# Работа с числами

- Числа
- Объект Number
- Методы экземпляра объекта Number
- Немного об объектах и автомобилях
- Статические методы объекта Number
- Глобальные функции
- Преобразования чисел

## Числа

В отличие от многих других языков программирования JavaScript не определяет множество различных числовых типов вроде **целочисленные** без знака, с **плавающей** точкой, **короткие**, **длинные** и т.п.

В современном JavaScript существует два типа чисел:

- Тип данных **number** использует числа с плавающей точкой двойной точности (double precision floating point numbers), а также специальные значения **Infinity**, **-Infinity** и **NaN**.
- Тип **BigInt** позволяет работать с целыми числами произвольной длины. BigInt числа нужны достаточно редко.

В JavaScript нет отдельного формата для целых чисел, все числа в JavaScript имеют плавающую точку. Тип **number** представлен в виде 64-битного формата. Формат хранит произвольное число в виде трёх значений: 1 бит на **знак** числа, 52 бита **значения** числа и ещё 11 бит **местоположения** точки.

С таким подходом можно эффективно хранить значения в диапазоне от $-(2^{53} - 1)$ до $2^{53} - 1$. Из-за того, что положение точки в числе хранится отдельным значением, формат и называется **числом с плавающей точкой**.

Проблема этого представления в том, что оно не может представить числа абсолютно точно, а только с некоторой погрешностью.

### Форма записи чисел

Для записи чисел используются **цифры**, для разделения **целой** и **десятичной** части используется **точка**:

```
let a = 50; // 50

let b = 4.7; // 4.7

let c = .7; // 0.7
```

Можно использовать **экспоненциальную** запись.

```
let d = 2e3; // 2000 (2*10^3)

let e = 5.8e-2; // 0,058 (5.8*10^-2)
```

Числа так же могут быть представлены в **двоичном** (0b), **восьмеричном** (0o) или **шестнадцатеричном** (0x) виде.

> ***Важно***. При выводе на экран такие числа будут автоматически преобразованы в **десятичную систему счисления**.

**example_1**. Запись чисел в различных системах счисления

```
<script>

    // двоичная с/с

    const binary = 0b11;

    console.log(binary); // 3

    // восьмеричная с/с

    const octal = 0o77;

    console.log(octal); // 63
```

```
    // шестнадцатеричная с/с
    const hexadecimal = 0xFF;
    console.log(hexadecimal); // 255
</script>
```

### Сложение чисел и строк

JavaScript использует оператор "+" как для **сложения** чисел, так и для **объединения** строк (**конкатенации**). Числа **складываются**, строки **объединяются**.

Если вы складываете **два числа**, результатом будет число:

```
var x = 10;
var y = 20;
console.log(x + y); // 30
```

Если вы складываете **две строки**, результатом будет объединенная строка:

```
var x = "10";
var y = "20";
console.log(x + y);  // "1020"
```

Если вы складываете **число** и **строку**, результатом будет объединенная строка:

```
var x = 10;
var y = "20";
console.log(x + y);  // "1020"
```

Обычной ошибкой является в следующем примере ожидать, что результатом будет 30:

```
var x = 10;
```

```
var y = 20;

console.log("Результат: " + x + y); // "Результат: 1020"
```

Ошибкой является в следующем примере ожидать, что результатом будет 102030. Интерпретатор JavaScript обрабатывает код слева направо. Таким образом, сначала будет вычислена сумма 10 + 20, так как **x** и **y** числа. А затем будет проведена конкатенация строк 30 + "30", потому что **z** — строка.

```
var x = 10;

var y = 20;

var z = "30";

console.log(x + y + z); // "3030"
```

### Числовые строки

Строки JavaScript могут содержать числовой контент:

```
var x = 100;  // x - число

var y = "100";  // y - строка
```

JavaScript будет стараться конвертировать строки в числа во всех арифметических операциях. Это сработает:

```
var x = "100";

var y = "10";

console.log(x / y);  // 10
```

Это тоже сработает:

```
var x = "100";

var y = "10";

console.log(x * y);  // 1000
```

И это сработает:

```
var x = "100";
var y = "10";
console.log(x - y);  // 90
```

А вот это не сработает. JavaScript использует оператор "+" для **конкатенации** строк.

```
var x = "100";
var y = "10";
console.log(x + y);  // "10010"
```

В качестве первого демонстрационного примера выполним расчет выражения, состоящего из числовых примитивов разных систем счисления.

