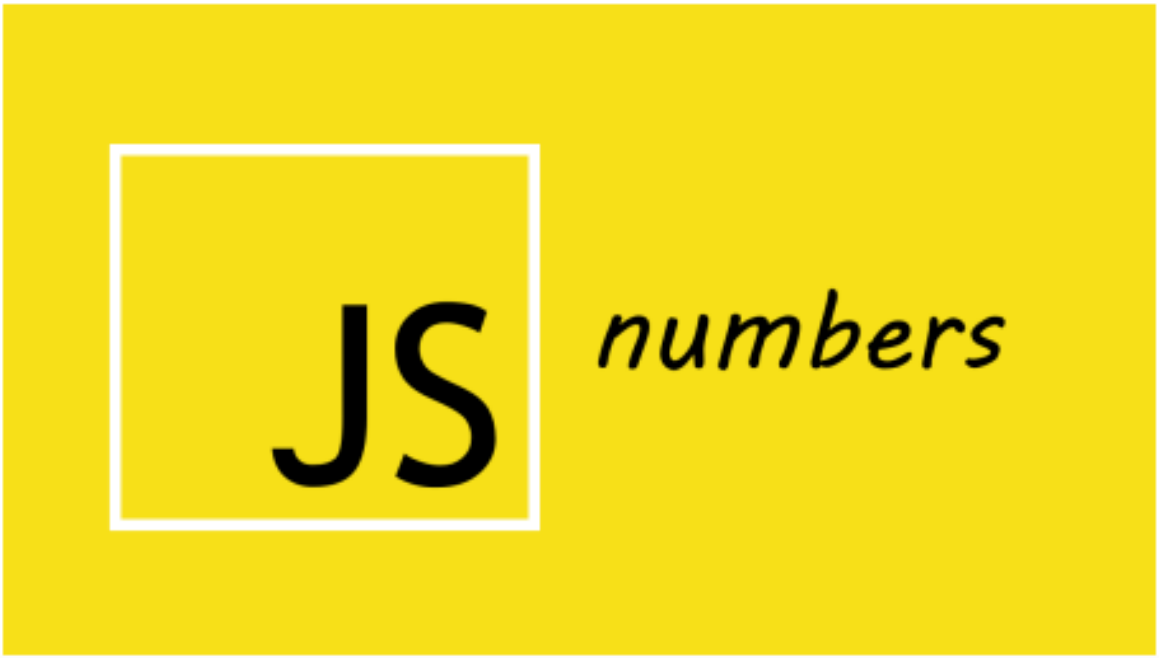
Лабораторная работа №5. Работа с числами в JavaScript



Содержание:

На этом занятии подробно рассмотрим числа, математические операторы, способы преобразования числа в строку и наоборот, а также много других важных моментов.

Числовой тип данных

В JavaScript в отличие от некоторых других языков имеется только один числовой тип `number`. В нём используется формат чисел с плавающей точкой двойной точности по стандарту IEEE 754.

То есть, в JavaScript нет отдельного формата для целых чисел. Все числа в JavaScript имеют плавающую точку.

Пример записи чисел:

```
let a = 50; // целое число
let b = 4.7; // число с дробной частью
let c = 2e3; // в экспоненциальной форме 2*10^3 (2000)
let d = 5.8e-2; // в экспоненциальной форме 5.8*10^-2 (0,058)
let e = 010; // в восьмеричной системы счисления
let f = 0xFF; // в шестнадцатеричной системе счисления
```

Как показано в примере, в JavaScript числа можно записывать в экспоненциальной форме, а также задавать в других системах счисления: в восьмеричной (впереди `0`) и шестнадцатеричной (впереди `0x`).

Узнать какой тип данных содержит переменная можно с помощью `typeof`:

```
let a = 7.29;
console.log(typeof(a)); // number
```

Кроме чисел, формат `number` также содержит специальные числовые значения:

- положительную бесконечность: `Infinity`;
- отрицательную бесконечность: `-Infinity`;
- не число: `NaN`.

Примеры выражений, результаты вычисления которых возвращают специальные числовые значения:

```
console.log(5 / 0); // Infinity
console.log(-7 / 0); // -Infinity
console.log(3 - '1rem'); // NaN
```

В JavaScript не возникают ошибки при выполнении математических операций. Если в результате вычисления получается очень большое число или очень маленькое выходящее за допустимые границы, то интерпретатор возвращает соответственно `Infinity` или `-Infinity`.

