**Арифметика**

**Унарный минус**

Изменение значения числа на противоположное.

**const** a = -**5**;

console.log(-a);

// -> 5

**Сумма**

Для записи суммы используется знак **+**.

**const** a = **5**;

console.log(a + **5**);

// -> 10

**Разность**

Для записи разности используется знак **-**.

**const** a = **5**;

console.log(a - **3**);

// -> 2

**Произведение**

Для записи произведения используется знак **\***.

**const** a = **5**;

console.log(a \* **3**);

// -> 15

**Деление**

Для записи деления используется знак **/**.

**const** a = **5**;

console.log(a / **2**);

// -> 2.5

**Остаток от деления**

Для записи операции нахождения остатка от деления используется знак **%**.

**const** a = **5**;

console.log(a % **2**);

// -> 1

**Возведение в степень**

Для записи операции возведения числа в степень используется оператор **\*\***.

**const** a = **5**;

console.log(a \*\* **3**);

// -> 125

**Инкремент**

Инкремент — это увеличение числа на 1.

**let** a = **5**;

console.log(++a);

console.log(++a);

// -> 6

// -> 7

**let** b = **5**;

console.log(b++);

console.log(b++);

// -> 5

// -> 6

**Обратите внимание!** Переменная, к которой применяется инкремент, изменяется.

Инкремент может быть записан как до числа, так и после. Если он записан до числа, сначала производится инкрементация (увеличение на 1) числа, а потом уже его запрос для вывода или последующих вычислений. А если после, то наоборот.

**let** a;

**let** b;

a = **3**;

b = **4**;

console.log(a++ \* b);

a = **3**;

b = **4**;

console.log(++a \* b);

// -> 12

// -> 16

**Декремент**

Декремент — операция уменьшения числа на 1.

Работает и используется по аналогии с инкрементом:

**let** a = **5**;

console.log(--a);

console.log(--a);

// -> 4

// -> 3

**let** b = **5**;

console.log(b--);

console.log(b--);

// -> 5

// -> 4

**Декремент**

Декремент — операция уменьшения числа на 1.

Работает и используется по аналогии с инкрементом:

**let** a = **5**;

console.log(--a);

console.log(--a);

// -> 4

// -> 3

**let** b = **5**;

console.log(b--);

console.log(b--);

// -> 5

// -> 4

**Приоритет операций**

Проанализируйте, какой результат будет выведен в консоль при выполнении выражения:

console.log(**10** + **5** \* **8** - **150** / **5** + **3**);

Посмотреть ответ

**23**

Если вы вспомнили верный порядок выполнения арифметических операций, скорее всего, у вас не возникло проблем при вычислении. В большинстве случаев порядок выполнения операций в *JavaScript* логичен и математически верен.

Например, в следующем выражении сначала производится возведение в степень, потом умножение:

console.log(**2** \* **5** \*\* **3** \* **2**);

// -> 500

Вы можете ознакомиться с полным перечнем приоритета операций, ссылка на него приложена в «[Подведении итогов](https://lms.skillfactory.ru/courses/course-v1:Skillfactory+PHP-1+17JUNE2020/courseware/9d5c9a19c997467eac806e141d38c761/b18a3cfd858645919e24511319c63928/8?activate_block_id=block-v1%3ASkillfactory%2BPHP-1%2B17JUNE2020%2Btype%40vertical%2Bblock%40bcf11edc5fb443349cf9983628b3bd2b)».

Для контроля приоритета можно использовать круглые скобки:

console.log((**2** \* **5**) \*\* (**3** \* **2**));

// -> 1000000

**toString(), parseInt(), parseFloat(), toFixed(), toPrecision()**

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| toString() | Преобразовывает число в текстовую форму. |
| parseInt() | Преобразовывает текст в целое число. |
| parseFloat() | Преобразовывает текст в вещественное число. |
| toFixed() | Форматирует число, используя запись с фиксированной запятой. |
| toPrecision() | Возвращает строку, представляющую объект Number с указанной точностью. |

**toString()**

Метод toString() позволяет преобразовать число в текстовую форму.

Синтакис следующий:

numObj.toString([radix])

numObj в этом случае — преобразуемое число, а radix — основание системы счисления, в которой требуется вывести число.

Если требуется вывести число в десятичной системе счисления, основание писать не требуется.