$(--x ** 011_2 + 67_8) / 2D_{16} \% 1000_2 + 33_{16} * 16_8 / ++x$

где: x = 6

**example_2**. Составление выражений с числами разных систем счисления

```
<script>
    // определение числовой переменной
    let x = 6;
    // составление выражения с числами разных систем счисления
    let num = (--x ** 0b011 + 0o67) / 0x2D % 0b1000 + 0x33 *
    0o16 / ++x;
    // вывод результата
    console.log(num); // 123
</script>
```

# Объект Number

Обычно в JavaScript работают с **примитивным числовым типом** (**литералом**). У примитивных числовых значений, вроде 3.14 или 124, не может быть свойств или методов (потому что они не объекты).

```
let primitive = 58;

console.log(primitive); // 58
```

Однако, в JavaScript все устроено так, что даже у примитивных значений доступны свои свойства и методы, потому что JavaScript интерпретирует любые значения, как объекты с выполняемыми свойствами и методами. Происходит это с помощью объекта **Number**.

**Number** — это **обёртка над примитивным числовым типом**, которая содержит встроенные свойства и методы, предоставляющие дополнительные возможности работы с числами.

> ***Важно***. Обёртка содержит **дополнительные свойства и методы для работы с числами**. Они не входят в стандарт типа данных "число" и поэтому выделены в отдельный модуль.

Например, **Number** позволяет использовать методы на числовых примитивах:

- проверка на специальные значения num.**isFinite**();
- конвертирование в строку num.**toString**();
- и др.

**Обёртка используется автоматически** и не требует дополнительной работы от программиста. JavaScript сам оборачивает число, когда программист вызывает метод или свойство на числовом примитиве.

```
// вызов методов на примитиве автоматически обернёт его в
Number()

let primitive = 5.001;
```

```
console.log(primitive.toString()); // '5.001'
```

В этом примере мы применили к **примитивному** типу данных метод **toString**() как к объекту. Пример показывает, что числа ведут себя как объекты, хотя на самом деле ими не являются.

> ***Важно***. При вызове метода для числа, обертка Number автоматически преобразует число к типу **объект**. У объекта имеются встроенные свойства и методы.

При желании определить **числа** как **объекты** можно **самостоятельно**. Объект **Number** создаётся через конструктор **Number**().

```
// обернуть числа можно вручную, вызвав конструктор Number()
let primitive = 50;
const num = new Number(100);
```

В этом случае переменные primitive и num будут разных типов. Проверить, какой тип данных содержит переменная можно с помощью **typeof**.

**example_3**. Числа могут быть объектами

```
<script>
    var primitive = 123;
    var num = new Number(123);
    console.log(typeof primitive); // number
    console.log(typeof num); // object
</script>
```

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

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

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

Использовать обёртку Number напрямую для создания чисел крайне не рекомендуется. Это замедляет скорость выполнения скрипта. Кроме этого, ключевое слово **new** в данном случае усложняет код и может привести к **неожиданным результатам**.

Например:

При использовании оператора сравнения "==", одинаковые числа равны. Однако, при использовании оператора сравнения "===", одинаковые числа не будут равными, потому что оператор "===" ожидает совпадения как по **значению**, так и по **типу**.

**example_4**. Сравнение и строгое сравнение чисел

```
<script>
    var x = 27;
    var y = 27;
    console.log(x == y); // true
    console.log(x === y); //true
    console.log("----------");
    x = 31;
    y = new Number(31);
    console.log(x == y); // true
    console.log(x === y); // false
    // x и y равны по значению, но не равны по типу (number и
    object)
</script>
```

Ситуация может быть еще хуже. Объекты не сравниваются, сравнение двух объектов JavaScript **всегда возвращает ложь** (false).

**example_5**. Сравнение чисел объявленных объектами

```
<script>

    var obj1 = new Number(100);

    var obj2 = new Number(100);

    console.log(obj1 === obj2); // false, obj1 и obj2 разные
    объекты

</script>
```

### Специальные значения

Кроме чисел, тип **number** также содержит специальные числовые значения:

- **Infinity** – положительная бесконечность;
- **–Infinity** – отрицательная бесконечность;
- **NaN** – не число.

### Infinity (Бесконечность)

В JavaScript не возникают ошибки при выполнении математических операций.