`NaN` возвращается, когда интерпретатор JavaScript не может вычислить математическую операцию. Например, когда мы попытаемся от числа `3` отнять строку `'1rem'`.

Числовой тип `number` является **примитивным типом**. Но при этом к числам можно применять методы как к объектам. Это осуществляется посредством обёртки `Number`. Она содержит дополнительные значения и методы для работы с числами:

```
let num = 10;
let str = num.toString(); // "10"
```

В этом примере мы применили к примитивному типу данных метод `toString()` как к объекту. При вызове метода числа автоматически оборачиваются в обёртку `Number()`, а у обёртки имеется метод `toString()`. Этот метод преобразует число в строку. Этот пример показывает, что числа ведут себя как объекты, хотя на самом деле ими не являются.

При этом использовать обёртку `Number` напрямую для создания чисел крайне не рекомендуется:

```
// так создавать числа не нужно
let num1 = new Number(10);
// всегда создавать числа нужно так
let num2 = 10;
console.log(typeof num1); // object
console.log(typeof num2); // number
```

В этом примере `num1` и `num2` имеют разные типы.

Если необходимо использовать методы на числе, то его нужно обернуть в круглые скобки или поставить точку дважды:

```
console.log((5).toFixed(2)); // 5.00
console.log(5..toFixed(2)); // 5.00
```

Кстати, на самом деле в JavaScript 2 числовых типа данных. Второй тип был добавлен в спецификацию ECMAScript 2020 (11 редакция). Называется он `bigint`. Этот тип предназначен для представления очень больших целых чисел и на практике используется только для чисел, которые слишком велики для представления с помощью `number`.

Ограничения по максимальному размеру и точность чисел

В JavaScript при сложении чисел с дробной частью можно получить не то число, которое ожидали.

```
const num = 0.2 + 0.4; // 0.6000000000000001
console.log(num === 0.6); // false
```

Погрешности происходят, потому что компьютер оперирует числами в двоичной системе счисления. Т.е. перед тем, как выполнить какие-то действия компьютер сначала должен преобразовать представленные в выражении числа в двоичную систему. Но, не любое десятичное число с дробной частью можно точно представить в двоичной системе счисления.

Например, число 0.25_{10} в двоичную систему преобразуется точно:

```
0.125 x 2 = 0.25 | 0
0.25 x 2 = 0.5 | 0
0.5 x 2 = 1 | 1

0.125₁₀ = 0.001₂
```

Например, число 0.2_{10} можно преобразовать в двоичную систему только с определённой точностью:

```
0.2 x 2 = 0.4 | 0
0.4 x 2 = 0.8 | 0
0.8 x 2 = 1.6 | 1
0.6 x 2 = 1.2 | 1
```

```
0.2 x 2 = 0.4 | 0
0.4 x 2 = 0.8 | 0
0.8 x 2 = 1.6 | 1
0.6 x 2 = 1.2 | 1
0.2 x 2 = 0.4 | 0
0.4 x 2 = 0.8 | 0
0.8 x 2 = 1.6 | 1
0.6 x 2 = 1.2 | 1
...

0.2₁₀ = 0.001100110011...₂
```

В результате эти погрешности скажутся при вычислении выражений и это нужно учитывать при написании кода в JavaScript.

Для этого вычислениях или при выводе чисел с дробной частью необходимо всегда указывать точность, с которой это необходимо делать.

Например, перепишем код, приведённый выше с определённой точностью используя метод `toFixed`:

```
const num = parseFloat((0.2 + 0.4).toFixed(2), 10); // 0.6
console.log(num === 0.6); // true
```

В этом примере, мы сначала с использованием метода `toFixed(2)` получили строковое представление числа, которое является результатом сложения чисел `0.2` и `0.4`, округленных до 2 знаков после точки. После этого с помощью метода `parseFloat` преобразовали текст в число и полученное значение присвоили переменной `num`.

