POLSKO-JAPOŃSKA
AKADEMIA TECHNIK
KOMPUTEROWYCH

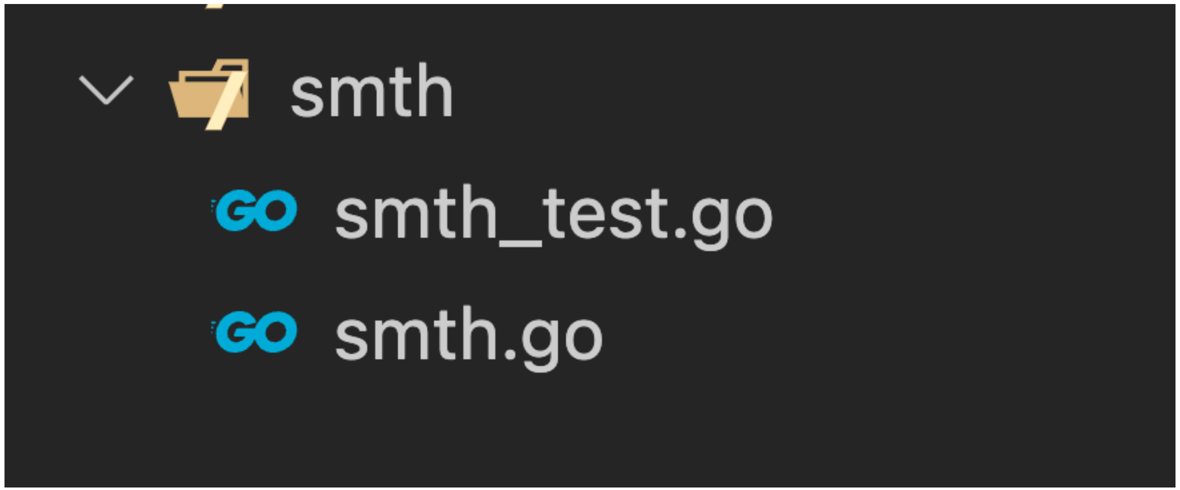
Testowanie kodu

Po co testy?

```go
package main

import "testing"

func TestHelloWorld(t *testing.T) {
    t.Fatal("Hello World")
}
```

Po co testy?

```
❯ go test
--- FAIL: TestHelloWorld (0.00s)
    main_test.go:6: Hello World
FAIL
exit status 1
FAIL	apSrv	0.403s
```



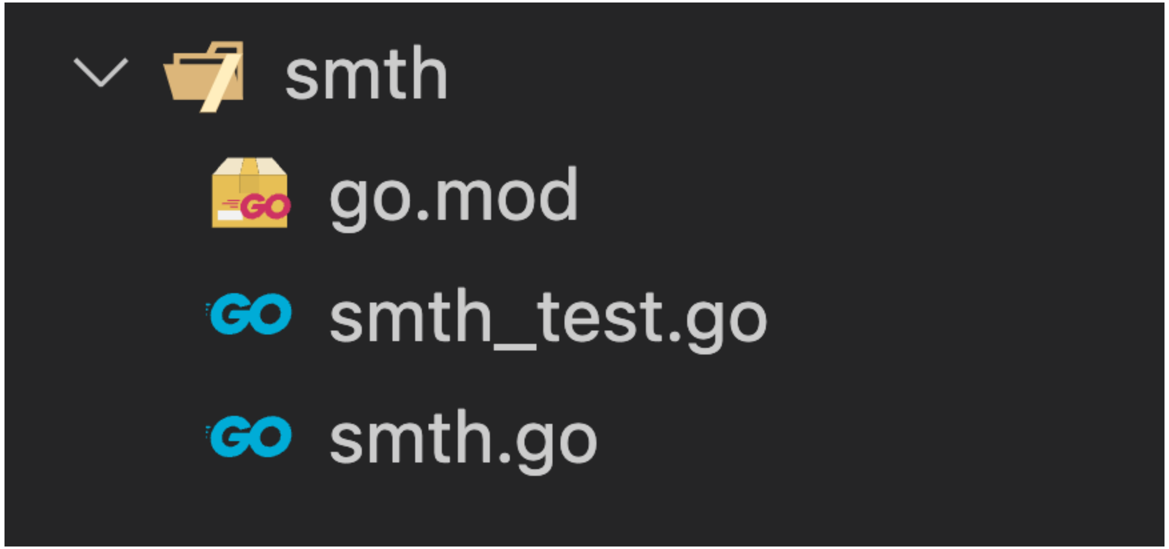
```
smth
    smth_test.go
    smth.go
```

