не с понедельника? А если потребуется вычислить день недели в феврале?

Все полученные алгоритмы могут быть написаны исключительно последовательно, без возвратов (цикл) и ветвлений (а если, а если) с помощью формул. Такие алгоритмы называются линейными, от слова "линия".

## undefined, NaN и null

В JavaScript существует аж ТРИ РАЗЛИЧНЫХ способа указать отсутствие. Результат неверных математических вычислений записывается с помощью Not a Number. Например логарифм отрицательного числа, поскольку на множестве вещественных чисел решения нет, а number - это вещественное число. Заметим, что NaN не равен сам себе! К счастью, они изготовили встроенное решение этой проблемы: функцию isNaN:

```
JavaScript
console.log(NaN == NaN);
// false
console.log(NaN === NaN);
// false
console.log(isNaN(NaN));
// true
```

undefined (неопределенность) и null (отсутствие объекта) настолько похожи, что РАВНЫ, но не равны с учётом типа:

```
JavaScript
console.log(undefined == null);
```

```
// true  
console.log(undefined === null);  
// false
```

На начальном этапе изучения JavaScript получить это значение можно разве что специально, но в дальнейшем вам будут встречаться функции, которые возвращают null, когда не могут вернуть объект. Это будет подчеркивать отсутствие объекта. Нельзя вернуть undefined, потому что мы знаем, что объекта нет, и нельзя вернуть false, потому что это не логический тип данных. С типом Object мы будем знакомиться позже.



## Подведём итоги

Вы познакомились с понятием линейного алгоритма без ответвлений и вариаций и несколькими типами данных, которые называются примитивными. Это не означает, что они легки в усвоении или бесполезны, это означает, что ещё существуют типы данных, состоящие из других типов, их называют составными типами. В JS они представлены собирательным типом Object.

Существует 8 типов данных в JS, из них два содержат по одному значению, один - два значения, два типа очень похожи до тех пор, пока мы не коснемся слишком больших целых чисел, на два примитивных типа `typeof` возвращает 'object' (объект), а сам object - это просто портал в целый мир настоящих интересных сложных типов данных, и даже строка вполне была бы хороша, но нельзя заменить её частично, можно сделать новую строку. Symbol мы даже ещё начинали изучать, он приведен здесь для полноты таблицы.

название	Примеры	Пояснение
<a href="#">number</a>	0, 9.8, 6.67e-11, -273, 0xC0DE, 0b11010101, 0o, Number.EPSILON, Number.NaN, Number.Infinity	Числа
<a href="#">bigInt</a>	100n	
<a href="#">string</a>	"Мама", 'мыла', `раму`	Строки
<a href="#">boolean</a>	true, false	Логика
undefined	undefined	Неопределенность
<a href="#">object</a>	Date, [0, 'ноль', []], {'boy': "мальчик"}, document	Составные типы
	null	Отсутствие
symbol	<a href="#">Symbol</a> ('tag')	Уникальные идентификаторы

## Приоритет операций в JS

Зубрить приоритет операций - бессмысленная трата времени. Если вы часто пишете сложные выражения, будете пользоваться справочником и запоминание произойдет автоматически. Часто избежать сложностей позволяют круглые скобки.

Как пользоваться таблицей? Первым приоритетом идет точка, вторым - круглые скобки. Ну, вам же и так ясно, что мы сначала читаем, что `log` - это метод консоль, а только потом его вызываем? В таблице это формализовано.

Оператор	Пояснение
<code>.</code> <code>[]</code>	Обращение за методом класса
<code>()</code>	Вызов метода или функции
<code>-</code> <code>+</code> <code>++</code> <code>--</code> <code>typeof</code>	отрицание, инкремент
<code>*</code> <code>/</code> <code>%</code>	умножение, деление, остаток
<code>+</code> <code>-</code>	сложение, вычитание
<code>&lt;</code> <code>&lt;=</code> <code>&gt;</code> <code>&gt;=</code> <code>in</code>	сравнение, вхождение
<code>==</code> <code>!=</code> <code>===</code> <code>!==</code>	Равенство, неравенство
<code>&amp;</code>	битовое-и
<code>^</code>	битовое-исключающее-или
<code> </code>	битовое-или
<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code> <code>&lt;&lt;=</code> <code>&gt;&gt;=</code> <code>&gt;&gt;&gt;=</code> <code>&amp;=</code> <code>^=</code> <code> =</code>	присваивание

Таблица намеренно сокращена до тех операций, которые мы уже прошли.