Потеря точности может быть связана также с целыми числами, которые выходят за пределы безопасного диапазона:

```
const num = 9999999999999999;
console.log(num); // 10000000000000000
```

Получить минимальное и максимальное безопасное целое число можно с помощью констант:

```
console.log(Number.MAX_SAFE_INTEGER); // 9007199254740991
console.log(Number.MIN_SAFE_INTEGER); // -9007199254740991
```

Теперь, если проверим число, приведённое выше, то увидим что оно больше безопасного:

```
const num = 9999999999999999;
console.log(num > Number.MAX_SAFE_INTEGER); // true
```

Кроме этого, проверить находится ли целое число в пределах безопасного диапазона можно посредством метода `isSafeInteger`:

```
const num = 9999999999999999;
console.log(Number.isSafeInteger(num)); // false
```

Основные арифметические операторы

В JavaScript имеются следующие математические операторы:

- сложение: `+`;
- вычитание: `-`;
- умножение: `*`;
- деление: `/`;
- остаток от деления: `%`;
- возведение в степень: `**`;
- увеличение значения переменной на 1: `++`;
- уменьшение значения переменной на 1: `--`.

Пример:

```
let num;
num = 6 + 3; // 9
num = num - 5; // 4
num = 4 * 7; // 28
num = 12 / 2; // 6
num = 11 % 3; // 2 – остаток от деления
```

Операторы `++` и `--` можно располагать как перед переменной, так и после неё:

```
let num = 3;
console.log(num++); // 3
console.log(++num); // 5
```

Когда мы располагаем оператор `++` после переменной `num`, то сначала возвращается значение переменной, и только потом увеличивается её значение на `1`.

В другом случае, когда оператор `++` идёт до переменной num, то сначала увеличивается значение этой переменной на `1`, и только потом возвращается её значение.

Оператор `--` действует аналогично:

```
let num = 3;
console.log(num--); // 3
console.log(--num); // 1
```

Кроме этого имеются комбинированные операторы: `+=`, `-=`, `*=`, `/=`, `%=` и `**=`:

```
let num1 = 3;
let num2 += 6; // 9
```

Для **сравнения чисел в JavaScript используются следующие операторы**:

- равенство с учетом типа: `===`, с попыткой преобразования: `==`;
- не равенство с учетом типа: `!==`, с попыткой преобразования: `!=`;
- больше: `>`, больше или равно: `>=`;
- меньше: `<`, меньше или равно: `<=`.

Сравнение чисел:

```
console.log(2 > 3); // false
console.log(5 >= 3); // true
```

При сравнении чисел необходимо учитывать точность:

```
console.log((0.2 + 0.4) === 0.6)); // false
```

Способ сравнить число до двух знаков после плавающей точки:

```
console.log((0.2 + 0.4).toFixed(2) === (0.6).toFixed(2)); // true
```

Преобразование строки в число и наоборот

1. Преобразование строки в число

Явно привести строку в число можно разными способами:

1.1. Использовать унарный оператор `+`, который необходимо расположить перед значением.

```
console.log(+'7.35'); // 7.35
console.log(+'строка'); // NaN
```

Этот способ пренебрегает пробелами в начале и конце строки, а также `\n` (переводом строки).

```
console.log(+' 7.35 '); // 7.35
console.log(+'7.35 \n '); // 7.35
```

При использовании оператора `+` обратите внимание, что `''`, `' '` и `'\n'` преобразуется в число `0`. Значение `null` и логические значения приводятся к следующим числам:

```
console.log(+null); // 0
console.log(+true); // 1
console.log(+false); // 0
console.log(+' '); // 0
```

1.2. С помощью глобальной функции `parseInt()` или метода `Number.parseInt()`. Они предназначены для преобразования **значения, переданного им на вход, в целое число**. В отличие от использования унарного оператора `+`, эти методы позволяют преобразовать строку в число, в которой **не все символы являются цифровыми**. Эти методы преобразовывают строку, начиная с первого символа. И как только встречают символ, не являющийся цифровым, они останавливают дальнейший разбор строки и возвращают полученное число:

```
console.log(parseInt('18px')); // 18
console.log(Number.parseInt('18px')); // 18
console.log(parseInt('33.3%')); // 33
console.log(Number.parseInt('33.3%')); // 33
```

Эти функции могут работать с разными системами счисления (двоичной, восьмеричной, десятичной, шестнадцатеричной). Основание системы счисления передаётся в вызов `parseInt()` или `Number.parseInt()` в качестве второго аргумента.

```
console.log(parseInt('18px', 10)); // 18
console.log(parseInt('33.3%', 10)); // 33
console.log(parseInt('101', 2)); // 5
console.log(parseInt('B5', 16)); // 181
```

Кроме этого рекомендуется всегда указывать основание системы счисления и не полагаться на значение, которое оно имеет в том или ином браузере по умолчанию.