Бесконечности используются, чтобы определить результат некоторых арифметических операций. Например, деление на ноль в JavaScript вернёт бесконечность:

```
console.log(5 / 0); // Infinity
```

Если попытаться создать число, которое находится вне диапазона доступных чисел, результатом будет тоже бесконечность:

```
console.log(1e999); // Infinity
```

### NaN (Не число)

Значение **NaN** используется, чтобы сообщить об операции, результатом которой оказалось не число. В JavaScript существует **пять операций**, которые могут вернуть **NaN**:

- ошибка **парсинга** числа (например, при попытке превратить строку в число);
- результат математической операции не находится в диапазоне **действительных** чисел (например, взятие корня от -1);
- один из операндов в арифметической операции — **NaN** (5 + NaN).
- результат арифметической операции не определён для переданных операндов (undefined + undefined);
- арифметическая операция со строкой, **кроме сложения** ('привет' * 5).

В следующем примере приведены некоторые способы получения специальных значений.

**example_6**. Специальные значения числового типа

```
<script>
    // специальные значения
    console.log(-7 / 0); // -Infinity
    console.log(2e999); // Infinity
    console.log('ten' / 2); // NaN
    console.log(100 / '2'); // 50
    console.log(100 / 'two'); // NaN
    console.log(NaN + '5'); // 'NaN5'
    console.log(NaN * 5); // NaN
</script>
```

Согласно спецификации, **NaN** не равен самому себе. Проверить, что в переменной хранится **NaN** простым сравнением **не получится**:

```
const result = NaN;
console.log(result === NaN); // false
```

Для проверки на **NaN** пользуйтесь методом **isNaN**() (см. далее).

# Методы экземпляра объекта Number

### toString

```
num.toString([radix]);
```

Метод num.**toString**() возвращает строковое представление указанного объекта **Number**.

#### Параметр

- **radix** – необязательный параметр. Целое число, определяющее основание системы счисления, используемой для представления числового значения.

**example_7**. Метод toString()

```
<script>
    // определим числовой примитив
    const number = 42;
    // преобразуем примитив методом toString()
    console.log(number.toString()); // '42'
    console.log(number.toString(2)); // '101010'
    console.log(number.toString(8)); // '52'
    console.log(number.toString(16)); // '2a'
</script>
```

### toFixed

```
num.toFixed([digits]);
```

Метод num.**toFixed**() форматирует число, используя запись с фиксированной запятой.

**Параметр**

- **digits** – необязательный параметр. Количество цифр после десятичной запятой, если аргумент опущен, он считается равным 0.

**example_8**. Метод toFixed()

```
<script>
    // определим числовой примитив
    var num = 123.456;
    // преобразуем примитив методом toFixed()
    console.log(num.toFixed()); // '123'
    console.log(num.toFixed(1)); // '123.5'
    console.log(num.toFixed(5)); // '123.45600'
    console.log((1.23e+3).toFixed(2));  // '1230.00'
    console.log(2.34.toFixed(2)); // '2.34'
</script>
```

## toPrecision

Метод **toPrecision**() возвращает строку, представляющую объект **Number** с указанной точностью.

```
num.toPrecision([precision]);
```

**Параметр**

- **precision –** необязательный параметр. Целое число, определяющее количество значащих цифр.

**Возвращаемое значение**

Строка, представляющая объект **Number** в записи с фиксированной запятой или в экспоненциальной записи, округлённое до **precision** значащих цифр.

Если аргумент **precision** опущен, поведение аналогично методу **toString**().

**example_9**. Метод toPrecision()

```
<script>

    // определим числовой примитив

    var val = 5.123456;

    // преобразуем примитив методом toPrecision()

    console.log(val.toPrecision());  // '5.123456'

    console.log(val.toPrecision(5));  // '5.1235'

    console.log(val.toPrecision(2));  // '5.1'

    console.log(val.toPrecision(1));  // '5'

</script>
```

# Немного об объектах и автомобилях

Пришло время несколько расширить наше представление об объектах.

Создание любого технически сложного изделия начинается с проектной документации, раскрывающей сущность изделия, определяющей архитектурные, конструктивные и инженерно-технические решения проекта.

Например, создаваемый автомобиль сначала существует в голове конструктора и на многочисленных **чертежах**, **схемах**, **рисунках**. Так вот – это и есть **объект** (или еще говорят – **класс**)**.** У такого **объекта** есть техническое описание множества реализуемых им свойств, характеристик и возможностей: длина, высота, клиренс, колесная база, крутящий момент и прочее.

