How can I format numbers as dollars currency string in JavaScript?

Ask Question

1486

475

I would like to format a price in JavaScript.
I'd like a function which takes a `float` as an argument and returns a `string` formatted like this:

```
"$ 2,500.00"
```

What's the best way to do this?

javascript formatting currency

edited Jul 6 '17 at 9:41

community wiki
11 revs, 7 users 31%
Daniel Magliola

8 There is no built-in function `formatNumber` in javascript – zerkms Feb 16 '12 at 20:39

413 Please, to anyone reading this in the future, do **not** use float to store currency. You will loose precision and data. You should store it as a integer number of cents (or pennies etc.) and then convert prior to output. – Philip Whitehouse Mar 4 '12 at 13:35

5 @user1308743 Float doesn't store decimal places. It stores numbers using a value, base and offset. 0.01 is not actually representable. See: en.wikipedia.org/wiki/Floating_point#Accuracy_problems – Philip Whitehouse Jun 10 '12 at 11:11

5 @user1308743: Imagine you represent a very big number (lets say you are a lucky guy and it is your bank account balance). Would you really want to loose money because of a precision deficiency ? – ereOn Aug 6 '12 at 9:14

127 So why hasn't anyone suggested the following? (2500).toLocaleString("en-GB", {style: "currency", currency: "GBP", minimumFractionDigits: 2}) developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/… – Nick Grealy Sep 25 '13 at 1:41

61 Answers

1 2 3 next

Number.prototype.toFixed

1569

This solution is compatible with every single major browser:

```
const profits = 2489.8237;

profits.toFixed(3) //returns 2489.82
profits.toFixed(2) //returns 2489.82
profits.toFixed(7) //returns 2489.82
```

All you need is to add the currency symbol (e.g. `"$" + profits.toFixed(2)`) and you will have your amount in dollars.

Custom function

If you require the use of `,` between each digit, you can use this function:

```
function formatMoney(n, c, d, t) {
  var c = isNaN(c = Math.abs(c)) ? 2
    d = d == undefined ? "." : d,
    t = t == undefined ? "," : t,
    s = n < 0 ? "-" : "",
    i = String(parseInt(n = Math.abs
    j = (j = i.length) > 3 ? j % 3 :

  return s + (j ? i.substr(0, j) + t
t) + (c ? d + Math.abs(n - i).toFixe
};

document.getElementById("b").addEven
  document.getElementById("x").inner
formatMoney(document.getElementById(
});
```

```
<label>Insert your amount: <input id
<br />
<button id="b">Get Output</button>
<p id="x">(press button to get outpu
```

Run code snippet Expand snippet

Use it like so:

```
(123456789.12345).formatMoney(2, ".",
```

If you're always going to use '.' and ',', you can leave them off your method call, and the method will default them for you.

```
(123456789.12345).formatMoney(2);
```

If your culture has the two symbols flipped (i.e. Europeans) and you would like to use the defaults, just paste over the following two lines in the `formatMoney` method:

```
    d = d == undefined ? "," : d,
    t = t == undefined ? "." : t,
```

Custom function (ES6)

If you can use modern ECMAScript syntax (i.e. through Babel), you can use this simpler function instead:

```
function formatMoney(amount, decimal
  try {
    decimalCount = Math.abs(decimalC
    decimalCount = isNaN(decimalCoun

    const negativeSign = amount < 0

    let i = parseInt(amount = Math.a
0).toFixed(decimalCount)).toString()
    let j = (i.length > 3) ? i.lengt

    return negativeSign + (j ? i.sub
i.substr(j).replace(/(\d{3})(?=\d)/g
Math.abs(amount - i).toFixed(decimal
  } catch (e) {
    console.log(e)
  }
};
document.getElementById("b").addEven
  document.getElementById("x").inner
formatMoney(document.getElementById(
});
```

```
<label>Insert your amount: <input id
<br />
<button id="b">Get Output</button>
<p id="x">(press button to get outpu
```

Run code snippet Expand snippet

edited Sep 12 '18 at 12:23

community wiki
10 revs, 9 users 46%
haykam

26 first of all, excellent, concise code. however, if you are american, you should change the defaults of `d` and `t` to be `.` and `,` respectively so that you don't have to specify them every time. also, i recommend modifying the beginning of the `return` statement to read: `return s + '$' + [rest]`, otherwise you will not get a dollar sign. – Jason Jan 31 '11 at 23:58

677 Not sure why people think this code is beautiful. It is indecipherable. It seems to work nicely, but it is not beautiful. – usr Oct 24 '12 at 16:28

82 Is this formatMoney function copied from some minified JavaScript code somewhere? Can you not post the original? What do the variables c, d, i, j, n, s, and t stand for? Judging by the amount of upvotes and comments this post has I can assume this code has been copy pasted into production websites everywhere... Good luck maintaining the code if it has a bug some day! – zuallauz Dec 17 '12 at 20:41

246 "poetry"? More like obscurity. This isn't code golf; use a little white space. Proper var names wouldn't hurt, either. – keithjgrant Dec 30 '12 at 14:07

31 *Any fool can write code that a computer can understand. Good programmers write code that humans can understand* – Liam Jun 7 '16 at 12:32

1109

Short and fast solution (works everywhere!)

```
(12345.67).toFixed(2).replace(/\d(?=(\
```

The idea behind this solution is replacing matched sections with first match and comma, i.e. `'$&,'`. The matching is done using lookahead approach. You may read the expression as *"match a number if it is followed by a sequence of three number sets (one or more) and a dot"*.