1.3. Для преобразования строки в число с плавающей точкой в JavaScript имеются методы `parseFloat()` и `Number.parseFloat()`. Друг от друга они ничем не отличаются, просто последний был добавлен в язык немного позднее:

```
let num = parseFloat('33.3%'); // 33.3
num = Number.parseFloat('33.3%'); // 33.3
num = parseFloat('3.14'); // 3.14
num = parseFloat('314e-2'); // 3.14
num = parseFloat('0.0314E+2'); // 3.14
```

Метод `parseFloat` в отличие от `parseInt` всегда рассматривает строку как число в десятичной системе счисления. Указать ему другую систему счисления нельзя.

2. Преобразование числа в строку.

Превратить число в строку можно с помощью метода `toString()`.

```
let str = (12.8).toString(); // '12.8'
```

Дополнительно можно передать систему счисления с учётом которой необходимо явно привести число к строке:

```
const num = 255;
const str = num.toString(16); // 'ff'
```

Проверки: isFinite, isNaN, isInteger

1. `Number.isFinite`.

Этот метод позволяет проверить, является ли значение конечным числом. В качестве результата `isFinite` возвращает `true`, если значение является конечным числом. В противном случае `false`:

```
console.log(Number.isFinite(73)); // true
console.log(Number.isFinite(-1 / 0)); // false
console.log(Number.isFinite(Infinity)); // false
console.log(Number.isFinite(NaN)); // false
console.log(Number.isFinite('20')); // false
```

Кроме `Number.isFinite` в JavaScript имеется также глобальная функция `isFinite`. Она в отличие от `Number.isFinite` выполняет проверку с учетом принудительное приведение переданного ей аргумента к числу:

```
console.log(Number.isFinite('20')); // false
console.log(isFinite('20')); // true
```

2. `Number.isNaN`.

Этот метод объекта `Number` предназначена для определения того, является ли значение `NaN`. Если это так, то `isNaN` возвращает `true`. В противном случае – `false`:

```
console.log(Number.isNaN(NaN)); // true
console.log(Number.isNaN(28)); // false
console.log(Number.isNaN('')); // false
```

Кроме `Number.isNaN` имеется также глобальный метод `isNaN`, он выполняет проверку с учетом приведения указанного типа данных к числу:

```
console.log(isNaN(NaN)); // true
console.log(isNaN('25px')); // true
console.log(isNaN(25.5)); // false
console.log(isNaN('25.5')); // false
console.log(isNaN('')); // false, т.к. один или несколько пробелов преобразуется к 0
console.log(isNaN(null)); // false
console.log(isNaN(true)); // false, т.к. значение true преобразуется к 1
```

3. Как проверить, является ли значение числом?

В JavaScript имеется метод, который позволяет определить, является ли значение целым числом. Называется он `isInteger`:

```
Number.isInteger('20'); // false
Number.isInteger(20); // true
```

В этом примере для `'20'` мы получили `false`, т.к. данное значение является строкой.

Если вам нужно проверить является ли значение числом, при это не важно целям или с плавающей точкой, то можно написать, например, вот такую функцию:

```
// isNumeric
const isNumeric = (value) => {
  return typeof(value) === 'number' && isFinite(value) && !isNaN(value);
}
```

Эта функция в зависимости от результата возвращает `true` или `false`.

После этого её можно использовать для проверки следующим образом:

```
const num1 = '12px';
const num2 = 23;
console.log(isNumeric(num1)); // false
console.log(isNumeric(num2)); // true
```

Если нужно проверить строковое значение, то его предварительно нужно преобразовать в строку, например, с помощью метода `parseFloat`:

```
const str1 = '';
const str2 = '20px';
console.log(isNumeric(parseFloat(str1))); // false
console.log(isNumeric(parseFloat(str2))); // true
```

При проверке строки `'20px'` мы получили `true`, потому что метод `parseFloat('20px')` возвращает нам число `20`.

Округление

Округление чисел в JavaScript можно выполнить разными способами.