Компания **Mitsubishi** имеет техническую документацию на большой набор **объектов**: Pajero, Pajero Sport, Outlander ....

На рисунке 1 представлен фрагмент описания **объекта** – **Mitsubishi Pajero**.

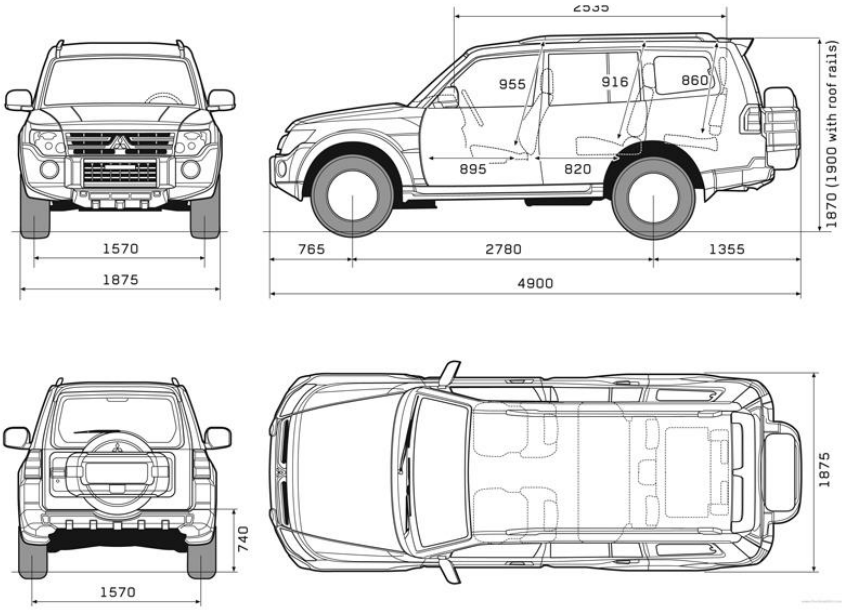


Рис.1 **Объект** Mitsubishi Pajero

Готовый автомобиль называется **экземпляром объекта**.

На рисунке 2 представлен **экземпляр объекта** Mitsubishi Pajero.


Рис.2. **Экземпляр объекта** Mitsubishi Pajero

На рисунке 3 еще один **экземпляр объекта** Mitsubishi Pajero.


Рис.3. Другой **экземпляр объекта** Mitsubishi Pajero

**Экземпляров** объекта **Mitsubishi Pajero** может быть сколько угодно (один из них стоит у меня в гараже). Экземпляр объекта **физически существует**, его можно потрогать, на нем можно ездить (в отличии от похожего и даже очень красивого, но нарисованного на ватмане).

Вернемся к объектам JavaScript.

```
// вызываем конструктор Number
let num1 = new Number(100);
let num2 = new Number(35);
let num3 = new Number(47);
// вызываем конструктор String
let str = new String("String Object");
/*
где:
-- Number,String – объекты JavaScript
-- num1, num2, num3, str – экземпляры объектов JavaScript
```

```
*/
```

Таким образом, следующие фразы:

- я поеду на **экземпляре** объекта Mitsubishi Pajero, или
- я вызвал метод **toString**()  на экземпляре объекта String (пример: num.**toString**()),

имеют вполне определенный смысл и равновероятны в моей жизни

| | **Объект JS** | **Объект Mitsubishi Pajero** |
|---|---|---|
| **Кто разрабатывает** | Корпорация Oracle | Конструкторское бюро |
| **Кто может внести изменения** | Программист уровня Middle обязан уметь | Условный "дядя Паша" из соседнего гаража |
| **Кто использует** | Любой программист (даже начинающий) | Все, у кого есть деньги на покупку экземпляра |

И помните, в программировании нет ничего такого, с чем невозможно разобраться. Вопрос ЖЕЛАНИЯ.

# Статические методы объекта Number

Далее, немного о **статических методах**. Что такое методы объектов и почему они статические, достаточно сложно объяснить без изучения соответствующей теории. Но основы знать необходимо уже сейчас. В следующем примере я попытался **простыми** словами объяснить смысл **непростых** вещей. Получилось?

