

# sprintf.js

**sprintf.js** is a complete open source JavaScript sprintf implementation for the *browser* and *node.js*.

Its prototype is simple:

```
string sprintf(string format , [mixed  
arg1 [, mixed arg2 [ ,...]]])
```

The placeholders in the format string are marked by % and are followed by one or more of these elements, in this order:

- An optional number followed by a \$ sign that selects which argument index to use for the value. If not specified, arguments will be placed in the same order as the placeholders in the input string.
- An optional + sign that forces to preceed the result with a plus or minus sign on numeric values. By default, only the - sign is used on negative numbers.
- An optional padding specifier that says what character to use for padding (if specified). Possible values are 0 or any other character preceded by a ' (single quote). The default is to pad with *spaces*.
- An optional - sign, that causes sprintf to left-align the result of this placeholder. The default is to right-align the result.
- An optional number, that says how many characters the result should have. If the value to be returned is shorter than this number, the result will be padded. When used with the j (JSON)

# License

`sprintjs` is licensed under the terms of the 3-clause BSD license.

array of arguments, rather than a variable number of arguments:  
`vsprintf` is the same as `sprintf` except that it accepts an

## JavaScript vsprintf

- type specifier, the padding length specifies the tab size used for indentation.
- An optional precision modifier, consisting of a `.` (dot) followed by a number, that says how many digits should be displayed for floating point numbers. When used with the `g` type specifier, it specifies the number of significant digits. When used on a string, it causes the result to be truncated.
- A type specifier that can be any of:
  - `%` — yields a literal `%` character
  - `b` — yields an integer as a binary number
  - `c` — yields an integer as the character with the ASCII value
  - `e` — yields a float using scientific notation
  - `u` — yields an integer as an unsigned decimal number
  - `f` — yields a float as `is`; see notes on precision above
  - `g` — yields a float as `is`; see notes on precision above
  - `o` — yields an integer as an octal number
  - `s` — yields a string as is
  - `X` — yields an integer as a hexadecimal number (upper-case)
  - `x` — yields an integer as a hexadecimal number (lower-case)
  - `j` — yields a JavaScript object or array as a JSON encoded string

- `sprintf` is the same as `sprintf` except that it accepts an array of arguments, rather than a variable number of arguments:  
`vsprintf` is the same as `sprintf` except that it accepts an

# Installation

## Via Bower

```
bower install sprintf
```

## Or as a node.js module

```
npm install sprintf-js
```

## Usage

```
var sprintf = require("sprintf-
js").sprintf,
    vsprintf = require("sprintf-
js").vsprintf

sprintf("%2$s %3$s a %1$s", "cracker",
"Polly", "wants")
vsprintf("The first 4 letters of the
english alphabet are: %s, %s, %s and
%s", ["a", "b", "c", "d"])
```

```
vsprintf("The first 4 letters of the
english alphabet are: %s, %s, %s and
%s", ["a", "b", "c", "d"])
```

## Argument swapping

You can also swap the arguments. That is, the order of the placeholders doesn't have to match the order of the arguments. You can do that by simply indicating in the format string which arguments the placeholders refer to:

```
sprintf("%2$s %3$s a %1$s", "cracker",
"Polly", "wants")
```

And, of course, you can repeat the placeholders without having to increase the number of arguments.

## Named arguments

Format strings may contain replacement fields rather than positional placeholders. Instead of referring to a certain argument, you can now refer to a certain key within an object. Replacement fields are surrounded by rounded parentheses - ( and ) - and begin with a keyword that refers to a key:

```
var user = {
  name: "Dolly"
}
```

Date().toString() })  
function() { return new  
sprintf("Current date and time: %s",  
1398005382890  
Date.now() // Current timestamp:  
sprintf("Current timestamp: %d",

invoked (with no arguments) in order to compute the value on-the-  
You can pass in a function as a dynamic value and it will be

supported  
Note: mixing positional and named placeholders is not (yet)

and Polly  
users: users} // Hello Polly, Molly  
(users[1].name)s and %(users[2].name)s",  
sprintf("Hello %(users[0].name)s, %  
]

{name: "Polly"}  
{name: "Molly"}  
{name: "Dolly"},  
var users = [

any number of keywords or indexes:  
Keywords in replacement fields can be optionally followed by

Dolly  
sprintf("Hello %(name)s", user) // Hello

Dates  
function() { return new  
sprintf("Current date and time: %s",  
1398005382890  
Date.now() // Current timestamp:  
sprintf("Current timestamp: %d",

invoked (with no arguments) in order to compute the value on-the-  
You can pass in a function as a dynamic value and it will be

## Computed values

supported  
Note: mixing positional and named placeholders is not (yet)

and Polly  
users: users} // Hello Polly, Molly  
(users[1].name)s and %(users[2].name)s",  
sprintf("Hello %(users[0].name)s, %  
]

{name: "Polly"}  
{name: "Molly"}  
{name: "Dolly"},  
var users = [

any number of keywords or indexes:  
Keywords in replacement fields can be optionally followed by

Dolly  
sprintf("Hello %(name)s", user) // Hello

# AngularJS

You can now use `sprintf` and `vprintf` (also aliased as `fmt` and `vfmt` respectively) in your AngularJS projects. See [demo](#).