TESTS:

```
1        --> "1.00"
12       --> "12.00"
123      --> "123.00"
1234     --> "1,234.00"
```

```
12345     --> "12,345.00"
123456    --> "123,456.00"
1234567   --> "1,234,567.00"
12345.67  --> "12,345.67"
```

DEMO:
http://jsfiddle.net/hAfMM/9571/

Extended short solution

You can also extend the prototype of `Number` object to add additional support of any number of decimals `[0 .. n]` and the size of number groups `[0 .. x]`:

```
/**
 * Number.prototype.format(n, x)
 *
 * @param integer n: length of decimal
 * @param integer x: length of section
 */
Number.prototype.format = function(n,
    var re = '\\d(?=(\\d{' + (x || 3)
    return this.toFixed(Math.max(0, ~~
};

1234..format();           // "1,234"
12345..format(2);         // "12,345.0
123456.7.format(3, 2);    // "12,34,56
123456.789.format(2, 4);  // "12,3456.
```

DEMO / TESTS:
http://jsfiddle.net/hAfMM/435/

Super extended short solution

In this super extended version you may set different delimiter types:

```
/**
 * Number.prototype.format(n, x, s, c)
 *
 * @param integer n: length of decimal
 * @param integer x: length of whole p
 * @param mixed   s: sections delimite
 * @param mixed   c: decimal delimiter
 */
Number.prototype.format = function(n,
    var re = '\\d(?=(\\d{' + (x || 3)
        num = this.toFixed(Math.max(0,

    return (c ? num.replace('.', c) :
',');
};

12345678.9.format(2, 3, '.', ',');  //
123456.789.format(4, 4, ' ', ':');  //
12345678.9.format(0, 3, '-');       //
```

DEMO / TESTS:
http://jsfiddle.net/hAfMM/612/

edited Jun 28 '18 at 14:18

community wiki
12 revs, 5 users 74%
VisioN

13 I actually went a step further: `.replace(/(\d)(?=(\d{3})+(?:\.\d+)?$)/g, "$1,")` . – kalisjoshua Mar 21 '13 at 2:50

3 CoffeeScript version with of VisioN & kalisjoshua regexp and way of specifying decimal place (so you can leave the default of 2 or specify 0 for no decimal): `Number.prototype.toMoney = (decimal=2) -> @toFixed(decimal).replace /(\d)(?=(\d{3})+(?:\.\d+)?$)/g, "$1,"` – Eric Anderson Jun 18 '13 at 15:43

9 @Abbas Yeah, replace `\.` with `$` (end of line), i.e. `this.toFixed(0).replace(/(\d)(?=(\d{3})+$)/g, "$1,")` . – VisioN Aug 15 '13 at 9:26

2 @hanumant The regular grammar is a bit complicated here, so I suggest you to read the manuals about regular expressions first (e.g. at MDN). The idea behind it is replacing matched sections with first match and comma, i.e. `$1,` . The matching is done using lookahead approach. You may read the expression as *"match a number if it is followed by a sequence of three number sets (one or more) and a dot"*. – VisioN Oct 22 '13 at 15:08

2 @JuliendePrabère Please give an example of a long number which doesn't work with this approach. – VisioN Mar 25 '14 at 10:53

Intl.numberformat

891

Javascript has a number formatter (part of the Internationalization API).

```
// Create our number formatter.
var formatter = new Intl.NumberFormat(
  style: 'currency',
  currency: 'USD',
  minimumFractionDigits: 2,
  // the default value for minimumFrac
  // and is usually already 2
});
```

```
formatter.format(2500); /* $2,500.00 *
```

JS fiddle

Use `undefined` in place of the first argument (`'en-US'` in the example) to use the system locale (the user locale in case the code is running in a browser).

Intl.NumberFormat vs Number.prototype.toLocaleString

A final note comparing this to the older `.toLocaleString`. They both offer essentially the same functionality. However, toLocaleString in its older incarnations (pre-Intl) does not actually support locales: it uses the system locale. Therefore, to be sure that you're using the correct version, MDN suggests to check for the existence of `Intl`. So if you need to check for Intl anyway, why not use *it* instead? However, if you choose to use the shim, that also patches `toLocaleString`, so in that case you can use it without any hassle:

```
(2500).toLocaleString('en-US', {
  style: 'currency',
  currency: 'USD',
}); /* $2,500.00 */
```

Some notes on browser support

- Browser support is no longer an issue nowadays with 97% support in the US/EU
- For other parts of the world (90% supported), the biggest offenders in terms of support are UC Mobile (stay away from that) and Opera Mini (crippled by design)
- There is a shim to support it on older browsers
- Have a look at CanIUse for more info

edited Nov 8 '18 at 10:19

community wiki

43 revs, 3 users 98%
aross

47 This idomatic JavaScript, simple and elegant solution is exactly what I was looking for. – Guilhem Soulas Feb 11 '16 at 12:44

8 unreliable on safari – chulian May 2 '16 at 0:13

6 I love this, but check support before you use it: caniuse.com/#feat=internationalization – jocull Aug 4 '16 at 17:29

6 Voting this one because it's a stupidly simple answer that works natively. – Trasiva Aug 30 '16 at 21:56

13 Pretty sure a quite high % of browsers now support this. This should be upvoted much more. – flq Dec 23 '16 at 21:34

178

Take a look at the JavaScript Number object and see if it can help you.

