
Sprig: Template functions for Go templates



The Go language comes with a built-in template language, but not very many template functions. Sprig is a library that provides more than 100 commonly used template functions.

It is inspired by the template functions found in Twig and in various JavaScript libraries, such as `underscore.js`.

IMPORTANT NOTES

Sprig leverages `mergo` to handle merges. In its v0.3.9 release, there was a behavior change that impacts merging template functions in sprig. It is currently recommended to use v0.3.10 or later of that package. Using v0.3.9 will cause sprig tests to fail.

Package Versions

There are two active major versions of the `sprig` package.

- v3 is currently stable release series on the `master` branch. The Go API should remain compatible with v2, the current stable version. Behavior change behind some functions is the reason for the new major version.
- v2 is the previous stable release series. It has been more than three years since the initial release of v2. You can read the documentation and see the code on the `release-2` branch. Bug fixes to this major version will continue for some time.

Usage

Template developers: Please use Sprig's function documentation for detailed instructions and code snippets for the >100 template functions available.

Go developers: If you'd like to include Sprig as a library in your program, our API documentation is available at [GoDoc.org](https://godoc.org).

For standard usage, read on.

Load the Sprig library

To load the Sprig `FuncMap`:

```
1
2 import (
3     "github.com/Masterminds/sprig/v3"
4     "html/template"
5 )
6
7 // This example illustrates that the FuncMap *must* be set before the
8 // templates themselves are loaded.
9 tpl := template.Must(
10     template.New("base").Funcs(sprig.FuncMap()).ParseGlob("*.html")
11 )
```

Calling the functions inside of templates

By convention, all functions are lowercase. This seems to follow the Go idiom for template functions (as opposed to template methods, which are TitleCase). For example, this:

```
1 {{ "hello!" | upper | repeat 5 }}
```

produces this:

```
1 HELLO!HELLO!HELLO!HELLO!HELLO!
```

Principles Driving Our Function Selection

We followed these principles to decide which functions to add and how to implement them:

- Use template functions to build layout. The following types of operations are within the domain of template functions:
 - Formatting
 - Layout
 - Simple type conversions
 - Utilities that assist in handling common formatting and layout needs (e.g. arithmetic)
- Template functions should not return errors unless there is no way to print a sensible value. For example, converting a string to an integer should not produce an error if conversion fails. Instead, it should display a default value.
- Simple math is necessary for grid layouts, pagers, and so on. Complex math (anything other than arithmetic) should be done outside of templates.
- Template functions only deal with the data passed into them. They never retrieve data from a source.

-
- Finally, do not override core Go template functions.