## Вопросы по теме:

- 1) В каком случае программа не выполнит команду и выдаст ошибку?
  1. Сложить целое и дробное число
  2. Попробовать вывести в консоль строку нулевой длины (например, "")
  3. Разделить отрицательное число на "0"
  4. Сравнить прописную (большую) и строчную (маленькую) буквы
  5. Сложить 2 строки, в которых есть символы как латиницы, так и кириллицы.
- 2) Выберите из списка модифицирующие операторы
  1. Сложение
  2. Деление

3. Арифметическое присваивание
4. Логическое "и"
5. Получение остатка от деления

3) Немодифицирующий оператор:

1. Игнорируется программой
2. Не изменяет тип данных
3. Не изменяет значение переменной
4. Не выводит результат в консоль
5. Не записывает данные в файл

4) Какая запись не является числом?

1. 0xC0DE
2. 0o755
3. 0xM0DE
4. 0b11001100
5. 6.67e-11

5) Помните, в первом классе нас предупреждали, что нельзя складывать километры и килограммы? Как избежать этой ошибки в вашей программе:

1. При присвоении значений переменным проставить обозначения единиц (9 км, 5 кг)
2. Существуют ключевые имена переменных, учитывающих размерность данных
3. Выстроить логику программы так, чтобы эта ошибка не возникала
4. Обозначить единицы измерения латинскими символами в комментариях
5. Использовать только строковые переменные

6) Выберите истинные утверждения:

1. 5 > "5"
2. "Коля" + "Иванов" == "Коля Иванов"
3. 1 + "6" === "16"
4. 17 == "17"
5. 17 === "17"

7) Шаблон - это...

1. Заготовка текста программы, в который осталось только вставить числовые значения
2. Строковая переменная для вывода на экран комментариев к программе
3. Сообщение об ошибке с предложением вариантов ее исправления
4. Строка в обратных кавычках, в которую могут быть добавлены специальные символы, осуществляющие подстановку значений, которые неизвестны на этапе написания программы и будут получены в процессе её работы
5. Подсказка, возникающая в редакторе в процессе написания программы

8) Какой из алгоритмов для вычисления количества прожитых вами дней правильный?

1. Умножить количество полных лет на 365 и прибавить к результату сегодняшнюю дату
2. Умножить количество полных лет на 365 и прибавить к результату дату рождения
3. Разделить количество прожитых лет на 4, прибавить к результату количество прожитых лет, умноженное на 365, и количество дней, прошедших с ближайшего дня рождения
4. Умножить количество полных лет на 365, прибавить к результату количество високосных лет за этот период, и количество прожитых дней неполного года.
5. Умножить количество полных лет на 365, прибавить к результату количество високосных февралей за этот период, и количество прожитых дней неполного года.

9) Можно ли искать решение домашнего задания в Интернете?

1. Нет, потому что в Интернете много недостоверной информации
2. Можно, это экономит время и силы, а поиск - сам по себе уже хорошая тренировка
3. Нет, потому что велика вероятность, что даже правильное решение не будет отвечать требованиям урока
4. Можно, если я понимаю, как работает найденное решение и могу ответить на любые вопросы по нему
5. Нет, потому что решение, рекомендованное преподавателем - единственно верное

10) В каких случаях необходимо использовать тип BigInt:

1. Расстояние от Москвы до Владивостока в километрах
2. Количество молекул в стакане воды
3. Население Китая
4. Число Пи
5. Стоимость пригородного билета в рублях

## Задания для самостоятельной работы:

1. Какой тип данных стоит использовать для хранения (ответ обосновать):
  - a. Номера машины?
  - b. Адреса?
  - c. Номера дома?
  - d. Роста человека? Его возраста? Пола?
  - e. Наличия или отсутствия той или иной детали в компьютере?
  - f. Объема диска сервера?
  - g. Его объема оперативной памяти?
2. Запишите с помощью логики утверждение "если завтра выходной, я не пойду на работу либо если сегодня выходной, я сяду читать книжку"
3. Напишите решение задачи: "От дома до метро идти 20 минут. На метро ехать 14 остановок до пересадки и 8 после пересадки. На пересадку тратится 7 минут, между станциями ехать 2 минуты 30 секунд. От метро до места учёбы идти 5 минут. Тот же путь на машине можно преодолеть за полтора часа. На машине - быстрее?"

## Словарь урока

**number** - намбер - число, номер

**big** - биг - большой

**safe** - сейф - безопасный

**integer** - интеджер - целый (сокращенно int)

**float** - флоат - плавающий (вещественные или действительные числа с плавающей запятой)

**boolean** - булеан - от фамилии профессора Буля, разработавшего алгебру логики

**delete** - дэлит - удалить

**true** - тру - истина

**false** - фолс - ложь

**infinity** - инфинити - бесконечность

Теперь пора научить компьютер выбирать варианты поведения в зависимости от условий