- `toLocaleString()` will format a number using location specific thousands separator.
- `toFixed()` will round the number to a specific number of decimal places.

To use these at the same time the value must have its type changed back to a number because they both output a string.

Example:

```
Number(someNumber.toFixed(1)).toLocale
```

edited Nov 22 '16 at 12:48

community wiki
4 revs, 4 users 65%
17 of 26

2 Thanks! Based on this idea I was able to make one that is short and simple enough! (and localized) Excellent. – Daniel Magliola Sep 29 '08 at 15:25

5 Actually You can. i.e. for dollars: '$'+(value +

0.001).toLocaleString().slice(0,-1) – Zaptree Nov 18 '13 at 3:33

6 Looks like it'd be great, but there is little browser support at the moment – acorncom Dec 6 '13 at 0:38

1 @acorncom Why do you say there is "little browser support"? The Number object has been around since Javascript 1.1. Please provide a reference that backs up your claim. – Doug S Aug 31 '15 at 3:43

1 Care should be taken that there is an old version of `toLocaleString` that uses the system locale, and a new (incompatible) one that comes from ECMAScript Intl API. Explained here. This answer seems to be intended for the old version. – aross Sep 14 '17 at 10:09

157

Below is the Patrick Desjardins (alias Daok) code with a bit of comments added and some minor changes:

```
/*
decimal_sep: character used as deciaml
thousands_sep: char used as thousands
*/
Number.prototype.toMoney = function(de
{
   var n = this,
   c = isNaN(decimals) ? 2 : Math.abs(
means user does not want to show any d
   d = decimal_sep || '.', //if no dec
decimal separator (we MUST use a decim

   /*
   according to [https://stackoverflow
argument-is-not-sent-to-the-javascript
   the fastest way to check for not de
'undefined'
   rather than doing value === undefin
   */
   t = (typeof thousands_sep === 'unde
to use a thousands separator you can p

   sign = (n < 0) ? '-' : '',

   //extracting the absolute value of
string
   i = parseInt(n = Math.abs(n).toFixe

   j = ((j = i.length) > 3) ? j % 3 :
   return sign + (j ? i.substr(0, j) +
"$1" + t) + (c ? d + Math.abs(n - i).t
}
```

and here some tests:

```
//some tests (do not forget parenthesi
decimals)
alert(123456789.67392.toMoney() + '\n'
123456789.67392.toMoney(0) + '\n' + (1
'\n' + 89.67392.toMoney() + '\n' + (89
//some tests (do not forget parenthesi
decimals)
alert((-123456789.67392).toMoney() + '
```

The minor changes are:

1. moved a bit the `Math.abs(decimals)` to be done only when is not `NaN` .
2. `decimal_sep` can not be empty string anymore (a some sort of decimal separator is a MUST)
3. we use `typeof thousands_sep === 'undefined'` as suggested in [How best to determine if an argument is not sent to the JavaScript function]()
4. `(+n || 0)` is not needed because `this` is a `Number` object

edited May 23 '17 at 12:03

community wiki
5 revs
Marco Demaio

7 You may want to use '10' as the radix in parseInt. Otherwise, any number that starts with '0' will use octal numbering. – sohtimsso1970 Nov 15 '11 at 16:01

3 @sohtimsso1970: sorry for the late response, but could you explain some more? I don't see where a number could be interpreted as octal. The `parseInt` is called on the absolute value of the INTEGER part of the number. The INTEGER part can not start with ZERO unless it's just a ZERO! And `parseInt(0) === 0` either octal or decimal. – Marco Demaio Feb 9 '12 at 12:20

4 @Tracker1: I understood that a number starting with `0` is considered octal by `parseInt` . But in this code is IMPOSSIBLE for `parseInt` to receive `016` as input (or any other octal formatted value), because the argument passed to `parseInt` is 1st processed by `Math.abs` function. So there is no way for `parseInt` to receive a number that starts with zero unless it's just a zero or `0.nn` (where `nn` are decimals). But both `0` and `0.nn` strings would be converted by `parseInt` into a plain ZERO as suppsed to be. – Marco Demaio Mar 20 '12 at 14:57

120

accounting.js is a tiny JavaScript library for number, money and currency formatting.

edited Apr 11 '18 at 15:03

community wiki
3 revs, 3 users 50%
GasheK

2 Looks like the IE7/IE8 bug is fixed. – Mat Schaffer Jan 17 '12 at 19:41

2 This is a great library, being able to pass the currency symbol is also a good idea, since all the currency details are contained in the single function call/settings – farinspace Oct 19 '12 at 22:15

2 I like the fact that you can do the reverse--pass a formatted currency string and get the numeric value. – Neil Monroe Jun 26 '14 at 16:25

2 accounting.js doesn't seem maintained lately. One fork with recent changes is github.com/nashdot/accounting-js – RationalDev Apr 20 '16 at 22:49

99

If amount is a number, say `-123` , then

```
amount.toLocaleString('en-US', { style
```

will produce the string `"-$123.00"` .

