# accounting.js

**accounting.js** is a tiny JavaScript library by Open Exchange Rates, providing simple and advanced number, money and currency formatting.

Features custom output formats, parsing/unformatting of numbers, easy localisation and spreadsheet-style column formatting (to line up symbols and decimals).

It's lightweight, has no dependencies and is suitable for all client-side and server-side JavaScript applications.

Tweet


- methods & examples
- demo
- instructions
- documentation
- roadmap
- feedback / support
- download
- links


## Library Methods

### **formatMoney()** - format any number into currency

The most basic library function for formatting numbers as money values, with customisable currency symbol, precision (decimal places), and thousand/decimal separators:

```
// Default usage:
accounting.formatMoney(12345678); // $12,345,678.00

// European formatting (custom symbol and separators), can also use options object as second parameter:
accounting.formatMoney(4999.99, "€", 2, ".", ","); // €4.999,99

// Negative values can be formatted nicely:
accounting.formatMoney(-500000, "£ ", 0); // £ -500,000

// Simple `format` string allows control of symbol position (%v = value, %s = symbol):
accounting.formatMoney(5318008, { symbol: "GBP",  format: "%v %s" }); // 5,318,008.00 GBP
```

### **formatColumn()** - format a list of values for column-display

This table demonstrates how **accounting.js** can take a list of numbers and money-format them with padding to line up currency symbols and decimal places

In order for the padded spaces to render correctly, the containing element must be CSS styled with `white-space: pre` (pre-formatted) - otherwise the browser will squash them into single spaces.

| Original Number: | With accounting.js: | Different settings: | European format: | Symbol after value: |
|---|---|---|---|---|
| 123.5 | $ 123.50 | HK$ 124 | € 123,50 | 123.50 GBP |
| 3456.615 | $ 3,456.62 | HK$ 3,457 | € 3.456,62 | 3,456.62 GBP |
| 777888.99 | $ 777,888.99 | HK$ 777,889 | € 777.888,99 | 777,888.99 GBP |
| -5432 | $ -5,432.00 | HK$ (5,432) | € -5.432,00 | -5,432.00 GBP |
| -1234567 | $ -1,234,567.00 | HK$ (1,234,567) | € -1.234.567,00 | -1,234,567.00 GBP |
| 0 | $ 0.00 | HK$ -- | € 0,00 | 0.00 GBP |

```
// Format list of numbers for display:
accounting.formatColumn([123.5, 3456.49, 777888.99, 12345678, -5432], "$ ");
```

### **formatNumber()** - format a number with custom precision and localisation

The base function of the library, which takes any number or array of numbers, runs `accounting.unformat()` to remove any formatting, and returns the number(s) formatted with separated thousands and custom precision:

```
accounting.formatNumber(5318008); // 5,318,008
accounting.formatNumber(9876543.21, 3, " "); // 9 876 543.210
```

### **toFixed()** - better rounding for floating point numbers

Implementation of toFixed() that treats floats more like decimal values than binary, fixing inconsistent precision rounding in JavaScript (where some .05 values round up, while others round down):

```
(0.615).toFixed(2); // "0.61"
accounting.toFixed(0.615, 2); // "0.62"
```

### **unformat()** - parse a value from any formatted number/currency string

Takes any number and removes all currency formatting. Aliased as `accounting.parse()`

```
accounting.unformat("£ 12,345,678.90 GBP"); // 12345678.9
```

# Demo / Try it out

Money formatting:

Enter any number into the box and choose currency. Uses `accounting.formatMoney()` :

$

Result: **$ 0.00**

Column formatting:

Edit the values in the table to see how **formatColumn()** keeps them aligned:

| | | |
|---|---|---|
| 1000000 | $ 1,000,000.00 | GBP 1,000,000 |
| -5000 | $ -5,000.00 | GBP (5,000) |
| 0 | $ 0.00 | GBP -- |

# Basic Instructions:

1. Download the script and put it somewhere, then reference it in your HTML like so:

```
<script src="path/to/accounting.js"></script>

<script type="text/javascript">
 // Library ready to use:
 accounting.formatMoney(5318008);
</script>
```

2. See the documentation and source-code for full method/parameter information.

# Documentation

Information on the parameters of each method. See library methods above for more examples. Optional parameters are in *[italics]*, with the default value indicated.

### accounting.settings

```
// Settings object that controls default parameters for library methods:
accounting.settings = {
 currency: {
  symbol : "$",   // default currency symbol is '$'
  format: "%s%v", // controls output: %s = symbol, %v = value/number (can be object: see below)
  decimal : ".",  // decimal point separator
  thousand: ",",  // thousands separator
  precision : 2   // decimal places
 },
 number: {
  precision : 0,  // default precision on numbers is 0
  thousand: ",",
  decimal : "."
 }
}

// These can be changed externally to edit the library's defaults:
accounting.settings.currency.format = "%s %v";

// Format can be an object, with `pos`, `neg` and `zero`:
accounting.settings.currency.format = {
 pos : "%s %v",   // for positive values, eg. "$ 1.00" (required)
 neg : "%s (%v)", // for negative values, eg. "$ (1.00)" [optional]
 zero: "%s  -- "  // for zero values, eg. "$  --" [optional]
};

// Example using underscore.js - extend default settings (also works with $.extend in jQuery):
accounting.settings.number = _.defaults({
 precision: 2,
```

```
 thousand: " "
}, accounting.settings.number);
```

