Транскрипт урока

Конечно, компьютеры отлично справляются с вычислениями чисел. И когда дело касается простой математики, JavaScript довольно прямолинеен. Есть пять основных операций: сложение, вычитание, умножение, деление и остаток от деления. Ещё есть скобки, как и в обычной математике, которые помогают явно указывать последовательность вычислений.

Взгляните на это: 25 \* 91. 25 и 91 называются **операндами**, а звёздочка - **оператором** умножения.

Вот немного более усложнённый пример: ((12 \* 5) - 4) / 12.

В начале JavaScript производит умножение, затем вычитает 4, поскольку есть скобки, а потом делит результат на 12.

Все операторы здесь **инфиксные**: они находятся между операндами (в данном случае — между числами). Ещё есть **префиксные** операторы (например, знак минус, который обозначает отрицательное число: -5) и **постфиксные** (например, быстрое увеличение на один: x++). Мы изучим их позже.

В какой-то момент вы столкнётесь с необычной проблемой: если вы попытаетесь сложить 0.1 + 0.2 в JavaScript, результатом будет 0.30000-много-много-нулей-4, а не 0.3. Это потому что компьютеры хранят числа в другом формате. Глубоко внутри все числа — это множество единиц и нулей, подчинённых определённым правилам и это не лучший формат для хранения любых чисел.

Это может показаться нелепым — почему мы позволяем компьютерам использовать такую плохую систему? На самом деле она не настолько плохая или глупая. Такой формат удобен для определённого набора задач и не слишком удобен для другого.

Также в JavaScript есть несколько терминов, которые нужны для формулировки определённых значений: разделите положительное число на 0 и получится "Бесконечность" — Infinity; разделите отрицательное число на 0 и получится "-Бесконечность" — -Infinity. В своих программах вы можете использовать Бесконечности как числа с другими операторами. Например, с Бесконечностью можно производить сложение.

Иногда вычисления не производят конкретного числа. Разделите 0 на строку и получится что-то не числовое. Нельзя сказать, что это ничто, это… просто не число. У JavaScript есть термин для такого понятия — NaN, который образован из "Not a Number" (не число).

Как и Бесконечность, NaN можно использовать в вычислениях с другими математическими операторами. Но Not a Number как бы всех подводит: если он присутствует в вычислении, результатом всегда будет NaN.

Вот случайный вопрос: какого размера Марс? Его радиус — 3390 километров, он почти в два раза меньше Земли. Но мы, конечно, заинтересованы там жить, поэтому нам важно сколько у нас будет поверхности. Другими словами, какая площадь поверхности у Марса?

Может, вы помните формулу: площадь поверхности сферы равна 4πr2. r — это радиус, а π примерно 3.14.

Давайте вычислим это в JavaScript:

4 **\*** 3.14 **\*** 3390 **\*** 3390;

Теперь представьте как вычислить площадь поверхности другой планеты. Например, Меркурия:

4 **\*** 3.14 **\*** 2440 **\*** 2440;

Этот новый код точно такой же, как и предыдущий, изменился только радиус. Если мы продолжим в таком же стиле, нам понадобится писать значение π самим каждый раз. Так не пойдёт, мы не хотим повторений в своих программах.

Мы можем заставить компьютер "помнить", что такое π и использовать это в вычислениях. Такой механизм называется "константы". Давайте создадим новую константу со значением числа π:

**const** pi **=** 3.14;

const — это специальное ключевое слово, после него — идентификатор — название вашей константы, затем знак равенства и значение.

Теперь мы можем писать "pi" вместо того, чтобы вводить вручную 3.14.

4 **\*** pi **\*** 3390 **\*** 3390; *// surface area of Mars*

4 **\*** pi **\*** 2440 **\*** 2440; *// surface area of Mercury*

Кстати, двойной слеш и текст после него — это **комментарии**: JavaScript просто игнорирует их и они не влияют на работу кода. Мы пишем комментарии для себя и других людей, чтобы легче понимать код.

Давайте поместим площадь поверхности Марса в другую константу:

**const** surface **=** 4 **\*** pi **\*** 3390 **\*** 3390;

Теперь surface это другой идентификатор, в нём хранится **результат** вычисления. Тогда как это вычисление произвелось? Ну, во-первых, JavaScript должен вспомнить что такое pi, а потом заменить его на число, значение:

4 **\*** pi **\*** 3390 **\*** 3390;

4 **\*** 3.14 **\*** 3390 **\*** 3390;

А затем идут умножения слева направо, потому что у нас нет скобок:

4 **\*** 3.14 **\*** 3390 **\*** 3390;

12.56 **\*** 3390 **\*** 3390;

42578.4 **\*** 3390;

144340776;

Мы можем вывести результат на экран с console.log: console.log(surface). Заметьте, что в этот раз мы не писали кавычки. Мы не будем выводить слово "surface". Это не строка. Мы выводим значение константы, называемой "surface".

Теперь ваша очередь. Переходите к тестам и упражнению. Вы узнаете как много энергии в вашем теле, благодаря Эйнштейну.