Here's a complete working example.

edited Apr 11 '18 at 15:05

community wiki
3 revs, 3 users 37%
cs01

7 This answer was almost there for me, but I needed it to be rounded to the nearest penny. This is what I used amount.toLocaleString('en-GB', { style: 'currency', currency: 'GBP',

maximumFractionDigits: 2 }); – Nico Nov 18 '14 at 11:47

3 Doesn't seem to work in Safari. It just

returns the number as a String without any formatting. – Lance Anderson May 8 '15 at 3:22

1 Wow, this is a really great answer. Should be top. – Ethan Jul 14 '17 at 23:56

96

Here's the best js money formatter I've seen:

```
Number.prototype.formatMoney = functio
    var n = this,
        decPlaces = isNaN(decPlaces = 
        decSeparator = decSeparator ==
        thouSeparator = thouSeparator 
        sign = n < 0 ? "-" : "",
        i = parseInt(n = Math.abs(+n |
        j = (j = i.length) > 3 ? j % 3
    return sign + (j ? i.substr(0, j) 
i.substr(j).replace(/(\d{3})(?=\d)/g,
Math.abs(n - i).toFixed(decPlaces).sli
};
```

It was re-formatted and borrowed from here: https://stackoverflow.com/a/149099/751484

You'll have to supply your own currency designator (you used $ above).

Call it like this (although note that the args default to 2, comma, & period, so you don't need to supply any args if that's your preference):

```
var myMoney=3543.75873;
var formattedMoney = '$' + myMoney.for
```

edited May 23 '17 at 11:47

community wiki
6 revs, 3 users 80%
Jonathan M

6 @hacklikecrack, all variables are local; they're in the `var` statement. – Jonathan M Nov 20 '13 at 17:58

3 sorry, yes, though you're redeclaring arguments. Indentation! ;) – hacklikecrack Feb 25 '14 at 16:18

71

There are already some great answers here. Here's another attempt, just for fun:

```
function formatDollar(num) {
    var p = num.toFixed(2).split(".");
    return "$" + p[0].split("").revers
        return  num=="-" ? acc : num +
    }, "") + "." + p[1];
}
```

And some tests:

```
formatDollar(45664544.23423) // "$45,6
formatDollar(45) // "$45.00"
formatDollar(123) // "$123.00"
formatDollar(7824) // "$7,824.00"
formatDollar(1) // "$1.00"
```

Edited: now it will handle negative numbers as well

edited Dec 7 '16 at 6:35

community wiki
2 revs, 2 users 91%
Wayne Burkett

1 @Steve - You're right, but you'd need to do something like `i = orig.length - i - 1` in the callback. Still, one less traversal of the array. – Wayne Burkett Dec 20 '11 at 22:29

11 A not about compatability: The `reduce` method was introduced in Ecmascript 1.8, and is not supported in Internet Explorer 8 and below. – Blaise May 10 '12 at 12:07

68

I think what you want is `f.nettotal.value = "$" + showValue.toFixed(2);`

answered Feb 16 '12 at 20:42

community wiki
crush

11 Once you append a $ sign to it, it is no longer a number, but a string. – crush Feb 16 '12 at 20:59

59

So why hasn't anyone suggested the following?

```
(2500).toLocaleString("en-GB", {style:
minimumFractionDigits: 2})
```

Works for most/some browsers:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/toLocaleString#Browser_Compatibility

answered Sep 25 '13 at 1:42

community wiki
Nick Grealy

1 Because 'locales' and 'options' arguments are supported just by a very small number of browsers, like Chrome 24, IE11 and Opera 15. Firefox, Safari and older versions of others still don't support it. – VisioN Sep 25 '13 at 6:50

3 Agreed, it's not fully supported across all browsers (yet), but it's still a solution. (And arguably the most valid solution, as its forward compatible with the non-supported browsers, and it's a documented feature of the Javascript api.) – Nick Grealy Sep 25 '13 at 22:59

1 I like this and am happy that it works with Indian digit grouping. – MSC Jul 3 '16 at 6:35

4 This is fully supported as of 2017 and should be the only correct answer – Evgeny Apr 14 '17 at 15:33

24

✓

Ok, based on what you said, i'm using this:

```
var DecimalSeparator = Number("1.2").t

var AmountWithCommas = Amount.toLocale
var arParts = String(AmountWithCommas)
var intPart = arParts[0];
var decPart = (arParts.length > 1 ? ar
decPart = (decPart + '00').substr(0,2)

return '£ ' + intPart + DecimalSeparat
```

I'm open to improvement suggestions (i'd prefer not to include YUI just to do this :-)) I already know I should be detecting the "." instead of just using it as the decimal separator...

answered Sep 29 '08 at 15:22

community wiki
Daniel Magliola

7 Note that your version doesn't properly round to two decimal digits. For example, 3.706 would be formatted as "£ 3.70", not as "£ 3.71" as it's supposed to be. – Ates Goral Sep 30 '08 at 23:33

