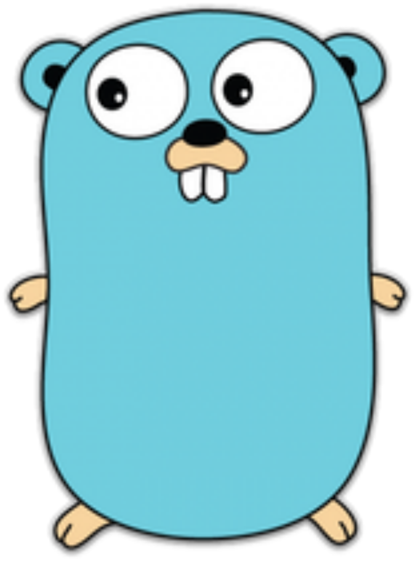
Golang Basic



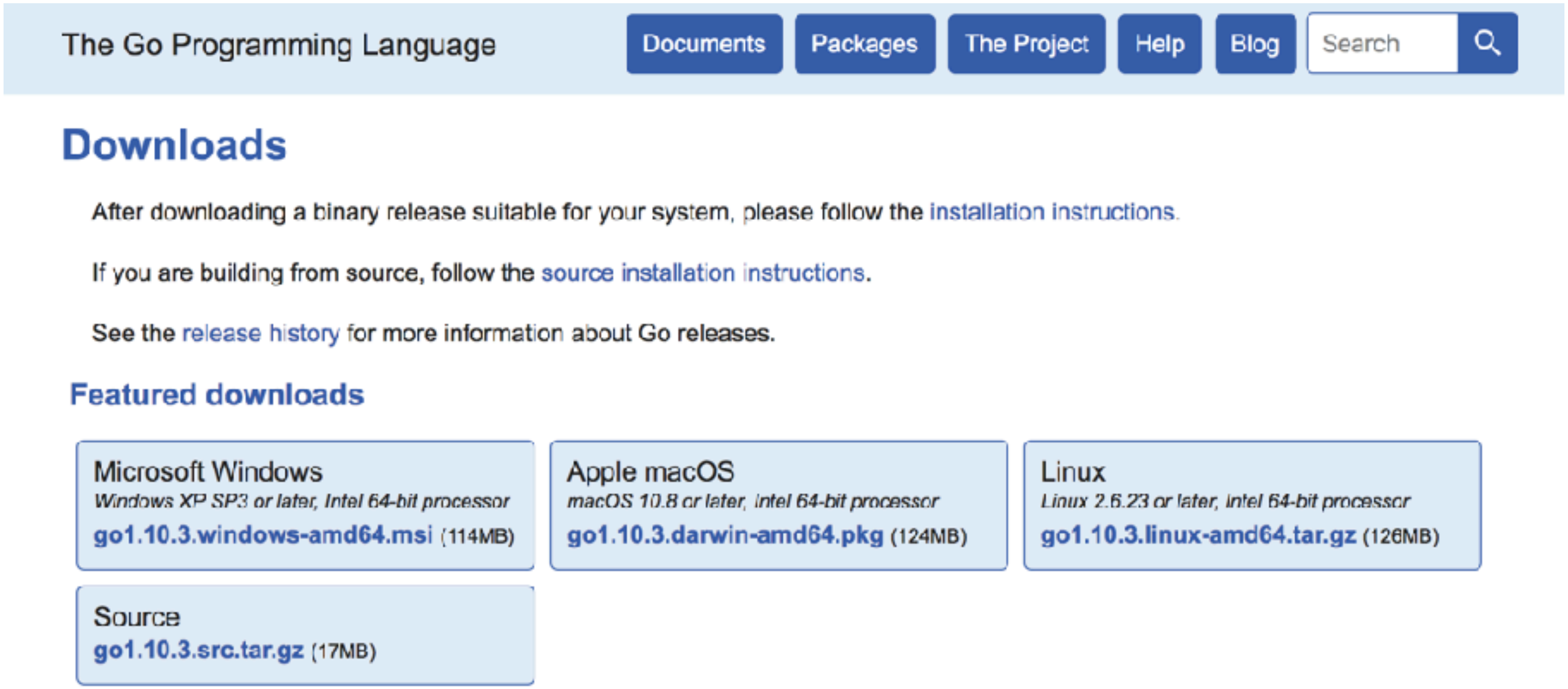
Agenda

- Day 1
  - Introduction
  - Go Installation
  - understand Go workspace with GOPATH
  - basic go command (env, fmt, test, build, run, install)
  - build runnable binary (windows, linux/MacOS)
  - basic syntax with TDD
  - how to write test using testing library

Agenda (2)

- Day 2
  - build Web Application with GoLang
    - User interface
    - REST API
  - Test
    - api handler with httptest

Go Installation



Downloads

After downloading a binary release suitable for your system, please follow the installation instructions.