1. С помощью специальных, предназначенных для этой задачи, методов:

- `Math.floor` – до ближайшего целого в меньшую сторону;
- `Math.ceil` – до ближайшего целого в большую сторону;
- `Math.round` – в большую сторону, если дробная часть `>=` 0.5; иначе в меньшую сторону;
- `Math.trunc` – путём отбрасывания дробной части;

```
let num = Math.floor(7.8); // 7
num = Math.ceil(7.2); // 8
num = Math.round(7.5); // 8
num = Math.trunc(7.9); // 7
```

2. Посредством `toFixed()`.

Этот метод округляет дробную часть числа до заданной точности, но результат возвращает в виде строки:

```
console.log((7.987).toFixed(2)); // "7.99"
console.log((7.987).toFixed(5)); // "7.98700"
// для дополнительного преобразования в число можно использовать метод parseFloat
console.log(parseFloat((7.987).toFixed(2))); // 7.99
console.log(parseFloat((7.987).toFixed(5))); // 7.987
```

3. Через `toPrecision()`.

Данный метод возвращает число в строковом формате, округленное с указанной точностью. При этом он может округлить целую и дробную часть числа. При этом значение может быть представлено как с плавающей точкой, так и в экспоненциальной форме:

```
let value = (1001).toPrecision(2); // "1.0e+3"
value = (1001).toPrecision(5); // "1001.0"
value = (12.4).toPrecision(1); // "1e+1"
value = (12.4).toPrecision(2); // "12"
value = (12.4).toPrecision(3); // "12.4"
value = (12.4).toPrecision(5); // "12.400"
```

4. С использованием логического оператора `~` или `^`:

```
// посредством двойного логического НЕ
console.log(~~7.9); // 7
// посредством использования логического ИЛИ
```

```
console.log(7.9 ^ 0); // 7
```

Форматирование чисел

В JavaScript вывести число в соответствии с региональными стандартами (языковыми настройками операционной системы) позволяет метод `toLocaleString()`.

В соответствии с настройками установленными в системе:

```
const num = 345.46;
const value = num.toLocaleString(); // "345,46"
```

Явно указав регион:

```
const value = (108.1).toLocaleString('ru-RU'); // "108,1"
```

Для отображения денежных величин:

```
let value = (2540.125).toLocaleString('ru-RU', { style: 'currency', currency: 'RUB' }); // "2 540,13 ₽"
value = (89.3).toLocaleString('ru-RU', { style: 'currency', currency: 'USD' }); // "89,30 $"
value = (2301.99).toLocaleString('ru-RU', { style: 'currency', currency: 'EUR' }); //"2 301,99 €"
```

В процентном формате:

```
const value = (0.45).toLocaleString('ru-RU', { style: 'percent' }); // "45 %"
```

С разделением групп разрядов (свойство `useGrouping`):

```
const value = (125452.32).toLocaleString('ru-RU', { useGrouping: true }); // "125 452,32"
```

С указанием числа десятичных знаков:

```
const value = (1240.4564).toLocaleString('ru-RU', { minimumFractionDigits: 2, maximumFractionDigits: 2 }); //"1 240,46"
```

alert, prompt и confirm - диалоговые окна в JavaScript

МЕТОД ALERT()

Функция `alert()` предназначена для вывода в браузере **предупреждающего модального диалогового окна с некоторым сообщением и кнопкой «ОК»**. При его появлении **дальнейшее выполнение кода страницы прекращается** до тех пор, пока пользователь не закроет это окно. Кроме этого, оно также блокирует возможность взаимодействия пользователя с остальной частью страницы. Применение этого окна в основном используется для вывода некоторых данных при изучении языка JavaScript, в реальных проектах команда `alert()` не используется.