24

Numeral.js - a js library for easy number formatting by @adamwdraper

```
numeral(23456.789).format('$0,0.00');
```

edited Feb 27 '16 at 14:45

community wiki
2 revs, 2 users 60%
Yarin

24

I use the library Globalize (from Microsoft):

It's a great project to localize numbers, currencies and dates and to have them automatically formatted the right way according to the user locale! ...and despite it should be a jQuery extension, it's currently a 100% independent library. I suggest you all to try it out! :)

edited Apr 11 '18 at 15:06

community wiki
2 revs, 2 users 50%
daveoncode

3 Wow, why is this not upvoted more? Big standardized library for all sorts of formatting. Industry-standard formatting parameters with correct globalization. Great answer!! – pbarranis Sep 10 '13 at 2:11

1 No longer in alpha (or beta). This seems to be very useful while we wait for Safari to meet the new standard and for IE < 11 to die. – Guy Schalnat Aug 21 '15 at 18:26

22

javascript-number-formatter (formerly at Google Code)

- Short, fast, flexible yet standalone. ~~Only 75 lines including MIT license info, blank lines & comments.~~
- Accept standard number formatting like `#,##0.00` or with negation `-000.####`.
- Accept any country format like `# ##0,00`, `#,###.##`, `#'###.##` or any type of non-numbering symbol.
- Accept any numbers of digit grouping. `#,##,#0.000` or `#,###0.##` are all valid.
- Accept any redundant/fool-proof formatting. `##,###,##.#` or `0#,#00#.###0#` are all OK.
- Auto number rounding.
- Simple interface, just supply mask & value like this: `format( "0.0000", 3.141592)`.
- Include a prefix & suffix with the mask

(excerpt from its README)

edited Aug 3 '16 at 17:47

community wiki
2 revs, 2 users 62%
Goodeq

There is a javascript port of the PHP function "number_format".

20

I find it very usefull as it is easy to use and recognisable for PHP developers.

```
function number_format (number, decima
    var n = number, prec = decimals;

    var toFixedFix = function (n,prec)
        var k = Math.pow(10,prec);
        return (Math.round(n*k)/k).toS
    };

    n = !isFinite(+n) ? 0 : +n;
    prec = !isFinite(+prec) ? 0 : Math
    var sep = (typeof thousands_sep ==
    var dec = (typeof dec_point === 'u

    var s = (prec > 0) ? toFixedFix(n,
    //fix for IE parseFloat(0.55).toFi

    var abs = toFixedFix(Math.abs(n),
    var _, i;

    if (abs >= 1000) {
        _ = abs.split(/\D/);
        i = _[0].length % 3 || 3;

        _[0] = s.slice(0,i + (n < 0))
               _[0].slice(i).replace(/
        s = _.join(dec);
    } else {
        s = s.replace('.', dec);
    }

    var decPos = s.indexOf(dec);
    if (prec >= 1 && decPos !== -1 &&
        s += new Array(prec-(s.length-
    }
    else if (prec >= 1 && decPos === -
        s += dec+new Array(prec).join(
    }
    return s;
}
```

(Comment block from the original, included below for examples & credit where due)

```
// Formats a number with grouped thous
//
// version: 906.1806
// discuss at: http://phpjs.org/functi
// +   original by: Jonas Raoni Soares
// +   improved by: Kevin van Zonnevel
// +     bugfix by: Michael White (htt
// +     bugfix by: Benjamin Lupton
// +     bugfix by: Allan Jensen (http
// +    revised by: Jonas Raoni Soares
// +     bugfix by: Howard Yeend
// +    revised by: Luke Smith (http:/
// +     bugfix by: Diogo Resende
// +     bugfix by: Rival
// +     input by: Kheang Hok Chin (ht
// +     improved by: davook
// +     improved by: Brett Zamir (htt
// +     input by: Jay Klehr
// +     improved by: Brett Zamir (htt
// +     input by: Amir Habibi (http:/
// +     bugfix by: Brett Zamir (http:
// *     example 1: number_format(1234
// *     returns 1: '1,235'
// *     example 2: number_format(1234
// *     returns 2: '1 234,56'
// *     example 3: number_format(1234
// *     returns 3: '1234.57'
// *     example 4: number_format(67, 
// *     returns 4: '67,00'
// *     example 5: number_format(1000
// *     returns 5: '1,000'
// *     example 6: number_format(67.3
// *     returns 6: '67.31'
// *     example 7: number_format(1000
// *     returns 7: '1,000.6'
// *     example 8: number_format(6700
// *     returns 8: '67.000,00000'
// *     example 9: number_format(0.9,
// *     returns 9: '1'
// *     example 10: number_format('1.
// *     returns 10: '1.20'
// *     example 11: number_format('1.
// *     returns 11: '1.2000'
// *     example 12: number_format('1.
// *     returns 12: '1.200'
```