Pierwszy test

```go
package smth

func HelloWorld() string {
    return "Goodbye, Mars!"
}
```

Nie działa?

```
❯ go test
go: go.mod file not found in current directory or any parent directory; see 'go help modules'

❯ go mod init smth
go: creating new go.mod: module smth
go: to add module requirements and sums:
	go mod tidy

❯ go test
--- FAIL: TestHelloWorld (0.00s)
    smth_test.go:9: want: Hello, World!; got: Goodbye, Mars!
FAIL
exit status 1
FAIL	smth	0.517s
```

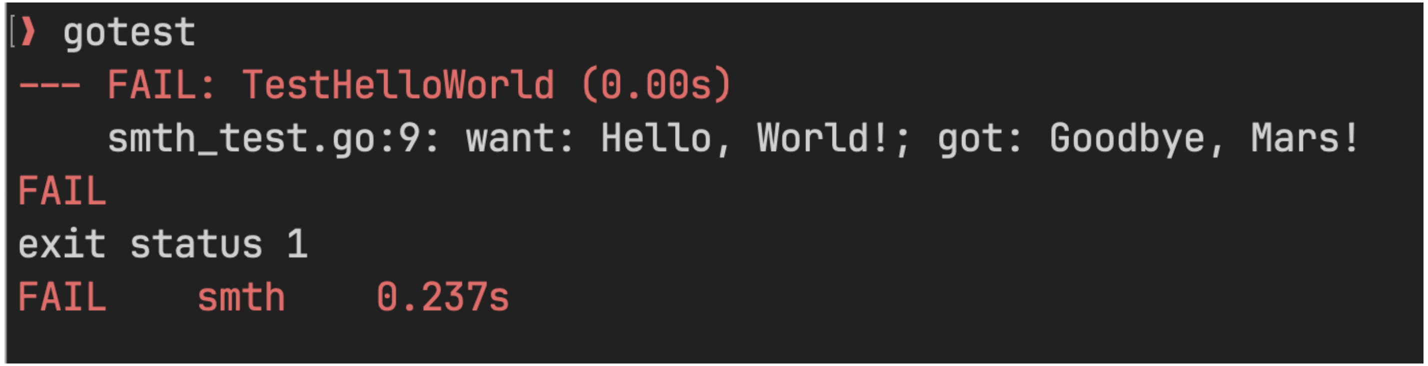


```
smth
  go.mod
  smth_test.go
  smth.go
```

Pierwszy test

```go
func TestHelloWorld(t *testing.T) {
    want := "Hello, World!"
    got := HelloWorld()
    if got != want {
        t.Errorf("want: %s; got: %s", want, got)
    }
}
```

```
❯ go install github.com/rakyll/gotest@latest
```



```
❯ gotest
--- FAIL: TestHelloWorld (0.00s)
    smth_test.go:9: want: Hello, World!; got: Goodbye, Mars!
FAIL
exit status 1
FAIL    smth    0.237s
```

Fatal / Error

```go
package smth

import "errors"

func Divide(a, b int) (int, error) {
    if b == 0 {
        return 0, errors.New("cannot divide by zero")
    }
    return a / b, nil
}
```

Fatal / Error

```go
func TestDivide(t *testing.T) {
    want := 5
    got, err := Divide(10, 0)

    if err != nil {
        t.Fatal(err)
    }

    if got != want {
        t.Fatalf("want: %d; got: %d", want, got)
    }
}
```