**example_10**. Статические методы объектов

```
<script>

    // --- строка (объект String) ---
```

```
    // инициализация строки через создание объекта String
    // str - экземпляр объекта String
    let str = new String("Welcome pechora_PRO ...");
    // вызов метода includes на экземпляре объекта String
    console.log(str.includes("pechora")); // true
    // --- число (объект Number) ---
    // инициализация числа через создание объекта Number
    // num - экземпляр объекта Number
    let num = new Number(2.333);
    // вызов метода toString на экземпляре объекта Number
    console.log(num.toString()); // '2.333'
    // !!! к статическим методам объекта необходимо обращаться
    // без создания экземпляра этого объекта
    // например:
    // вызов метода isFinite() происходит напрямую на объекте
    Number
    // без создания экземпляра объекта
    /*
        let digit = new Number(25);
        digit.isFinite(); // ошибка вызова
        Number.isFinite(digit); // так правильно
    */
    // вызов метода isFinite на объекте Number
    console.log(Number.isFinite(NaN)); // false
</script>
```

Объект Number располагает **набором статических методов** для работы с числами. Рассмотрим некоторые, наиболее встречаемые из них.

## parseInt

```
Number.parseInt(string[, radix]);
```

Метод **parseInt**() принимает строку в качестве аргумента и возвращает **целое число** в соответствии с указанным **основанием системы счисления**..

### Параметры

- **string** – значение для разбора. Если параметр не является строкой, он будет в неё преобразован. Ведущие **пробельные** символы в строке **игнорируются**.
- **radix** – необязательный параметр. Целое число, представляющее **основание системы счисления** для числа в указанной выше строке.

### Возвращаемое значение

Целое число, полученное **парсингом** (разбором и интерпретацией) переданной строки. Если **первый символ строки** не может быть преобразован в число, то возвращается **NaN**.

```
console.log(Number.parseInt('12.53')); // 12
```

```
console.log(Number.parseInt('   101')); // 101
```

```
console.log(Number.parseInt('user')); // NaN
```

Если основание системы счисления имеет значение **undefined** (не определено) или равно **0** (или не указано), то **JavaScript по умолчанию** предполагает следующее:

- Если значение входного параметра string начинается с "0x" или "0X", **за основание системы счисления принимается 16**, и интерпретации подвергается оставшаяся часть строки.

```
console.log(Number.parseInt('0xF', 16)); // 15
```

```
console.log(Number.parseInt('0x12')); // 18
```

- Если значение входного параметра string начинается с "0", **за основание системы счисления принимается либо 8, либо 10**, в зависимости от браузера.

```
console.log(Number.parseInt('0011')); // 11
```

- Если значение входного параметра string начинается с любого другого символа, **система счисления считается десятичной** (основание 10).

Если строка начинается с **чисел**, а далее идут текстовые символы, то парсинг прервётся **на первом символе**, который не удастся конвертировать в число.

```
console.log(Number.parseInt('123hello')); // 123
console.log(Number.parseInt('50px')); // 50
console.log(Number.parseInt('0b011')); // 0
```

Достаточно много вариантов применения. Может быть, следующий демонстрационный пример поможет разобраться.

**example_11**. Метод parseInt()

```
<script>
    // только эксперименты расставят все на свои места
    console.log(Number.parseInt('0011')); // 11 (10-тичное)
    console.log(Number.parseInt('0011', 2)); // 3 (2-ичное)
    // 16-ричная система счисления распознается
    console.log(Number.parseInt('0x011')); // 17 (16-ричное)
    // 8-, 2-ичные системы счисления не распознаются
    console.log(Number.parseInt('0o0011')); // 0
    console.log(Number.parseInt('0b0011')); // 0
    // указание основания для системы счисления не поможет
    console.log(Number.parseInt('0o0011', 8)); // 0
    console.log(Number.parseInt('0b0011', 2)); // 0
```

```
    // вот так можно
    console.log(Number.parseInt('0011', 8)); // 8 (8-ричное)
    console.log(Number.parseInt('0011', 2)); // 3 (2-ичное)
</script>
```

### parseFloat

```
Number.parseFloat(string)
```

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

#### Параметр

- **string** – строка, представляющая значение, которое вы хотите разобрать.