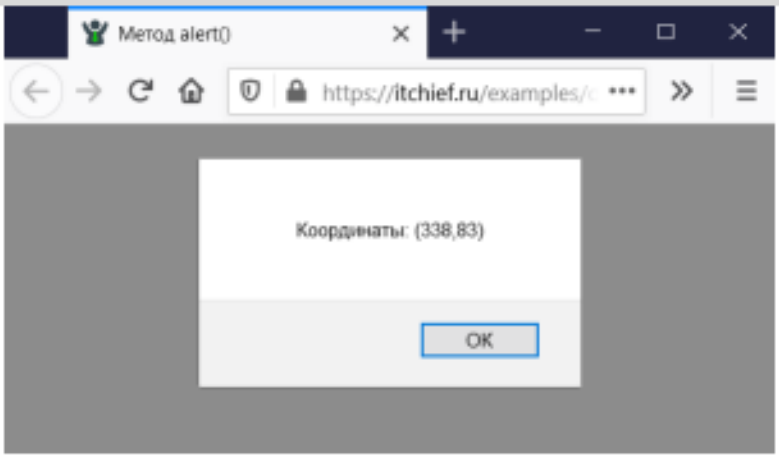
Синтаксис метода `alert()`:

```
// message - текст сообщения
alert(message);
```

Метод `alert()` имеет один аргумент (`message`) - текст сообщения, которое необходимо вывести в модальном диалоговом окне. В качестве результата `alert()` ничего не возвращает.

Например, выведем при клике в диалоговое окно alert координаты курсора:

```
// es6
document.addEventListener('click', (e) => {
alert(`Координаты: (${e.clientX},${e.clientY})`);
});
// es5
document.addEventListener('click', function (e) {
alert('Координаты: (' + e.clientX + ',' + e.clientY + ')');
});
```



Если alert сообщение нужно вывести на нескольких строках, то в этом случае следует воспользоваться «символом перевода строки», который в JavaScript записывается как `\n`:

```
// перенос строки в alert
alert('Строка 1\nСтрока 2');
```

МЕТОД PROMPT()

Метод `prompt()` предназначен для вывода диалогового окна с сообщением, текстовым полем для ввода данных и кнопками «OK» и «Отмена». Это окно предназначено для запроса данных, которые пользователю нужно ввести в текстовое поле.

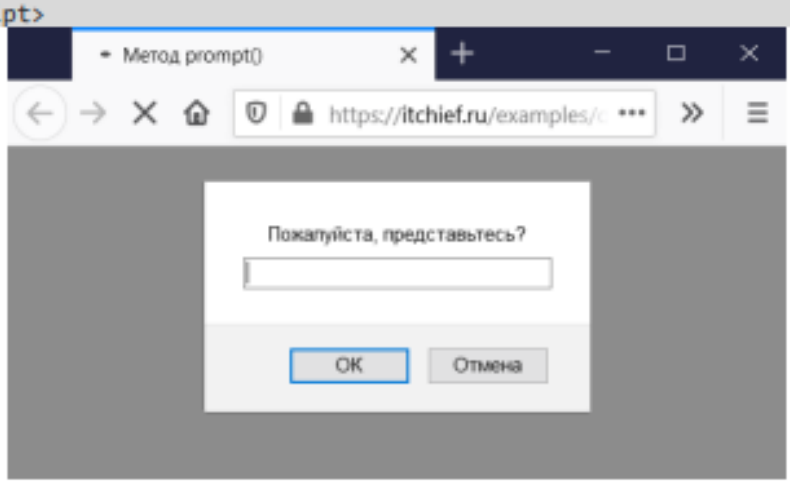
Синтаксис:

```
// message - текст сообщения (является не обязательным), предназначено для
//информирования пользователя о том, какие данные у него запрашиваются
// default - начальное значение для поля ввода, которое будет по умолчанию в нём
//отображаться (является не обязательным)
const result = prompt(message, default);
```

В переменную `result` возвращается значение введённое пользователем или `null`. Если пользователь не ввёл данные (поле ввода пустое) и нажал на «ОК», то в `result` будет находиться пустая строка.

Например, запросим имя пользователя при входе его на сайт с помощью prompt, а затем выведем его в элемент `#welcome`:

```
<div id="welcome"></div>
<script>
const name = prompt('Пожалуйста, представьтесь?');
if (name) {
  document.querySelector('#welcome').innerHTML = `<b>${name}</b>, добро пожаловать на сайт!`;
} else {
  document.querySelector('#welcome').innerHTML = '<b>Гость</b>, добро пожаловать на сайт!';
}
</script>
```



МЕТОД CONFIRM()

Метод `confirm()` объекта `window` применяется для вывода модального диалогового окна с сообщением и кнопками «ОК» и «Отмена». Оно обычно используется для запроса у пользователя разрешения на выполнение того или иного действия.

