

A green cartoon gopher with large eyes and glasses is the central figure, surrounded by a chaotic pile of various snacks and candies. The snacks include bags of Cheez-It, Ruffles, Doritos, and Goldfish Cracker; boxes of Kit-Kat, Pocky, and Mochi; bags of Airheads and Popcorn; and various candies like Skittles, Gummies Go!, and Oh-Reo. The gopher is holding a bag of Nachos. The word "GoDDD" is written in large white letters across the gopher's chest.

GoDDD

Antti Kupila
@akupila

GO

GoFmt

Tests

GoDoc



```
package name
```

```
import (  
    "unicode"  
)
```

```
func FormatName(s string) string {  
    first := rune(s[0])  
    remain := s[1:]  
    return string(unicode.ToUpper(first)) + remain  
}
```



```
package name
```

```
import "testing"
```

```
func TestFormatName(t *testing.T) {  
    tests := []struct {  
        input string  
        want  string  
    }{  
        {"antti", "Antti"},  
    }  
    for _, tt := range tests {  
        t.Run(tt.input, func(t *testing.T) {  
            got := FormatName(tt.input)  
            if got != tt.want {  
                t.Errorf("FormatName() got = %q, want = %q", got, tt.want)  
            }  
        })  
    }  
}
```

```
> github.com/akupila/goddd go test ./name  
ok      github.com/akupila/goddd/name    0.001s
```

package name

import "testing"

```
func TestFormatName(t *testing.T) {  
    tests := []struct {  
        input string  
        want  string  
    }{  
        {"antti", "Antti"},  
        {"antti kupila", "Antti Kupila"},  
        {"ANTTI", "Antti"},  
    }  
    for _, tt := range tests {  
        t.Run(tt.input, func(t *testing.T) {  
            got := FormatName(tt.input)  
            if got != tt.want {  
                t.Errorf("FormatName() got = %q, want = %q", got, tt.want)  
            }  
        })  
    }  
}
```

```
> github.com/akupila/goddd go test ./name
--- FAIL: TestFormatName (0.00s)
    --- FAIL: TestFormatName/ANTTI (0.00s)
        name_test.go:20: FormatName() got = "ANTTI", want = "Antti"
    --- FAIL: TestFormatName/antti_kupila (0.00s)
        name_test.go:20: FormatName() got = "Antti kupila", want = "Antti Kupila"
FAIL
FAIL    github.com/akupila/goddd/name    0.001s
```

```
package name
```

```
import (  
    "strings"  
    "unicode"  
)
```

```
func FormatName(s string) string {  
    s = strings.ToLower(s)  
    parts := strings.Split(s, " ")  
    for i, part := range parts {  
        first := rune(part[0])  
        remain := part[1:]  
        parts[i] = string(unicode.ToUpper(first)) + remain  
    }  
    return strings.Join(parts, " ")  
}
```

```
> github.com/akupila/goddd go test ./name
ok      github.com/akupila/goddd/name    0.001s
```

```
> github.com/akupila/goddd golint ./name
```

```
name/name.go:5:1: exported function FormatName should have comment or be unexported
```

```
package name
```

```
import (  
    "strings"  
    "unicode"  
)
```

```
// returns the user's name so it is properly capitalized  
func FormatName(s string) string {  
    s = strings.ToLower(s)  
    parts := strings.Split(s, " ")  
    for i, part := range parts {  
        first := rune(part[0])  
        remain := part[1:]  
        parts[i] = string(unicode.ToUpper(first)) + remain  
    }  
    return strings.Join(parts, " ")  
}
```



```
> github.com/akupila/goddd golint ./name  
name/name.go:8:1: comment on exported function FormatName should be of the form "FormatName ..."
```

```
package name
```