#### Возвращаемое значение

Возвращает **число**, полученное из разобранной строки или **NaN**, если первый символ не удалось преобразовать в число. Пробелы игнорируются.

```
console.log(Number.parseFloat('12.12')); // 12.12
console.log(Number.parseFloat('12')); // 12
console.log(Number.parseFloat('  12.42')); // 12.42
console.log(Number.parseFloat('  1.15  ')); // 1.15
console.log(Number.parseFloat('pechora')); // NaN
```

Если строка начинается с чисел, а заканчивается текстовыми символами, то парсинг прервётся на первом символе, который не удастся конвертировать в число.

```
parseFloat('123.12hello'); // 123.12
```

Если встречается вторая точка, то остаток строки так же отбрасывается.

```
console.log(Number.parseFloat('12.34.567')); // 12.34
```

**example_12**. Применение методов parseInt(), parseFloat()

```
<script>
	// получим ссылку на элемент
	let div = document.querySelector('div');
	// прочитаем значения в переменные
	let width = div.style.width;
	let height = div.style.height;
	// всегда анализируем, что происходит
	console.log(Number.parseInt(width));
	console.log(Number.parseFloat(height));
	// установим новые значения
	div.style.width = Number.parseInt(width) + 500 + "px";
	div.style.height = Number.parseFloat(height) + 20 + "px";
	// сравниваем размеры блока до и после приращения
	console.log(div.style.width);
	console.log(div.style.height);
</script>
```

### isFinite

```
Number.isFinite(testValue);
```

Метод **isFinite**() определяет, является ли переданное значение конечным числом.

#### Параметр

- **testValue** – значение, проверяемое на конечность.

Если аргумент является **NaN**, положительной или отрицательной бесконечностью, а также не является числом функция вернёт **false**. Иначе возвращается **true**.

**example_12**. Метод isFinite()

```
<script>

    console.log(Number.isFinite(Infinity));  // false

    console.log(Number.isFinite(NaN));  // false

    console.log(Number.isFinite(-Infinity));  // false

    console.log(Number.isFinite(0));  // true

    console.log(Number.isFinite(2e6));  // true

    console.log(Number.isFinite('0'));  // false

    console.log(Number.isFinite('hello'));  // false
</script>
```

### isNaN

В JavaScript невозможно полагаться на "==" и "===" для определения, является ли переменная или литерал нечисловым значением NaN или нет, так как проверки NaN == NaN и NaN === NaN в качестве значения вернут **false**. Для таких целей используется метод isNaN().

```
Number.isNaN(value);
```

Метод **isNaN**() определяет является ли литерал или переменная нечисловым значением **NaN** или нет.

#### Параметр

- **value** – литерал или переменная которые будут проверяться на нечисловое значение – **NaN**.

#### Возвращаемое значение

Метод возвращает **true** только для числовых значений, имеющих значение **NaN**. Напомню, что значения **Infinity**, **-Infinity** и **NaN** являются специальными значениями числового типа.

**example_14**. Метод isNaN()

```
<script>
    // true
    console.log(Number.isNaN(NaN));
    console.log(Number.isNaN('then' / 2));
    console.log(Number.isNaN(100 / 'two'));
    console.log(Number.isNaN(NaN * 5));
    // можно проверить
    // console.log('then' / 2);  // NaN
    // console.log(NaN * 5);   // NaN
    // false
    console.log(Number.isNaN(0));
    console.log(Number.isNaN(37));
    console.log(Number.isNaN('37'));
    console.log(Number.isNaN('37.37'));
    // можно проверить
    // console.log(37);
    // console.log('37.37');
</script>
```

# Глобальные функции

Это еще не все. Язык JavaScript имеет набор **глобальных функций**. **Некоторые** из функций предназначены для работы с числами и полностью **совпадают** по названию со **статическими методами** объекта **Number**.

Например:

- **parseInt**()
- **parseFloat**()
- **isFinite**()
- **isNaN**()
- и пр. (будем проходить в других темах).

Поскольку эти методы являются глобальными, а глобальным объектом является окно браузера, эти методы на самом деле являются **оконными методами**:

- **parseInt**() - это то же самое, что window.**parseInt**().
- **isNaN**() - это то же самое, что window.**isNaN**().

> ***Важно***. Действия, выполняемые глобальными функциями, **полностью совпадают** с соответствующими методами или могут **отличаться**. Для каждой пары "функция - метод" **своя реализация**. Для ознакомления используйте справочники в открытых источниках.