edited Sep 18 '14 at 20:25

community wiki
2 revs, 2 users 69%
DaMayan

20

+1 to Jonathan M for providing the original method. Since this is explicitly a currency formatter, I went ahead and added the currency symbol (defaults to '$') to the output, and added a default comma as the thousands separator. If you don't actually want a currency symbol (or thousands separator), just use "" (empty string) as your argument for it.

```
Number.prototype.formatMoney = functio
currencySymbol) {
    // check the args and supply defau
    decPlaces = isNaN(decPlaces = Math
    decSeparator = decSeparator == und
    thouSeparator = thouSeparator == u
    currencySymbol = currencySymbol ==

    var n = this,
        sign = n < 0 ? "-" : "",
        i = parseInt(n = Math.abs(+n |
        j = (j = i.length) > 3 ? j % 3

    return sign + currencySymbol + (j
i.substr(j).replace(/(\d{3})(?=\d)/g,
Math.abs(n - i).toFixed(decPlaces).sli
};
```

edited Oct 21 '14 at 4:01

community wiki
XML

2 You're right. That's an error I brought in from Jonathan M's original, where they're all chained as a single var expression. Those should be simple assignments. Fixing. – XML Oct 18 '13 at 19:27

2 `this` is a perfectly useful variable name. Converting it to `n` so you can save 3 characters at definition time may have been necessary in an era when RAM and bandwidth were counted in KB, but is merely obfuscatory in an era when the minifier will take care of all that before it ever hits production. The other clever micro-optimizations are at least debatable. – XML Oct 18 '13 at 19:49

19

A shorter method (for inserting space, comma or point) with regular expression ?

```
Number.prototype.toCurrencyString=
    return this.toFixed(2).replace
}

n=12345678.9;
alert(n.toCurrencyString());
```

edited Jan 4 '12 at 12:46

community wiki
2 revs
Julien de Prabère

15

Patrick Desjardins' answer looks good, but I prefer my javascript simple. Here's a function I just wrote to take a number in and return it in currency format (minus the dollar sign)

```
// Format numbers to two decimals with
function formatDollar(num) {
    var p = num.toFixed(2).split(".");
    var chars = p[0].split("").reverse
    var newstr = '';
    var count = 0;
    for (x in chars) {
        count++;
        if(count%3 == 1 && count != 1)
            newstr = chars[x] + ',' +
        } else {
            newstr = chars[x] + newstr
        }
    }
    return newstr + "." + p[1];
}
```

edited May 23 '17 at 12:10

community wiki
2 revs
Tim Saylor

15

There is a built-in `function` toFixed in `javascript`

```
var num = new Number(349);
document.write("$" + num.toFixed(2));
```

edited Aug 22 '12 at 19:10

community wiki
Gate

3 `toFixed()` is a function of the `Number` object and won't work on `var num` if it was a `String`, so the additional context helped me. – timborden Nov 20 '12 at 14:03

14

I suggest the NumberFormat class from Google Visualization API.

You can do something like this:

```
var formatter = new google.visualizati
    prefix: '$',
    pattern: '#,###,###.##'
});

formatter.formatValue(1000000); // $ 1
```

I hope it helps.

edited Oct 2 '12 at 21:25

community wiki
2 revs
juanchopx2

14

Haven't seen this one. It's pretty concise and easy to understand.

```
function moneyFormat(price, sign = '$
  const pieces = parseFloat(price).to
  let ii = pieces.length - 3
  while ((ii-=3) > 0) {
    pieces.splice(ii, 0, ',')
  }
  return sign + pieces.join('')
}

console.log(
  moneyFormat(100),
  moneyFormat(1000),
  moneyFormat(10000.00),
  moneyFormat(1000000000000000000)
)
```

Run code snippet Expand snippet

Here is a version with more options in the final output to allow formatting different currencies in different locality formats.

```
// higher order function that takes c
price
const makeMoneyFormatter = ({
  sign = '$',
  delimiter = ',',
  decimal = '.',
  append = false,
  precision = 2,
  round = true,
  custom
} = {}) => value => {

  const e = [1, 10, 100, 1000, 10000,

  value = round
    ? (Math.round(value * e[precision
    : parseFloat(value)

  const pieces = value
    .toFixed(precision)
    .replace('.', decimal)
    .split('')

  let ii = pieces.length - (precision

  while ((ii-=3) > 0) {
    pieces.splice(ii, 0, delimiter)
  }

  if (typeof custom === 'function') {
    return custom({
      sign,
      float: value,
      value: pieces.join('')
    })
  }
```

```
    return append
      ? pieces.join('') + sign
      : sign + pieces.join('')
  }

  // create currency converters with th
  const formatDollar = makeMoneyFormatt
  const formatPound = makeMoneyFormatte
    sign: '£',
    precision: 0
  })
  const formatEuro = makeMoneyFormatter
    sign: '€',
    delimiter: '.',
    decimal: ',',
    append: true
  })

  const customFormat = makeMoneyFormatt
    round: false,
    custom: ({ value, float, sign }) =>
  })

  console.log(
    formatPound(1000),
    formatDollar(10000.0066),
    formatEuro(100000.001),
    customFormat(999999.555)
  )
```

Run code snippet Expand snippet

edited Jun 8 '17 at 5:41

community wiki
6 revs
synthet1c

13

```
function CurrencyFormatted(amount)
{
    var i = parseFloat(amount);
    if(isNaN(i)) { i = 0.00; }
    var minus = '';
    if(i < 0) { minus = '-'; }
    i = Math.abs(i);
    i = parseInt((i + .005) * 100);
    i = i / 100;
    s = new String(i);
    if(s.indexOf('.') < 0) { s += '.00
    if(s.indexOf('.') == (s.length - 2
    s = minus + s;
    return s;
}
```

From WillMaster.

answered Sep 29 '08 at 15:16

community wiki
Bill the Lizard

13

This might be a little late, but here's a method I just worked up for a coworker to add a locale-aware `.toCurrencyString()` function to all numbers. The internalization is for number grouping only, NOT the currency sign - if you're outputting dollars, use `"$"` as supplied, because `$123 4567` in Japan or China is the same number of USD as `$1,234,567` is here in the US. If you're outputting euro/etc., then change the currency sign from `"$"` .