Fatal / Error

```
› go test
--- FAIL: TestDivide (0.00s)
    smth_test.go:10: cannot divide by zero
FAIL
exit status 1
FAIL    smth    0.504s
```

Fatal / Error

```
func TestDivide(t *testing.T) {
    want := 5
    got, err := Divide(10, 0)

    if err != nil {
        t.Error(err)
    }

    if got != want {
        t.Errorf("want: %d; got: %d", want, got)
    }
}
```

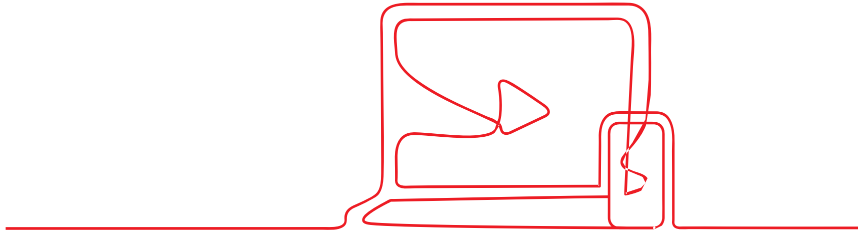
Fatal / Error

```
❯ go test
--- FAIL: TestDivide (0.00s)
    smth_test.go:10: cannot divide by zero
    smth_test.go:14: want: 5; got: 0
FAIL
exit status 1
FAIL    smth    0.468s
```

```
❯ go get github.com/stretchr/testify
```

Testify - Thou Shalt Write Tests

```go
import (
    "github.com/stretchr/testify/assert"
    "testing"
)

func TestDivide(t *testing.T) {
    want := 5
    got, err := Divide(10, 0)

    assert.Nil(t, err)
    assert.Equal(t, want, got)
}
```



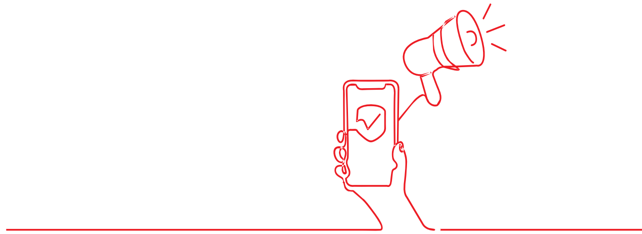
Testify

```
❯ go test
--- FAIL: TestDivide (0.00s)
    smth_test.go:12:
            Error Trace:    /Users/landmaj/Documents/pjatk/smth/smth_test.go:12
            Error:          Expected nil, but got: &errors.errorString{s:"cannot divide by zero"}
            Test:           TestDivide
    smth_test.go:13:
            Error Trace:    /Users/landmaj/Documents/pjatk/smth/smth_test.go:13
            Error:          Not equal:
                            expected: 5
                            actual  : 0
            Test:           TestDivide
FAIL
exit status 1
FAIL    smth    0.453s
```

Testujemy prawdziwy kod

```
func CalculatePrice(price, tax int64) (net, gross float64) {
    // za długa funkcja na slajd
}
```

https://go.dev/play/p/vqNIIt4PZi2



Testujemy prawdziwy kod

```
func TestCalculatePrice(t *testing.T) {
    wantNet, wantGross := 18.51, 19.99
    gotNet, gotGross := CalculatePrice(1999, 8)

    assert.Equal(t, wantNet, gotNet)
    assert.Equal(t, wantGross, gotGross)
}
```

Jakie przypadki testowe?

- Różne wartości ceny i podatku.
- Zerowa cena.
- Zerowy podatek nie występuje, ale dobrze byłoby go sprawdzić.
- Funkcja nie zwraca błędów, więc wartości ujemne można pominąć, bo są bez sensu.
- Wszelkie przypadki, o których wiemy lub spodziewamy się, że mogą spowodować błąd.