**example_15**. Статические методы и глобальные функции

```
<script>
    // -- parseInt()
    // функция
    console.log(window.parseInt("-0XF", 16));  // -15
    // метод
    console.log(Number.parseInt("-0XF", 16));  // -15
```

```
    // -- parseFloat()
    // функция
    console.log(parseFloat('0.0314E+2'));  // 3.14
    // метод
    console.log(Number.parseFloat('3.14'));  // 3.14
</script>
```

# Преобразования чисел

В конце урока небольшой обобщающий материал, который может пригодиться при выполнении вычислений в ваших скриптах.

### Преобразование строки в число

Явно привести строку в число можно несколькими способами:

- Использовать **унарный оператор** "+", который необходимо расположить перед значением (в этом уроке еще не рассматривали).

```
console.log(+'7.35'); // 7.35
console.log(+'string'); // NaN
// оператор пренебрегает пробелами в начале и конце строки,
// а также переводом строки
console.log(+' 7.35 '); // 7.35
console.log(+'7.35 \n '); // 7.35
```

- 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.

```
console.log(parseInt('18px')); // 18

console.log(Number.parseInt('18.5px')); // 18

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

- Для преобразования строки в число с плавающей точкой в JavaScript имеются глобальная функция **parseFloat**() или метод Number.**parseFloat**().

```
console.log(parseFloat('33.3%')); // 33.3

console.log(Number.parseFloat('33.3px')); // 33.3

console.log(parseFloat('3.14')); // 3.14
```

### Преобразование числа в строку

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

```
console.log((12.8).toString()); // '12.8'
```

# **Задание** № 11

В файле **data.js** раздаточного материала вам предложен **JavaScript массив объектов**, представляющих тизеры фильмов.

Изучите структуру объектов. Напишите скрипт, запрашивающий у пользователя **номер** тизера для вывода (число в диапазоне от **0** до **3**). Запрос выполнить с помощью диалогового окна **prompt**().

Вывести выбранный тизер в **браузер**.

**Важно**. Ввод **номера** тизера выполнять в **двоичной системе счисления**.

Сопоставление чисел систем счисления:

- $0_{10}$ - $000_2$
- $1_{10}$ - $001_2$
- $2_{10}$ - $010_2$
- $3_{10}$ - $011_2$

**Примерный формат** вывода тизера представлен на рисунке ниже.

**Название: Зеленая миля**

**Год выпуска: 1999**

**Страна: США**

**Актеры:**

Том Хэнкс
Дэвид Морс
Майкл Кларк Дункан

# **Задание** № 12

Напишите скрипт, вычисляющий **результат** выражений, представленных в файле. Операнды выражений представлены в виде чисел, записанных в различных **системах счисления**.

Результат каждого выражения выведите в **консоль**.

**Выражение 1**

- **$(206_8$ / $1000011_2$ + $1000_2$ - -$13B_{16}$) / $31_8$**

**Выражение 2**

- **$(200_{16}$ – $48_{16}$) / $50_8$ % 111**

**Выражение 3**

- **--x % $100_2$ + $127_8$ - -(--y) / x++**

**где: x = $11000_2$;  y = $285_{16}$**

# **Задание** № 13

В файле **index.html** раздаточного материала представлен скрипт, в котором программист пытается изменить ширину блока тега **div**. Причем, ширина блока задается числом в **двоичной системе счисления**. При выполнении скрипта происходит **ошибка**.

**Проанализируйте** код скрипта, **найдите** и **исправьте** ошибку, допущенную программистом.

# Задание № 14

В файле **index.html** вам предложен документ с выводом геометрической фигуры **треугольник**. Текстовые поля в свойстве **value** содержат значения для **основания** и **высоты** треугольника.

```
33
34          <div class="fields">
35              <div class="input">
36                  <label class="base">Основание <i>a</i></label>
37                  <!-- значение для основания фигуры -->
38                  <input value="5" type="text" id="base">
39              </div>
40              <div class="input">
41                  <label for="height">Высота <i>h</i></label>
42                  <!-- значение для высоты фигуры -->
43                  <input value="16" type="text" id="height">
44              </div>
45              <div class="input">
46                  <label for="area"><i>S</i></label>
47                  <!-- элемент для записи результата -->
48                  <input value="" type="text" id="area">
49              </div>
50          </div>
51      </div>
52
```

Напишите скрипт расчета **площади** фигуры:

**S = (a * h) / 2**

Запишите **результат** в текстовое поле.

**Важно**. Результат расчета выводить в виде числа с **точностью** - **два символа** после плавающей точки. Например: 56.00

Поэкспериментируйте со **значениями основания** и **высоты** фигуры.