Number.prototype.toLocaleString()

The `toLocaleString()` method returns a string with a language-sensitive representation of this number.

JavaScript Demo: Number.toLocaleString()

```
1  function eArabic(x){
2    return x.toLocaleString('ar-EG');
3  }
4
5  console.log(eArabic(123456.789));
6  // expected output: "١٢٣٬٤٥٦٫٧٨٩"
7
8  console.log(eArabic("123456.789"));
9  // expected output: "123456.789"
10
11 console.log(eArabic(NaN));
12 // expected output: "ليس رقم"
13
```

Run ›

Reset

The new `locales` and `options` arguments customize the behavior of the function and let applications specify the language whose formatting conventions should be used. In older implementations, which ignore the `locales` and `options` arguments, the locale used and the form of the string returned are entirely implementation dependent.

Syntax

```
numObj.toLocaleString([locales [, options]])
```

Parameters

Check the Browser compatibility section to see which browsers support the `locales` and `options` arguments, and the Example: Checking for support for `locales` and `options` arguments for feature detection.

> **Note:** ECMAScript Internationalization API, implemented with Firefox 29, added the `locales` argument to the `Number.toLocaleString()` method. If the argument is `undefined`, this method returns localized digits specified by the OS, while the previous versions of Firefox returned Western Arabic digits. This change has been reported as a regression affecting backward compatibility, and has been fixed in Firefox 55. (bug 999003)

`locales`

Optional. A string with a BCP 47 language tag, or an array of such strings. For the general form and interpretation of the `locales` argument, see the Intl page. The following Unicode extension key is allowed:

`nu`

The numbering system to be used. Possible values include: `"arab"`, `"arabext"`, `"bali"`, `"beng"`, `"deva"`, `"fullwide"`, `"gujr"`, `"guru"`, `"hanidec"`, `"khmr"`, `"knda"`, `"laoo"`, `"latn"`, `"limb"`, `"mlym"`, `"mong"`, `"mymr"`, `"orya"`, `"tamldec"`, `"telu"`, `"thai"`, `"tibt"`.

`options`

Optional. An object with some or all of the following properties:

`localeMatcher`

The locale matching algorithm to use. Possible values are `"lookup"` and `"best fit"`; the default is `"best fit"`. For information about this option, see the Intl page.

`style`

The formatting style to use. Possible values are `"decimal"` for plain number formatting, `"currency"` for currency formatting, and `"percent"` for percent formatting; the default is `"decimal"`.

`currency`

The currency to use in currency formatting. Possible values are the ISO 4217 currency codes, such as `"USD"` for the US dollar, `"EUR"` for the euro, or `"CNY"` for the Chinese RMB — see the Current currency & funds code list. There is no default value; if the `style` is `"currency"`, the `currency` property must be provided.

`currencyDisplay`

How to display the currency in currency formatting. Possible values are `"symbol"` to use

a localized currency symbol such as €, `"code"` to use the ISO currency code, `"name"` to use a localized currency name such as `"dollar"`; the default is `"symbol"`.

`useGrouping`
Whether to use grouping separators, such as thousands separators or thousand/lakh/crore separators. Possible values are `true` and `false`; the default is `true`.

The following properties fall into two groups: `minimumIntegerDigits`, `minimumFractionDigits`, and `maximumFractionDigits` in one group, `minimumSignificantDigits` and `maximumSignificantDigits` in the other. If at least one property from the second group is defined, then the first group is ignored.

`minimumIntegerDigits`
The minimum number of integer digits to use. Possible values are from 1 to 21; the default is 1.

`minimumFractionDigits`
The minimum number of fraction digits to use. Possible values are from 0 to 20; the default for plain number and percent formatting is 0; the default for currency formatting is the number of minor unit digits provided by the ISO 4217 currency code list (2 if the list doesn't provide that information).

`maximumFractionDigits`
The maximum number of fraction digits to use. Possible values are from 0 to 20; the default for plain number formatting is the larger of `minimumFractionDigits` and 3; the default for currency formatting is the larger of `minimumFractionDigits` and the number of minor unit digits provided by the ISO 4217 currency code list (2 if the list doesn't provide that information); the default for percent formatting is the larger of `minimumFractionDigits` and 0.

`minimumSignificantDigits`
The minimum number of significant digits to use. Possible values are from 1 to 21; the default is 1.

`maximumSignificantDigits`
The maximum number of significant digits to use. Possible values are from 1 to 21; the default is 21.

Return value

A string with a language-sensitive representation of the given number.

Examples

Using `toLocaleString`

In basic use without specifying a locale, a formatted string in the default locale and with default options is returned.

```
var number = 3500;

console.log(number.toLocaleString()); // Displays "3,500" if in U.S. English locale
```

Checking for support for `locales` and `options` arguments

The `locales` and `options` arguments are not supported in all browsers yet. To check for support in ES5.1 and later implementations, the requirement that illegal language tags are rejected with a `RangeError` exception can be used:

```
function toLocaleStringSupportsLocales() {
  var number = 0;
  try {
    number.toLocaleString('i');
  } catch (e) {
    return e.name === 'RangeError';
  }
  return false;
}
```

Prior to ES5.1, implementations were not required to throw a range error exception if `toLocaleString` is called with arguments.