Синтаксис метода `confirm()`:

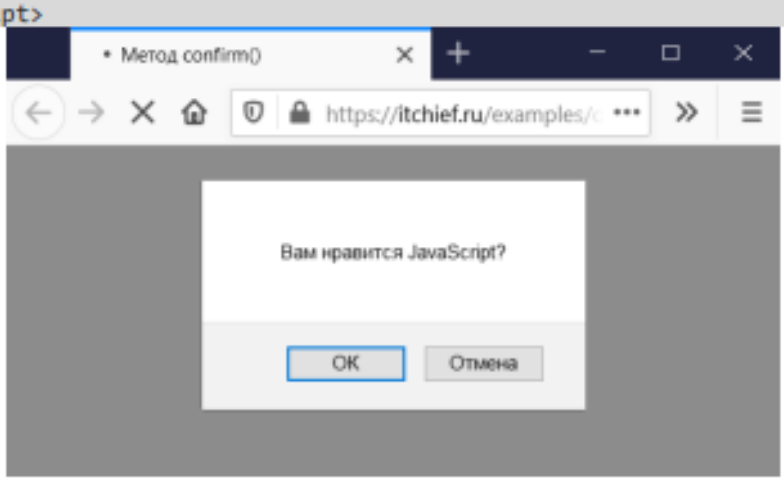
```
// question - текст сообщения (вопроса)
const result = confirm(question);
```

В переменную `result` возвращается:

- `true` - если пользователь нажал на кнопку «ОК»;
- `false` - в остальных случаях.

Например, выведем в элемент `#result` текст в зависимости то того на какую кнопку нажал пользователь в диалоговом окне confirm:

```
<div id="result"></div>
<script>
// es6
const result = confirm('Вам нравится JavaScript?');
if (result) {
document.querySelector('#result').textContent = 'Вы ответили, что Вам нравится
JavaScript';
} else {
document.querySelector('#result').textContent = 'Вы ответили, что Вам не нравится
JavaScript';
}
</script>
```



Примеры

1. Пример, в котором напишем две функции **для проверки числа соответственно на четность и не четность**. А затем используем их в коде:

```
// функция для проверки числа на чётность
const isEven = (value) => {
  return value % 2 === 0;
}
// функция для проверки числа на нечётность
const isOdd = (value) => {
  return Math.abs(value % 2) === 1;
}

const value = 20;
if (Number.isInteger(value) && isEven(value)) {
  console.log(`Число ${value} чётное!`);
} else if (Number.isInteger(value) && isOdd(value)) {
  console.log(`Число ${value} не чётное!`);
} else {
  console.log(`Значение ${value} не является целым числом!`);
}
```

2. Пример на JavaScript в котором получим **простые числа** от 2 до 100:

```
// функция, определяющая является ли число простым
const isPrime = (value) => {
  if (isNaN(value) || !isFinite(value) || value % 1 || value < 2) {
```

```
    return false;
  }
  const max = Math.floor(Math.sqrt(value));
  for (let i = 2; i <= max; i++) {
    if (value % i === 0) {
      return false;
    }
  }
  return true;
}
// массив, который будет содержать простые числа от 2 до 100
const primNums = [];
for (let i = 2; i <= 100; i++) {
  if (isPrime(i)) {
    primNums.push(i);
  }
}
// выведем в консоль значение переменной primNums
console.log(primNums);
```

3. Пример, в котором вычислим **целую и дробную часть числа**:

```
const num = 7.21;
// целая часть числа
let intNum = Math.floor(num); // 7
// дробная часть числа
let fractNum = num % 1; // 0.20999999999999996
// с точностью до 2 знаков
fractNum = parseFloat((num % 1).toFixed(2)); // 0.21
// 2 способ
fractNum = num - Math.floor(num); // 0.20999999999999996
```

В этом примере, получение целой части числа выполняется посредством `Math.floor()`, а дробной части с помощью получения остатка от деления на `1` или вычитания числа от его целой части.

4. Пример, в котором определим, **делится ли число нацело**, используя оператор `%`:

```
const num = 9;
// если остаток от деления на 3 значения переменной num равен 0, то, да, число
целое; иначе нет
if (num % 3 === 0) {
  console.log(`Число ${num} делится на 3`);
} else {
  console.log(`Число ${num} не делится на 3 без остатка`);
}
```