Jakie przypadki testowe?

- 1999, 8
- 3499, 8
- 11499, 23
- 0, 23

Sub-testy

```go
func TestCalculatePrice(t *testing.T) {
    t.Run("typical case", func(t *testing.T) {
        wantNet, wantGross := 18.51, 19.99
        gotNet, gotGross := CalculatePrice(1999, 8)

        assert.Equal(t, wantNet, gotNet)
        assert.Equal(t, wantGross, gotGross)
    })

    t.Run("free trial", func(t *testing.T) {
        wantNet, wantGross := 0.0, 0.0
        gotNet, gotGross := CalculatePrice(0, 8)

        assert.Equal(t, wantNet, gotNet)
        assert.Equal(t, wantGross, gotGross)
    })
}
```

Sub-testy

```
❯ go test -v
=== RUN   TestCalculatePrice
=== RUN   TestCalculatePrice/typical_case
=== RUN   TestCalculatePrice/free_trial
--- PASS: TestCalculatePrice (0.00s)
    --- PASS: TestCalculatePrice/typical_case (0.00s)
    --- PASS: TestCalculatePrice/free_trial (0.00s)
PASS
ok      smth    0.144s
```

DRY

```go
func TestCalculatePrice(t *testing.T) {
    tests := []struct {
        name      string
        price     int64
        tax       int64
        wantNet   float64
        wantGross float64
    }{
        {"typical case", 1999, 8, 18.51, 19.99},
        {"free trial", 0, 8, 0, 0},
    }

    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            gotNet, gotGross := CalculatePrice(tt.price, tt.tax)
            assert.Equal(t, tt.wantNet, gotNet)
            assert.Equal(t, tt.wantGross, gotGross)
        })
    }
}
```

Self-documenting code

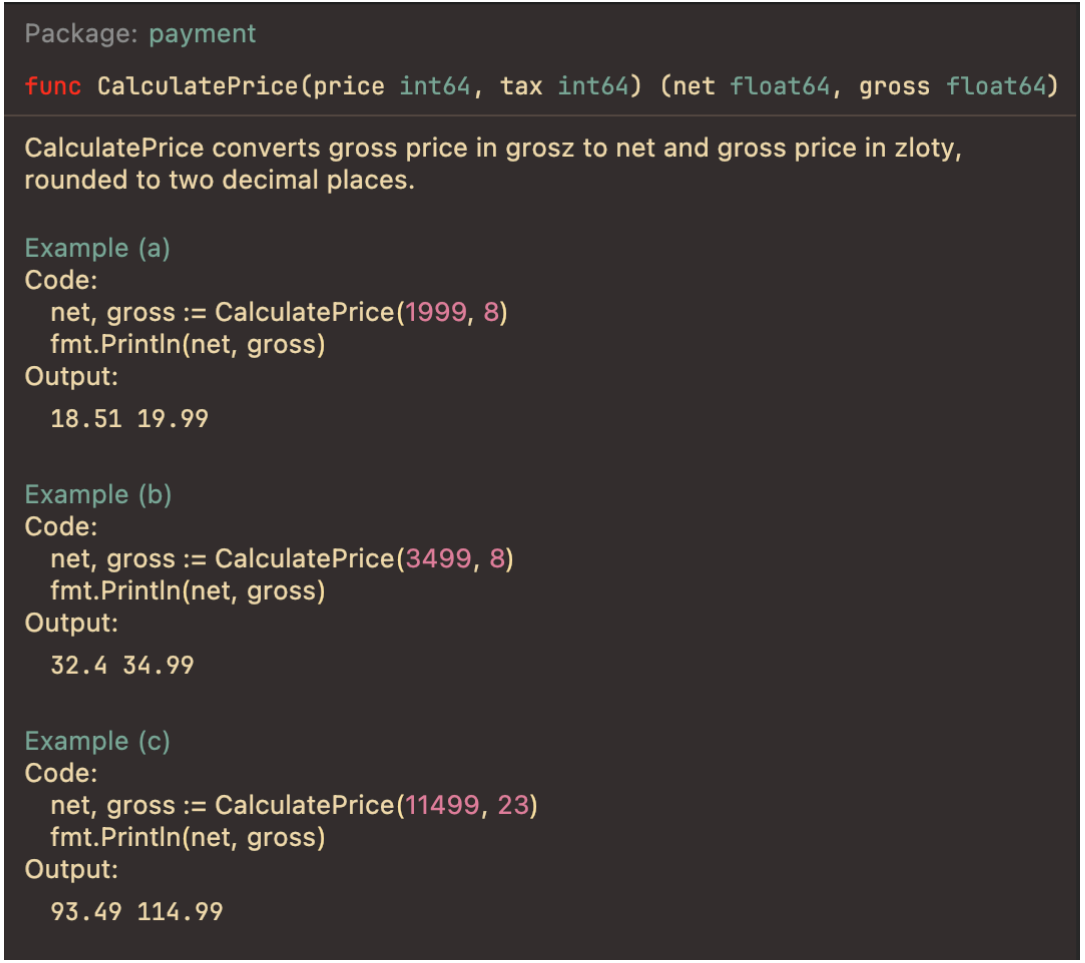
```
func ExampleCalculatePrice() {
    net, gross := CalculatePrice(1999, 8)
    fmt.Println(net, gross)
    // Output: 18.51 19.99
}
```