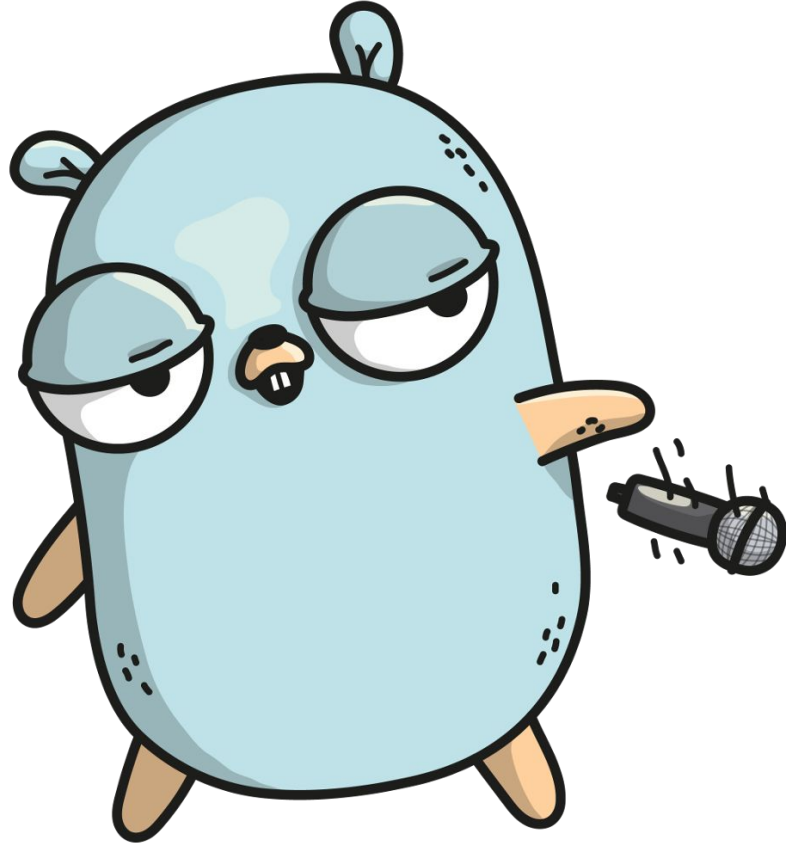
```
import (  
    "strings"  
    "unicode"  
)
```

```
// FormatName ...
```

```
func FormatName(s string) string {  
    s = strings.ToLower(s)  
    parts := strings.Split(s, " ")  
    for i, part := range parts {  
        first := rune(part[0])  
        remain := part[1:]  
        parts[i] = string(unicode.ToUpper(first)) + remain  
    }  
    return strings.Join(parts, " ")  
}
```

```
> github.com/akupila/goddd golint ./name  
> github.com/akupila/goddd echo $?
```

```
0
```



```
$ godoc -h :6060
```



Package name

```
import "github.com/akupila/goddd/name"
```

[Overview](#)[Index](#)

Overview ►

Index ▼

```
func FormatName(s string) string
```

Package files

name.go

func **FormatName**

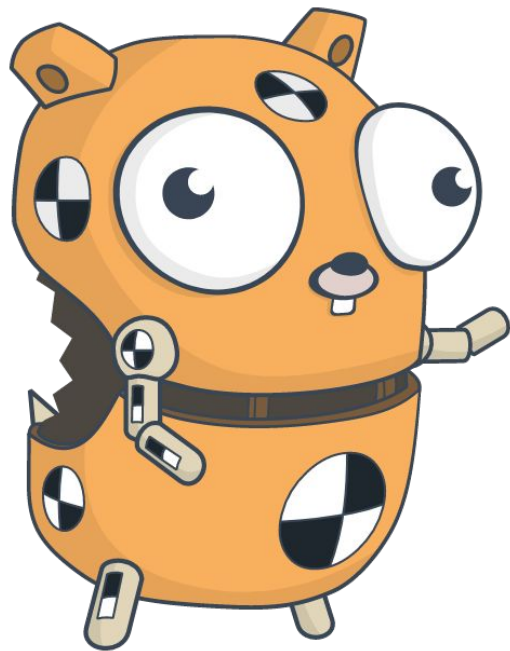
```
func FormatName(s string) string
```

FormatName ...

Build version go1.10.1.

Except as [noted](#), the content of this page is licensed under the Creative Commons Attribution 3.0 License, and code is licensed under a [BSD license](#).