**const** a = **30**;

console.log(a.toString());

console.log(a.toString(**2**));

console.log(a.toString(**8**));

console.log(a.toString(**16**));

console.log(a.toString(**3**));

console.log(a.toString(**36**));

// -> 30

// -> 11110

// -> 36

// -> 1e

// -> 1010

// -> u

**Обратите внимание!** Значением для основания системы счисления должно быть целое число от 2 до 36:

**const** a = **30**;

console.log(a.toString(**37**));

// -> Uncaught RangeError: toString() radix argument must be between 2 and 36

Если как основание вводится не целое число, то будет взята лишь его целая часть:

**const** a = **7**;

console.log(a.toString(**2.71**));

// -> 111

**parseInt()**

parseInt() является обратной по отношению к toString() операцией, преобразование текстовых данных в число:

console.log(parseInt('30'));

console.log(parseInt('11110', **2**));

console.log(parseInt('36', **8**));

console.log(parseInt('1E', **16**));

console.log(parseInt('1010', **3**));

console.log(parseInt('U', **36**));

// -> 30

// -> 30

// -> 30

// -> 30

// -> 30

// -> 30

parseInt является методом объекта Number:

console.log(Number.parseInt('U', **36**));

// -> 30

Первым аргументом метода указывается строка, вторым — основание системы счисления. Если система счисления десятичная, указание её не требуется. В случае, если число нельзя распознать, возвращается NaN.

console.log(parseInt(''));

// -> NaN

parseInt(), как видно из названия, работает только с целыми числами.

**parseFloat()**

Функция parseFloat() принимает строку в качестве аргумента и возвращает десятичное число (число с плавающей точкой).

В отличие от метода parseInt(), parseFloat() работает только в десятичной системе счисления.

**toFixed()**

Аргумент toFixed() задаёт количество знаков после запятой. Если аргумент не указан, то берётся значение по умолчанию — ноль. Значением аргумента может быть число от 1 до 100.

**const** PI = **3.1415926**;

console.log(PI.toFixed());

console.log(PI.toFixed(**2**));

console.log(PI.toFixed(**4**));

console.log(PI.toFixed(**10**));

console.log((PI \* **100**).toFixed(**2**));

// -> 3

// -> 3.14

// -> 3.1416

// -> 3.1415926000

// -> 314.16

**toPrecision()**

Аргумент toPrecision() задаёт количество значимых знаков. Если аргумент не указан, то toPrecision() возвращает то же значение, что и toString(). Значением аргумента может быть число от 1 до 100.

**const** PI = **3.1415926**;

console.log(PI.toPrecision());

console.log(PI.toPrecision(**2**));

console.log(PI.toPrecision(**4**));

console.log(PI.toPrecision(**10**));

console.log((PI \* **100**).toPrecision(**2**));

// -> 3.1415926

// -> 3.1

// -> 3.142

// -> 3.141592600

// -> 310

**Неточные вычисления**

Попробуйте выполнить следующие выражение в консоли:

console.log(**0.2** + **0.1**);

Какой результат получим?

Посмотреть ответ

**0.30000000000000004**

Удивлены?

А теперь вернёмся в тему «[Запись чисел в памяти компьютера](https://lms.skillfactory.ru/courses/course-v1:Skillfactory+PHP-1+17JUNE2020/courseware/9d5c9a19c997467eac806e141d38c761/b18a3cfd858645919e24511319c63928/2?activate_block_id=block-v1%3ASkillfactory%2BPHP-1%2B17JUNE2020%2Btype%40vertical%2Bblock%40e92e3b7cbd144e87bedce97298044448)» и ещё раз прочитаем про хранение вещественных чисел.

Дело в том, что числа 0.2 и 0.1 в двоичной системе счисления можно записать лишь приблизительно. Они являются бесконечными периодическими дробями в ней, как, например, 1/3 в десятичной системе счисления (0,333333333...)

Двоичная запись 0.1:

**0.0001100110011001100110011001100110011001100110011001100**..

Двоичная запись 0.2:

**0.001100110011001100110011001100110011001100110011001100**..

Соответственно, и сумма в двоичной записи будет выглядеть не совсем точно, а последующее преобразование в десятичную систему счисления покажет потерю точности.

Ещё один пример:

console.log(**1**e8 + **1**e-**8**);

console.log(**1**e9 + **1**e-**9**);

// -> 100000000.00000001

// -> 1000000000

Такое поведение объясняется тем, что значимая часть числа ограничена по своему размеру: незначащие цифры отсекаются.

**В большинстве случаев подобные проблемы несущественны.** Однако стоит относиться к вещественным числам с определённой осторожностью.

Например, при сравнениях:

**const** a = **0.1** + **0.2**;

**const** b = **0.3**;

console.log(a == b);

// -> false

Если подобное сравнение будет условием выхода из цикла, цикл может стать бесконечным.

Или при нахождении суммы большого количество малых переменных: при складывании малых чисел с большими вы будете терять точность.