Self-documenting code

```go
func ExampleCalculatePrice_a() {
    net, gross := CalculatePrice(1999, 8)
    fmt.Println(net, gross)
    // Output: 18.51 19.99
}

func ExampleCalculatePrice_b() {
    net, gross := CalculatePrice(3499, 8)
    fmt.Println(net, gross)
    // Output: 32.4 34.99
}

func ExampleCalculatePrice_c() {
    net, gross := CalculatePrice(11499, 23)
    fmt.Println(net, gross)
    // Output: 93.49 114.99
}

func ExampleCalculatePrice_d() {
    net, gross := CalculatePrice(0, 23)
    fmt.Println(net, gross)
    // Output: 0 0
}
```

Self-documenting code

```
❯ go test -v
=== RUN   ExampleCalculatePrice_a
--- PASS: ExampleCalculatePrice_a (0.00s)
=== RUN   ExampleCalculatePrice_b
--- PASS: ExampleCalculatePrice_b (0.00s)
=== RUN   ExampleCalculatePrice_c
--- PASS: ExampleCalculatePrice_c (0.00s)
=== RUN   ExampleCalculatePrice_d
--- PASS: ExampleCalculatePrice_d (0.00s)
PASS
ok      smth    0.350s
```



Package: payment

```
func CalculatePrice(price int64, tax int64) (net float64, gross float64)
```

CalculatePrice converts gross price in grosz to net and gross price in zloty, rounded to two decimal places.

Example (a)
Code:

```
net, gross := CalculatePrice(1999, 8)
fmt.Println(net, gross)
```

Output:

```
18.51 19.99
```

Example (b)
Code:

```
net, gross := CalculatePrice(3499, 8)
fmt.Println(net, gross)
```

Output:

```
32.4 34.99
```

Example (c)
Code:

```
net, gross := CalculatePrice(11499, 23)
fmt.Println(net, gross)
```

Output:

```
93.49 114.99
```

func Abs

```
func Abs(x float64) float64
```

Abs returns the absolute value of x.