[Terms of Service](#) | [Privacy Policy](#)



“~~Trivial~~
~~Good~~ code is self documenting

-- Unknown



GoDDD

GoDoc Driven Development

“
Don't document the program;
Program the document

-- Unknown

GoDoc

- Generate docs from your code
- Special formatting
- Looks good in html and in code
- Examples are executed as tests
- godoc.org automatically generates docs

<https://godoc.org/github.com/natefinch/godocgo>

<https://godoc.org/github.com/fluhus/godoc-tricks>

```
package name
```

```
// FormatName capitalizes a person's name so it can be correctly printed.  
func FormatName(s string) string {  
    return "not implemented"  
}
```

```
package name
```

```
import (  
    "fmt"  
)
```

```
func ExampleFormatName() {  
    input := "antti"  
    got := FormatName(input)  
    fmt.Println(got)  
    // Output: Antti  
}
```



Antti Kupila

@akupila



[#golang](#) pro tip: put your examples in a `_test` package so the code contains the package name and becomes copy-pasteable (as it's likely used from another package)

11:28 AM · May 16, 2019 · [Twitter Web Client](#)

||| [View Tweet activity](#)

2 Retweets **1** Like




```
package name
```

```
import (  
    "fmt"
```

```
    "github.com/akupila/goddd/name"  
)
```

```
func ExampleFormatName() {  
    input := "antti"  
    got := name.FormatName(input)  
    fmt.Println(got)  
    // Output: Antti  
}
```

```
package name_test

import (
    "fmt"

    "github.com/akupila/goddd/name"
)

func ExampleFormat() {
    input := "antti"
    got := name.Format(input)
    fmt.Println(got)
    // Output: Antti
}
```

```
package name_test
```

```
import (  
    "fmt"
```

```
  
    "github.com/akupila/goddd/name"  
)
```

```
func ExampleFormat() {  
    input := "antti"  
    got := name.Format(input)  
    fmt.Println(got)  
    // Output: Antti  
}
```

```
func ExampleFormat_familyName() {  
    input := "antti kupila"  
    got := name.Format(input)  
    fmt.Println(got)  
    // Output: Antti Kupila  
}
```



Package name

```
import "github.com/akupila/goddd/name"
```

[Overview](#)[Index](#)[Examples](#)

Overview ▼

Index ▼

```
func Format(s string) string
```

Examples

[Format](#)[Format \(FamilyName\)](#)

Package files

[name.go](#)

func **Format**

```
func Format(s string) string
```

Format formats a person's name so it can be correctly printed.

▼ Example

Code:

```
input := "antti"  
got := name.Format(input)  
fmt.Println(got)
```

Output:

```
Antti
```

▸ [Example \(FamilyName\)](#)



Jaana B. Dogan

@rakyll



A good API is not just easy to use but also hard to misuse.

6:43 PM · Oct 16, 2017 · [Twitter Web Client](#)

382 Retweets **884** Likes



package name

```
// Format formats a person's name so it can be correctly printed.
//
// The returned value will always have the first letter capitalized, while the
// remaining letters are lowercase, as determined by the unicode package.
//
//   alice -> Alice
//
// The input may in any combination of uppercase/lowercase characters.
// Characters that are uppercase may be converted to lowercase if needed.
//
// In case the name contains hyphenated names, they are both capitalized.
func Format(s string) string {
    return "not implemented"
}
```

Package name

```
import "github.com/akupila/goddd/name"
```

[Overview](#)[Index](#)[Examples](#)

Overview ▼

Index ▼

```
func Format(s string) string
```

Examples

[Format](#)[Format \(FamilyName\)](#)

Package files

[name.go](#)

func Format

```
func Format(s string) string
```

Format formats a person's name so it can be correctly printed.

The returned value will always have the first letter capitalized, while the remaining letters are lowercase, as determined by the unicode package.

```
alice -> Alice
```

The input may in any combination of uppercase/lowercase characters. Characters that are uppercase may be converted to lowercase if needed.

In case the name contains hyphenated names, they are both capitalized.

▸ [Example](#)

▸ [Example \(FamilyName\)](#)

```
// Package name provides utilities for working with names of people.  
package name
```


Package name

```
import "github.com/akupila/goddd/name"
```

[Overview](#)[Index](#)[Examples](#)

Overview ▼

Package name provides utilities for working with names of people.

Index ▼

```
func Format(s string) string
```

Examples

```
Format
```

```
Format (FamilyName)
```

Package files

```
doc.go name.go
```

func Format

```
func Format(s string) string
```

Format formats a person's name so it can be correctly printed.