Declare this anywhere in your HEAD or wherever necessary, just before you need to use it:

```
  Number.prototype.toCurrencyString =
    if (typeof prefix === 'undefined')
    if (typeof suffix === 'undefined')
    var _localeBug = new RegExp((1).to
'\\.') + "$");
    return prefix + (~~this).toLocaleS
1).toFixed(2).toLocaleString().replace
  }
```

Then you're done! Use `(number).toCurrencyString()` anywhere you need to output the number as currency.

```
var MyNumber = 123456789.125;
alert(MyNumber.toCurrencyString()); //
MyNumber = -123.567;
alert(MyNumber.toCurrencyString()); //
```

edited Apr 11 '18 at 15:07

community wiki
4 revs, 2 users 97%
Jay Dansand

12

The main part is inserting the thousand-separators, that could be done like this:

```
<script type="text/javascript">
function ins1000Sep(val){
  val = val.split(".");
  val[0] = val[0].split("").reverse().
  val[0] = val[0].replace(/(\d{3})/g,"
  val[0] = val[0].split("").reverse().
  val[0] = val[0].indexOf(",")==0?val[
  return val.join(".");
}
function rem1000Sep(val){
  return val.replace(/,/g,"");
}
function formatNum(val){
  val = Math.round(val*100)/100;
  val = (""+val).indexOf(".")>-1 ? val
  var dec = val.indexOf(".");
  return dec == val.length-3 || dec ==
}
</script>

<button onclick="alert(ins1000Sep(form
```

edited Sep 29 '08 at 15:18

community wiki
2 revs
roenving

10

As usually, there are multiple ways of doing the same thing but I would avoid using `Number.prototype.toLocaleString` since it can return different values based on the user settings.

I also don't recommend extending the `Number.prototype` - extending native objects prototypes is a bad practice since it can cause conflicts with other people code (e.g. libraries/frameworks/plugins) and may not be compatible with future JavaScript implementations/versions.

I believe that Regular Expressions are the best approach for the problem, here is my implementation:

```
/**
 * Converts number into currency forma
 * @param {number} number   Number tha
 * @param {string} [decimalSeparator]
 * @param {string} [thousandsSeparator
 * @param {int} [nDecimalDigits]    Nu
 * @return {string} Formatted string (
'12,345.67')
 */
function numberToCurrency(number, deci
    //default values
    decimalSeparator = decimalSeparato
    thousandsSeparator = thousandsSepa
    nDecimalDigits = nDecimalDigits ==

    var fixed = number.toFixed(nDecima
        parts = new RegExp('^(-?\\d{1,
$').exec( fixed ); //separate begin [$

    if(parts){ //number >= 1000 || num
        return parts[1] + parts[2].rep
(parts[4] ? decimalSeparator + parts[4
    }else{
        return fixed.replace('.', deci
    }
}
```

edited on 2010/08/30: added option to set number of decimal digits. edited on 2011/08/23: added option to set number of decimal digits to zero.

edited Aug 23 '11 at 15:33

community wiki
5 revs
Miller Medeiros

10

Here are some solutions, all pass the test suite, test suite and benchmark included, if you want copy and paste to test, try This Gist.

Method 0 (RegExp)

Base on https://stackoverflow.com/a/14428340/1877620, but fix if there is no decimal point.

```
if (typeof Number.prototype.format ===
    Number.prototype.format = function
        if (!isFinite(this)) {
            return this.toString();
        }

        var a = this.toFixed(precision
        a[0] = a[0].replace(/\d(?=(\d{
        return a.join('.');
    }
}
```

Method 1

```
if (typeof Number.prototype.format ===
    Number.prototype.format = function
        if (!isFinite(this)) {
            return this.toString();
        }

        var a = this.toFixed(precision
            // skip the '-' sign
            head = Number(this < 0);

        // skip the digits that's befo
        head += (a[0].length - head) %
        a[0] = a[0].slice(0, head) + a
        return a.join('.');
    };
}
```

Method 2 (Split to Array)

```
if (typeof Number.prototype.format ===
    Number.prototype.format = function
        if (!isFinite(this)) {
            return this.toString();
        }

        var a = this.toFixed(precision

        a[0] = a[0]
            .split('').reverse().join(
            .replace(/\d{3}(?=\d)/g, '
            .split('').reverse().join(

        return a.join('.');
    };
}
```

Method 3 (Loop)

```
if (typeof Number.prototype.format ===
    Number.prototype.format = function
        if (!isFinite(this)) {
            return this.toString();
        }

        var a = this.toFixed(precision
        a.push('.');

        var i = a.indexOf('.') - 3;
        while (i > 0 && a[i-1] !== '-'
            a.splice(i, 0, ',');
            i -= 3;
        }

        a.pop();
        return a.join('');
    };
}
```