Special cases are:

```
Abs(±Inf) = +Inf
Abs(NaN) = NaN
```

▾ Example

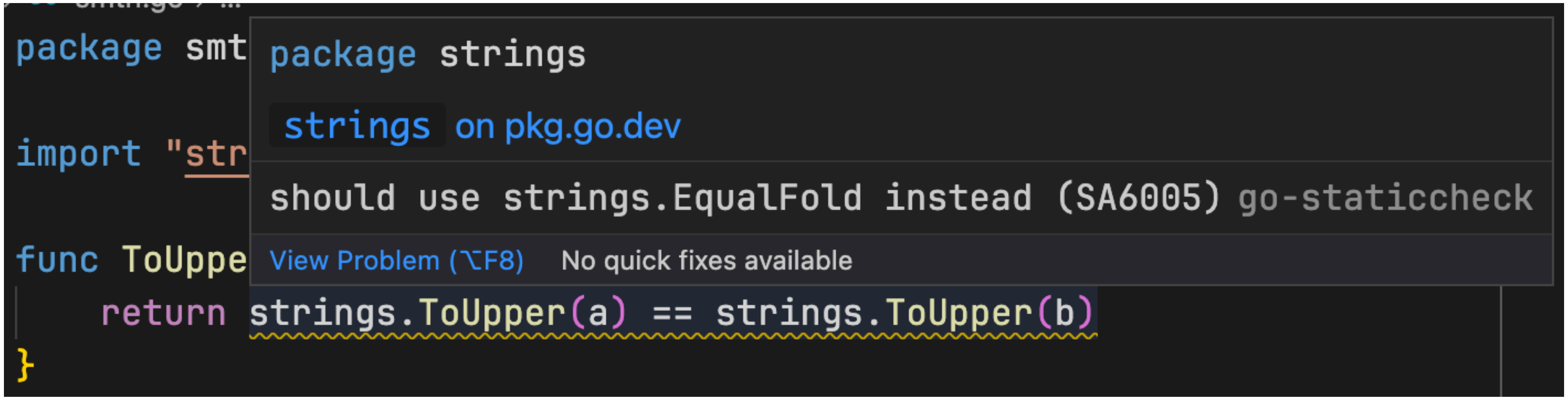
```
package main

import (
    "fmt"
    "math"
)

func main() {
    x := math.Abs(-2)
    fmt.Printf("%.1f\n", x)

    y := math.Abs(2)
    fmt.Printf("%.1f\n", y)
}
```

```
Output:

2.0
2.0
```

Share Format Run

Benchmark

```go
func ToUpper(a, b string) bool {
    return strings.ToUpper(a) == strings.ToUpper(b)
}

func ToLower(a, b string) bool {
    return strings.ToLower(a) == strings.ToLower(b)
}

func EqualFold(a, b string) bool {
    return strings.EqualFold(a, b)
}
```



```
package smt
import "str
func ToUppe
    return strings.ToUpper(a) == strings.ToUpper(b)
}
```

```
package strings

strings on pkg.go.dev

should use strings.EqualFold instead (SA6005) go-staticcheck
```

Benchmark

```
func BenchmarkToUpper(b *testing.B) {
    for i := 0; i < b.N; i++ {
        ToUpper("hello world", "Hello World")
    }
}
```

Benchmark

```
func BenchmarkToUpper(b *testing.B) {
    for i := 0; i < b.N; i++ {
        ToUpper("hello world", "Hello World")
    }
}

func BenchmarkToLower(b *testing.B) {
    for i := 0; i < b.N; i++ {
        ToLower("hello world", "Hello World")
    }
}

func BenchmarkEqualFold(b *testing.B) {
    for i := 0; i < b.N; i++ {
        EqualFold("hello world", "Hello World")
    }
}
```

Benchmark

```
❯ go test -bench=.
```

Benchmark

```
❯ go test -bench=.
goos: darwin
goarch: arm64
pkg: smth
BenchmarkToUpper-8        14763910         81.42 ns/op
BenchmarkToLower-8        25610630         46.95 ns/op
BenchmarkEqualFold-8     131347342          9.159 ns/op
PASS
ok      smth    5.743s
```

Benchmark

```go
func BenchmarkSleep(b *testing.B) {
    tests := []struct {
        name     string
        duration int64
        want     int64
    }{
        {"zero", 0, 0},
        {"one", 1, 1},
        {"hundred", 100, 100},
    }
    for _, tt := range tests {
        b.Run(tt.name, func(b *testing.B) {
            time.Sleep(time.Duration(tt.duration) * time.Millisecond)
            assert.LessOrEqual(b, b.Elapsed().Milliseconds(), tt.want)
        })
    }
}
```