Дополнение к уроку

Возведение в степень

Стандарт es7 вводит новый оператор \*\*, полезный при возведении в степень:

2 **\*\*** 4; *// 16*

3 **\*\*** 2; *// 9*

Другой способ возвести число в степень:

Math.pow(2, 4); *// 16*

Math.pow(3, 2); *// 9*

Почему константы?

Возможно у вас возник вопрос: "почему константы, а не переменные?". Константы гораздо проще, чем переменные. Они всегда однозначно определены и никогда не меняются. В школе и в ВУЗе, в математике и физике мы имели дело только с константами.

В будущих уроках вы увидите, что единственное место, где требуются переменные — это циклы. Практически во всех других случаях они не нужны, и их присутствие усложняет код (мы объясним это позже в курсе).

**const** pi **=** 3.14;

**const** e **=** 2.7;

**const** strange **=** pi **\*** e;

console.log(strange); *// => 8.478000000000002*

<https://repl.it/@hexlet/programming-basics-first-program-const>

Арность

Арность — это количество аргументов. Операторы +, -, \* и / применяются к двум операндам. В таком случае их арность составляет 2, поэтому мы можем называть эти операторы *бинарными*.

Есть ещё один оператор, который выглядит как двоичный минус, но он унарный (одинарный). Это когда - используется для обозначения отрицательного числа, например -327.

Ещё бывают *тернарные* (троичные) операторы, но вы их встретите позже.

Ассоциативность

Ассоциативность (или фиксированность) определяет, как операторы группируются при отсутствии скобок. Рассмотрим выражение a ~ b ~ c. Если у оператора ~ левая ассоциативность, это выражение будет трактоваться как (a ~ b) ~ c.

Undefined

Рассмотрим следующий код:

**const** a;

console.log(a);

Что в этом случае выведется на экран? Современный JavaScript вообще не позволит вам создать константу без значения, но если позволит, полученный console.log выведет undefined. Это специальный идентификатор.

Вы можете установить значение undefined сами, вот таким способом:

**const** a **=** **undefined**;

Но самому этого делать не стоит.

Выводы

* Математика в JavaScript выглядит и воспринимается как привычная математика:
  + Символы +, -, \*, / означают то, что вы думаете.
  + Ещё есть символ % — остаток от деления. Он вычисляет остаток от деления первого операнда на второй. Например, 11%5 это 1, а 10%2 это 0.
  + Есть Infinity и -Infinity.
  + Когда вычисление оказывается не числом, JavaScript использует NaN, который означает "не число" (Not a Number). Например: 0/"word" это NaN
    - Если в вычислении присутствует NaN, то результатом всегда будет NaN. Например: 120 + 5 / NaN это NaN
* Заставьте компьютер "помнить" что-нибудь, создав константу. Например: const age = 39;

#### **Дополнительные материалы**

1. [Floating Point Numbers - Computerphile. Отличное объяснение проблемы чисел с плавающей запятой (та странная проблема из видео-урока)](https://www.youtube.com/watch?v=PZRI1IfStY0)
2. [What Every Programmer Should Know About Floating-Point Arithmetic or Why don’t my numbers add up? Специальный сайт, посвященный проблеме чисел с плавающей запятой](https://floating-point-gui.de/)
3. [Decimal to IEEE754 number converter — стандарт, описывающий формат хранения чисел с плавающей запятой](https://www.h-schmidt.net/FloatConverter/IEEE754.html)
4. [Arithmetic operators in JavaScript / Mozilla Developer Network](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators)
5. [Infinity in JavaScript / Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Infinity)
6. [NaN in JavaScript / Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NaN)
7. [What Every Computer Scientist Should Know About Floating-Point Arithmetic: long and detailed explanation with formulas](https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html)
8. [Arity (Арность)](https://en.wikipedia.org/wiki/Arity)
9. [Operator associativity (Ассоциативность)](https://en.wikipedia.org/wiki/Operator_associativity)

#### **Дополнительные материалы**

1. [Floating Point Numbers - Computerphile. Отличное объяснение проблемы чисел с плавающей запятой (та странная проблема из видео-урока)](https://www.youtube.com/watch?v=PZRI1IfStY0)
2. [What Every Programmer Should Know About Floating-Point Arithmetic or Why don’t my numbers add up? Специальный сайт, посвященный проблеме чисел с плавающей запятой](https://floating-point-gui.de/)
3. [Decimal to IEEE754 number converter — стандарт, описывающий формат хранения чисел с плавающей запятой](https://www.h-schmidt.net/FloatConverter/IEEE754.html)
4. [Arithmetic operators in JavaScript / Mozilla Developer Network](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators)
5. [Infinity in JavaScript / Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Infinity)
6. [NaN in JavaScript / Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NaN)
7. [What Every Computer Scientist Should Know About Floating-Point Arithmetic: long and detailed explanation with formulas](https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html)
8. [Arity (Арность)](https://en.wikipedia.org/wiki/Arity)
9. [Operator associativity (Ассоциативность)](https://en.wikipedia.org/wiki/Operator_associativity)