fmt.Printf formatting tutorial and cheat sheet

yourbasic.org/golang

Related

Go string handling overview [cheat sheet]

40+ essential string functions: literals, concatenation, equality, ordering, indexing, UTF-8, search, join, replace, split, trim, strip, lowercase/uppercase.

yourbasic.org

Runes and character encoding

A rune is a type meant to represent a Unicode code point. Strings, however, are sequences of bytes (typically containing Unicode text encoded in UTF-8).

yourbasic.org

Efficient string concatenation [full guide]

Use a strings.Builder together with the fmt package for a clean and simple way to build strings efficiently without redundant copying.

yourbasic.org

Most Read

Do you make these Go coding mistakes?

Why Go? – Key advantages you may have overlooked

Go string handling overview [cheat sheet]

Type, value and equality of interfaces

Concurrent programming

See all 178 Go articles

Basics

With the Go `fmt` package you can format numbers and strings padded with spaces or zeroes, in different bases, and with optional quotes.

You submit a **template string** that contains the text you want to format plus some **annotation verbs** that tell the `fmt` functions how to format the trailing arguments.

Printf

In this example, `fmt.Printf` formats and writes to standard output:

```
fmt.Printf("Binary: %b\\%b", 4, 5) // Prints `Binary: 100\101`
```

- the **template string** is `"Binary: %b\\%b"`,
- the **annotation verb** %b formats a number in binary, and
- the **special value** `\\` is a backslash.

As a special case, the verb %%, which consumes no argument, produces a percent sign:

```
fmt.Printf("%d %%", 50) // Prints `50 %`
```

Sprintf (format without printing)

Use `fmt.Sprintf` to format a string without printing it:

```
s := fmt.Sprintf("Binary: %b\\%b", 4, 5) // s == `Binary: 100\101`
```

Find fmt errors with vet

If you try to compile and run this incorrect line of code

```
fmt.Printf("Binary: %b\\%b", 4) // An argument to Printf is missing.
```

you'll find that the program will compile, and then print

```
Binary: 100\%!b(MISSING)
```

To catch this type of errors early, you can use the vet command – it can find calls whose arguments do not align with the format string.

```
$ go vet example.go
example.go:8: missing argument for Printf("%b"): format reads arg 2, have onl
```

Cheat sheet

Default formats and type

- **Value:** `[]int64{0, 1}`

Format	Verb	Description
[0 1]	%v	Default format
[]int64{0, 1}	%#v	Go-syntax format
[]int64	%T	The type of the value

Integer (indent, base, sign)

- **Value:** 15

Format	Verb	Description
15	%d	Base 10
+15	%+d	Always show sign

Format	Verb	Description
␣␣15	`%4d`	Pad with spaces (width 4, right justified)
15␣␣	`%-4d`	Pad with spaces (width 4, left justified)
0015	`%04d`	Pad with zeroes (width 4)
1111	`%b`	Base 2
17	`%o`	Base 8
f	`%x`	Base 16, lowercase
F	`%X`	Base 16, uppercase
0xf	`%#x`	Base 16, with leading 0x

Character (quoted, Unicode)

- **Value:** `65` (Unicode letter A)

Format	Verb	Description
A	`%c`	Character
'A'	`%q`	Quoted character
U+0041	`%U`	Unicode
U+0041 'A'	`%#U`	Unicode with character

Boolean (true/false)

Use `%t` to format a boolean as `true` or `false`.

Pointer (hex)

Use %p to format a pointer in base 16 notation with leading `0x`.

Float (indent, precision, scientific notation)

- **Value:** `123.456`

Format	Verb	Description
1.234560e+02	%e	Scientific notation
123.456000	`%f`	Decimal point, no exponent
123.46	`%.2f`	Default width, precision 2
␣123.46	`%8.2f`	Width 8, precision 2
123.456	%g	Exponent as needed, necessary digits only

String or byte slice (quote, indent, hex)

- **Value:** `"café"`

Format	Verb	Description
café	`%s`	Plain string
␣␣café	`%6s`	Width 6, right justify
café␣␣	`%-6s`	Width 6, left justify
"café"	`%q`	Quoted string
636166c3a9	`%x`	Hex dump of byte values
63 61 66 c3 a9	`% x`	Hex dump with spaces

Special values

Value	Description
`\a`	U+0007 alert or bell
`\b`	U+0008 backspace
`\\`	U+005c backslash
`\t`	U+0009 horizontal tab
`\n`	U+000A line feed or newline
`\f`	U+000C form feed
`\r`	U+000D carriage return

Value	Description
`\v`	U+000b vertical tab

Arbitrary values can be encoded with backslash escapes and can be used in any `" "` string literal.

There are four different formats:

- `\x` followed by exactly two hexadecimal digits,
- `\` followed by exactly three octal digits,
- `\u` followed by exactly four hexadecimal digits,
- `\U` followed by exactly eight hexadecimal digits.

The escapes `\u` and `\U` represent Unicode code points.

```
fmt.Println("\\caf\u00e9") // Prints \café
```

Further reading

40+ practical string tips [cheat sheet]