Usage Example

```
console.log('======== Demo ========')
console.log(
    (1234567).format(0),
    (1234.56).format(2),
    (-1234.56).format(0)
);
var n = 0;
for (var i=1; i<20; i++) {
    n = (n * 10) + (i % 10)/100;
    console.log(n.format(2), (-n).form
}
```

Separator

If we want custom thousands separator or decimal separator, use `replace()`:

```
123456.78.format(2).replace(',', ' ').
```

Test suite

```
function assertEqual(a, b) {
    if (a !== b) {
        throw a + ' !== ' + b;
    }
}

function test(format_function) {
    console.log(format_function);
    assertEqual('NaN', format_function
    assertEqual('Infinity', format_fun
    assertEqual('-Infinity', format_fu

    assertEqual('0', format_function.c
    assertEqual('0.00', format_functio
    assertEqual('1', format_function.c
    assertEqual('-1', format_function.
    // decimal padding
    assertEqual('1.00', format_functio
    assertEqual('-1.00', format_functi
    // decimal rounding
    assertEqual('0.12', format_functio
    assertEqual('0.1235', format_funct
    assertEqual('-0.12', format_functi
    assertEqual('-0.1235', format_func
    // thousands separator
    assertEqual('1,234', format_functi
    assertEqual('12,345', format_funct
    assertEqual('123,456', format_func
    assertEqual('1,234,567', format_fu
    assertEqual('12,345,678', format_f
    assertEqual('123,456,789', format_
    assertEqual('-1,234', format_funct
    assertEqual('-12,345', format_func
    assertEqual('-123,456', format_fun
    assertEqual('-1,234,567', format_f
    assertEqual('-12,345,678', format_
    assertEqual('-123,456,789', format
    // thousands separator and decimal
    assertEqual('1,234.12', format_fun
    assertEqual('12,345.12', format_fu
    assertEqual('123,456.12', format_f
    assertEqual('1,234,567.12', format
    assertEqual('12,345,678.12', forma
    assertEqual('123,456,789.12', form
    assertEqual('-1,234.12', format_fu
    assertEqual('-12,345.12', format_f
    assertEqual('-123,456.12', format_
    assertEqual('-1,234,567.12', forma
    assertEqual('-12,345,678.12', form
    assertEqual('-123,456,789.12', for
}

console.log('======== Testing ========
test(Number.prototype.format);
test(Number.prototype.format1);
test(Number.prototype.format2);
test(Number.prototype.format3);
```

Benchmark

```
function benchmark(f) {
    var start = new Date().getTime();
    f();
    return new Date().getTime() - star
}
```

```
function benchmark_format(f) {
    console.log(f);
    time = benchmark(function () {
        for (var i = 0; i < 100000; i+
            f.call(123456789, 0);
            f.call(123456789, 2);
        }
    });
    console.log(time.format(0) + 'ms')
}

// if not using async, browser will st
// this will create a new thread to be
async = [];
function next() {
    setTimeout(function () {
        f = async.shift();
        f && f();
        next();
    }, 10);
}

console.log('======== Benchmark ======
async.push(function () { benchmark_for
next();
```

edited May 23 '17 at 12:34

community wiki
6 revs
Steely Wing

10

I found this from: accounting.js . Its very easy and perfectly fits my need.

```
// Default usage:
accounting.formatMoney(12345678); //

// European formatting (custom symbol
second parameter:
accounting.formatMoney(4999.99, "€",

// Negative values can be formatted r
accounting.formatMoney(-500000, "£ ",

// Simple `format` string allows cont
accounting.formatMoney(5318008, { syr

// Euro currency symbol to the right
accounting.formatMoney(5318008, {symb
format: "%v%s"}); // 1.008,00€
```

Run code snippet Expand snippet

edited Jun 7 '17 at 8:01

community wiki

2 revs, 2 users 94%
Faysal Haque

9

A simple option for proper comma placement by reversing the string first and basic regexp.

```
String.prototype.reverse = function()
    return this.split('').reverse().jo
};

Number.prototype.toCurrency = function
     // format decimal or round to nea
     var n = this.toFixed( round_decim

     // convert to a string, add comma
     // by reversing string
     return (n + '').reverse().replace
};
```

edited Dec 2 '11 at 23:58

community wiki
2 revs, 2 users 77%
troy

8

Patrick Desjardins (ex Daok)'s example worked well for me. I ported over to coffeescript if anyone is interested.

```
Number.prototype.toMoney = (decimals =
",") ->
    n = this
    c = if isNaN(decimals) then 2 else
    sign = if n < 0 then "-" else ""
    i = parseInt(n = Math.abs(n).toFix
    j = if (j = i.length) > 3 then j %
    x = if j then i.substr(0, j) + tho
    y = i.substr(j).replace(/(\d{3})(?
    z = if c then decimal_separator +
    sign + x + y + z
```

edited Feb 9 '12 at 12:04

community wiki
3 revs, 3 users 69%
jc00ke

1 2 3 next

protected by VisioN Feb 11 '13 at 9:46

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?