A check that works in all hosts, including those supporting ECMA-262 prior to ed 5.1, is to test for the features specified in ECMA-402 that are required to support regional options for `Number.prototype.toLocaleString` directly:

```
function toLocaleStringSupportsOptions() {
  return !!(typeof Intl == 'object' && Intl && typeof Intl.NumberFormat == 'function');
}
```

This tests for a global `Intl` object, checks that it's not `null` and that it has a `NumberFormat` property that is a function.

Using `locales`

This example shows some of the variations in localized number formats. In order to get the

format of the language used in the user interface of your application, make sure to specify that language (and possibly some fallback languages) using the `locales` argument:

```
var number = 123456.789;

// German uses comma as decimal separator and period for thousands
console.log(number.toLocaleString('de-DE'));
// → 123.456,789

// Arabic in most Arabic speaking countries uses Eastern Arabic digits
console.log(number.toLocaleString('ar-EG'));
// → ١٢٣٤٥٦٫٧٨٩

// India uses thousands/lakh/crore separators
console.log(number.toLocaleString('en-IN'));
// → 1,23,456.789

// the nu extension key requests a numbering system, e.g. Chinese decimal
console.log(number.toLocaleString('zh-Hans-CN-u-nu-hanidec'));
// → 一二三,四五六.七八九

// when requesting a language that may not be supported, such as
// Balinese, include a fallback language, in this case Indonesian
console.log(number.toLocaleString(['ban', 'id']));
// → 123.456,789
```

Using `options`

The results provided by `toLocaleString` can be customized using the `options` argument:

```
var number = 123456.789;

// request a currency format
console.log(number.toLocaleString('de-DE', { style: 'currency', currency: 'EUR' }));
// → 123.456,79 €

// the Japanese yen doesn't use a minor unit
console.log(number.toLocaleString('ja-JP', { style: 'currency', currency: 'JPY' }))
// → ￥123,457

// limit to three significant digits
console.log(number.toLocaleString('en-IN', { maximumSignificantDigits: 3 }));
// → 1,23,000

// Use the host default language with options for number formatting
var num = 30000.65;
```

```
console.log(num.toLocaleString(undefined, {minimumFractionDigits: 2, maximumFractionDigits: 2}));
// → "30,000.65" where English is the default language, or
// → "30.000,65" where German is the default language, or
// → "30 000,65" where French is the default language
```

Performance

When formatting large numbers of numbers, it is better to create a `NumberFormat` object and use the function provided by its `NumberFormat.format` property.

Specifications

Specification	Status	Comment
ECMAScript 3rd Edition (ECMA-262)	ST Standard	Initial definition. Implemented in JavaScript 1.5.
ECMAScript 5.1 (ECMA-262) The definition of 'Number.prototype.toLocaleString' in that specification.	ST Standard	
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Number.prototype.toLocaleString' in that specification.	ST Standard	
ECMAScript Latest Draft (ECMA-262) The definition of 'Number.prototype.toLocaleString' in that specification.	D Draft	
ECMAScript Internationalization API 1.0 (ECMA-402) The definition of 'Number.prototype.toLocaleString' in that specification.	ST Standard	
ECMAScript Internationalization API 2.0 (ECMA-402) The definition of 'Number.prototype.toLocaleString' in that specification.	ST Standard	
ECMAScript Internationalization API 4.0 (ECMA-402) The definition of 'Number.prototype.toLocaleString' in that specification.	D Draft	

Browser compatibility

Update compatibility data on GitHub

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet	Node.js
`toLocaleString`	Yes	Yes	1	Yes	Yes	Yes	Yes	Yes	4	Yes	Yes	Yes	Yes
`locales`	24	Yes	29	11	15	10	4.4	26	56	14	10	Yes	Yes
`options`	24	Yes	29	11	15	10	4.4	26	56	14	10	Yes	Yes

. Full support

See also

- `Number.prototype.toString()`

Tags: Internationalization JavaScript Method Number Prototype

Contributors to this page: mdnwebdocs-bot, Thanaen, chharvey, Nekiono, jonathan-s, wbamberg, fscholz, jameshkramer, myf, thetalecrafter, eduardoboucas, Mems, Robg1, Shelah, Mingun, kohei.yoshino, acdha, Norbert, Sheppy, Nfroidure, ethertank, aseville, evilpie, Sevenspade, Brettz9

Last updated by: mdnwebdocs-bot, Mar 18, 2019, 4:52:50 PM

MDN web docs
moz://a

moz://a

Other languages:
English (US) (en-US)

Terms Privacy Cookies

MDN

Web Technologies
Learn Web Development
About MDN
Feedback

Mozilla

About
Contact Us
Firefox