The returned value will always have the first letter capitalized, while the remaining letters are lowercase, as determined by the unicode package.

```
alice -> Alice
```

The input may in any combination of uppercase/lowercase characters. Characters that are uppercase may be converted to lowercase if needed.

In case the name contains hyphenated names, they are both capitalized.

► [Example](#)

► [Example \(FamilyName\)](#)

```

package name_test

import (
    "fmt"
    "testing"

    "github.com/akupila/goddd/name"
)

func TestFormat(t *testing.T) {
    tests := []struct{
        input string
        want  string
    }{
        {"antti", "Anttti"},
        {"ANTTI", "Anttti"},
        {"antti kupila", "Anttti Kupila"},
        {"josé-maria gonzales", "José-Maria Gonzales"},
    }
    for _, tt := range tests {
        t.Run(tt.input, func(t *testing.T) {
            got := name.Format(tt.input)
            if got != tt.want {
                t.Errorf("FormatName() got = %q, want = %q", got, tt.want)
            }
        })
    }
}

```

package name

```
// Format formats a person's name so it can be correctly printed.
//
// The returned value will always have the first letter capitalized, while the
// remaining letters are lowercase, as determined by the unicode package.
//
//   alice -> Alice
//
// The input may in any combination of uppercase/lowercase characters.
// Characters that are uppercase may be converted to lowercase if needed.
//
// In case the name contains hyphenated names, they are both capitalized.
func Format(s string) string {
    return "not implemented"
}
```

func Title

```
func Title(s string) string
```

Title returns a copy of the string `s` with all Unicode letters that begin words mapped to their title case.

BUG(rsc): The rule Title uses for word boundaries does not handle Unicode punctuation properly.

▼ Example

```
package main

import (
    "fmt"
    "strings"
)

func main() {
    fmt.Println(strings.Title("her royal highness"))
}
```

Her Royal Highness

Program exited.

Run

Format

Share

package name

```
// Format formats a person's name so it can be correctly printed.
//
// The returned value will always have the first letter capitalized, while the
// remaining letters are lowercase, as determined by the unicode package.
//
//   alice -> Alice
//
// The input may in any combination of uppercase/lowercase characters.
// Characters that are uppercase may be converted to lowercase if needed.
//
// In case the name contains hyphenated names, they are both capitalized.
func Format(s string) string {
    return strings.Title(strings.ToLower(s))
}
```

```
> github.com/akupila/goddd go test ./name  
ok      github.com/akupila/goddd/name    0.001s
```

Work smarter
Not harder





Brad Fitzpatrick ✓ @bradfitz · Sep 6, 2018

After thinking about it on & off for months, I've started sketching out a new [#golang](#) HTTP client package during Dylan naps.

I'm much more excited about it, now that I can see it in godoc, even if the implementation is largely panic("TODO").

Doc later. On vacation. Fun only. :)



8



5



166



Antti Kupila @akupila · Sep 8, 2018

Godoc driven development 🔥



1



1



Brad Fitzpatrick ✓

@bradfitz

Replying to [@akupila](#)

Always!

9:39 PM · Sep 8, 2018 from [Wicko, Polska](#) · [Twitter for Android](#)




```
$ cloc src --include-ext=go
```

```
5661 text files.
```

```
5525 unique files.
```

```
1540 files ignored.
```

```
3 errors:
```

```
Line count, exceeded timeout: src/vendor/golang.org/x/net/idna/tables10.0.0.go
```

```
Line count, exceeded timeout: src/vendor/golang.org/x/net/idna/tables11.0.0.go
```

```
Line count, exceeded timeout: src/vendor/golang.org/x/net/idna/tables9.0.0.go
```

```
github.com/AlDanial/cloc v 1.82 T=21.35 s (193.3 files/s, 86838.4 lines/s)
```

Language	files	blank	comment	code
Go	4127	138764	221333	1493796
SUM:	4127	138764	221333	1493796



That's all Folks!

type **Speaker**

```
type Speaker struct {  
    Name string  
}
```

func (Speaker) QA

```
func (s Speaker) QA(questions []Question) []Answer
```

QA starts Questions & Answers for the given speaker. The speaker will attempt to answer any questions.

Speaker panics if answer to question is not known.