### accounting.formatMoney()

```
// Standard usage and parameters (returns string):
accounting.formatMoney(number,[symbol = "$"],[precision = 2],[thousand = ","],[decimal = "."],[format = "%s%v"])

// Second parameter can be an object:
accounting.formatMoney(number, [options])

// Available fields in options object, matching `settings.currency`:
var options = {
 symbol : "$",
 decimal : ".",
 thousand: ",",
 precision : 2,
 format: "%s%v"
};

// Example usage:
accounting.formatMoney(12345678); // $12,345,678.00
accounting.formatMoney(4999.99, "€", 2, ".", ","); // €4.999,99
accounting.formatMoney(-500000, "£ ", 0); // £ -500,000

// Example usage with options object:
accounting.formatMoney(5318008, {
 symbol: "GBP",
 precision: 0,
 thousand: "·",
 format: {
  pos : "%s %v",
  neg : "%s (%v)",
  zero: "%s  --"
 }
});

// Will recursively format an array of values:
accounting.formatMoney([123, 456, [78, 9]], "$", 0); // ["$123", "$456", ["$78", "$9"]]
```

### accounting.formatColumn()

```
// Standard usage and parameters (returns array):
accounting.formatColumn(list, [symbol = "$"],[precision = 2],[thousand = ","],[decimal = "."],[format = "%s%v"])

// Second parameter can be an object (see formatNumber for available options):
accounting.formatColumn(list, [options])

// Example usage (NB. use a space after the symbol to add arbitrary padding to all values):
var list = [123, 12345];
accounting.formatColumn(list, "$ ", 0); // ["$    123", "$ 12,345"]

// List of numbers can be a multi-dimensional array (formatColumn is applied recursively):
var list = [[1, 100], [900, 9]];
accounting.formatColumn(list); // [["$  1.00", "$100.00"], ["$900.00", "$  9.00"]]
```

### accounting.formatNumber()

```
// Standard usage and parameters (returns string):
accounting.formatNumber(number, [precision = 0], [thousand = ","], [decimal = "."])

// Second parameter can also be an object matching `settings.number`:
accounting.formatNumber(number, [object])

// Example usage:
accounting.formatNumber(9876543); // 9,876,543
accounting.formatNumber(4999.99, 2, ".", ","); // 4.999,99

// Example usage with options object:
```

```
accounting.formatNumber(5318008, {
 precision : 3,
 thousand : " "
});

// Will recursively format an array of values:
accounting.formatNumber([123456, [7890, 123]]); // ["123,456", ["7,890", "123"]]
```

### accounting.toFixed()

```
// Standard usage and parameters (returns string):
accounting.toFixed(number, [precision = 0]);

// Example usage:
accounting.toFixed(0.615, 2); // "0.62"

// Compare to regular JavaScript `Number.toFixed()` method:
(0.615).toFixed(2); // "0.61"
```

### accounting.unformat()

```
// Standard usage and parameters (returns number):
accounting.unformat(string, [decimal]);

// Example usage:
accounting.unformat("GBP £ 12,345,678.90"); // 12345678.9

// If a non-standard decimal separator was used (eg. a comma) unformat() will need it in order to work out
// which part of the number is a decimal/float:
accounting.unformat("€ 1.000.000,00", ","); // 1000000
```

# Roadmap

Next Version:

- ~~Add more fine-grained control of formatting, with negatives and zero-values~~
- ~~Implement `map()` and type-checking helper methods to clean up API methods~~
- Find performance bottlenecks and work on speed optimisations
- Write more tests, docs and examples, add FAQ
- Implement feedback

Later:

- Add padding parameter to override amount of space between currency symbol and value.
- Add digit-grouping control, to allow eg. "$10,0000"
- Add choice of rounding method for precision (up, down or nearest-neighbour).
- Add several other general and excel-style money formatting methods.
- Create NPM package, if there's demand for it.
- Create wrapper for jQuery as a separate plugin (not in core) to allow eg.
  `$('td.accounting').formatMoney()`

See the Github Issues page for currently active issues.

# Feedback / Support

Please create issues on the accounting.js Github repository if you have feedback or need support, or contact Open Exchange Rates here.

# Download

- **accounting.js** - Latest version from Github (12kb)
- **accounting.min.js** - Latest version from Github (3kb, minified)
- Or check out the accounting.js Github repository for the full package.

# Links

accounting.js is maintained by **Open Exchange Rates** - the lightweight currency data API for startups, SMEs and Fortune 500s.

Feedback, support or questions? **Contact Open Exchange Rates** for guidance.

Bugs, issues, suggestions or contributions? Please **post them here**.

accounting.js works great with **money.js** - the tiny (1kb) standalone JavaScript currency conversion library, for web & nodeJS

---

Tweet