Benchmark

```
❯ go test -bench=.
goos: darwin
goarch: arm64
pkg: smth
BenchmarkEqualFold/zero-8          1000000000          0.0000027 ns/op
BenchmarkEqualFold/one-8           1000000000          0.001145 ns/op
--- FAIL: BenchmarkEqualFold/hundred
    smth_test.go:23:
            Error Trace:    /Users/landmaj/Documents/pjatk/smth/smth_test.go:23
                                /opt/homebrew/Cellar/go/1.20.2/libexec/src/testing/benchmark.go:193
                                /opt/homebrew/Cellar/go/1.20.2/libexec/src/testing/benchmark.go:233
                                /opt/homebrew/Cellar/go/1.20.2/libexec/src/runtime/asm_arm64.s:1172
            Error:          "101" is not less than or equal to "100"
            Test:           BenchmarkEqualFold/hundred
--- FAIL: BenchmarkEqualFold
FAIL
exit status 1
FAIL    smth    0.358s
```

Benchmark

Która operacja jest najszybsza?

- string (+=)
- bytes.Buffer
- strings.Builder

To zależy od długości dodawanego ciągu znaków.

Mocki

```go
import "time"

type Clock interface {
    Now() time.Time
}

func FormattedTime(clock Clock) string {
    return clock.Now().Format("15:04")
}
```

Mocki

```go
type fakeClock struct {
    t time.Time
}

func (fc fakeClock) Now() time.Time {
    return fc.t
}

func TestFormattedTime(t *testing.T) {
    want := "12:34"
    got := FormattedTime(fakeClock{t: time.Date(0, 0, 0, 12, 34, 0, 0, time.UTC)})
    assert.Equal(t, want, got)
}
```

Mocki

```
type fakeClock struct {
    t time.Time
}

func (fc fakeClock) Now() time.Time {
    return fc.t
}

func TestFormattedTime(t *testing.T) {
    want := "12:34"
    got := FormattedTime(fakeClock{t: time.Date(0, 0, 0, 12, 34, 0, 0, time.UTC)})
    assert.Equal(t, want, got)
}
```

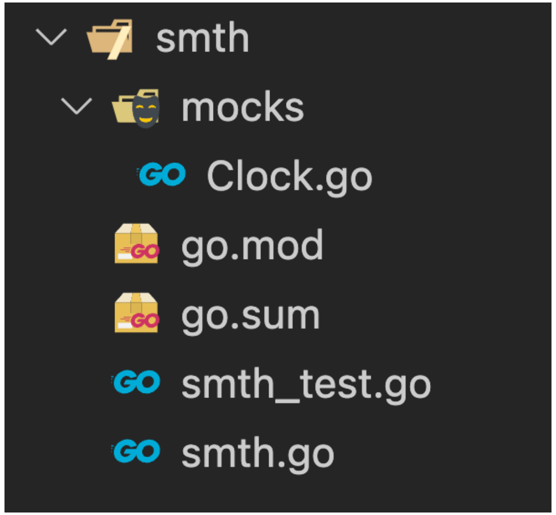
```
❯ go install github.com/vektra/mockery/v2@latest
```

Mocki

```
//go:generate mockery --name Clock
type Clock interface {
    Now() time.Time
}
```

Mocki

- go generate



```
smth
  mocks
    Clock.go
  go.mod
  go.sum
  smth_test.go
  smth.go
```

Mocki

```
func TestFormattedTime(t *testing.T) {
    want := "12:34"
    mock := mocks.NewClock(t)
    mock.On("Now").Return(time.Date(2021, 1, 1, 12, 34, 0, 0, time.UTC))
    got := FormattedTime(mock)
    assert.Equal(t, want, got)
}
```