If you are building from source, follow the source installation instructions.

See the release history for more information about Go releases.

Featured downloads

Microsoft Windows
Windows XP SP3 or later, Intel 64-bit processor
go1.10.3.windows-amd64.msi (114MB)

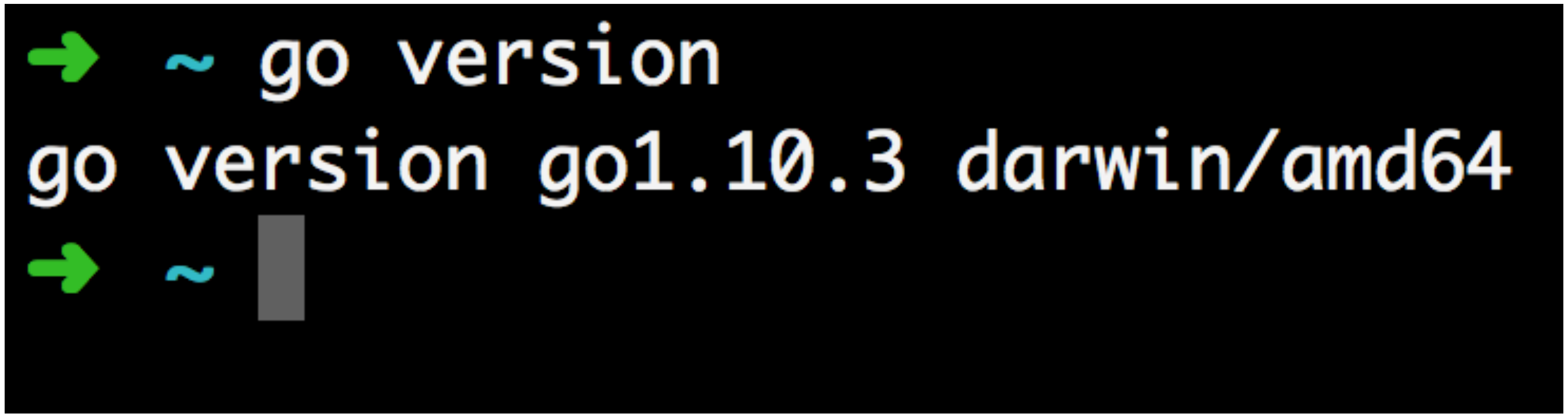
Apple macOS
macOS 10.8 or later, Intel 64-bit processor
go1.10.3.darwin-amd64.pkg (124MB)

Linux
Linux 2.6.23 or later, Intel 64-bit processor
go1.10.3.linux-amd64.tar.gz (126MB)

Source
go1.10.3.src.tar.gz (17MB)

https://golang.org/dl/

Go Installation



```
➜  ~ go version
go version go1.10.3 darwin/amd64
➜  ~ 
```

Set GOPATH

Windows:

```
setx GOPATH %USERPROFILE%/<go-path>
```

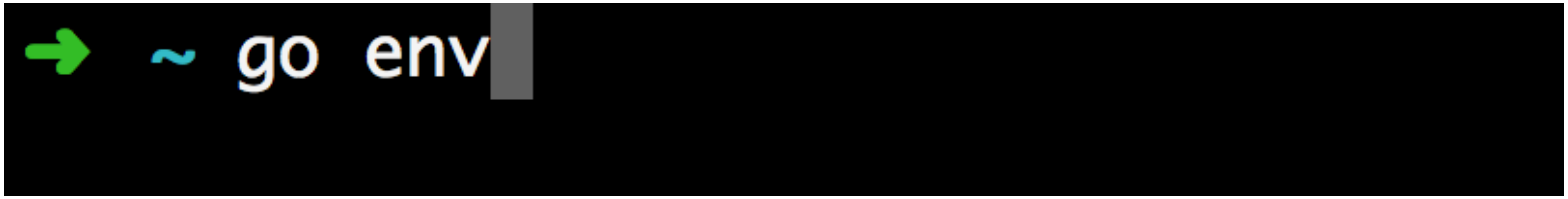
or

Control Panel -> System -> Advanced -> Environment Variables

Linux/Mac:

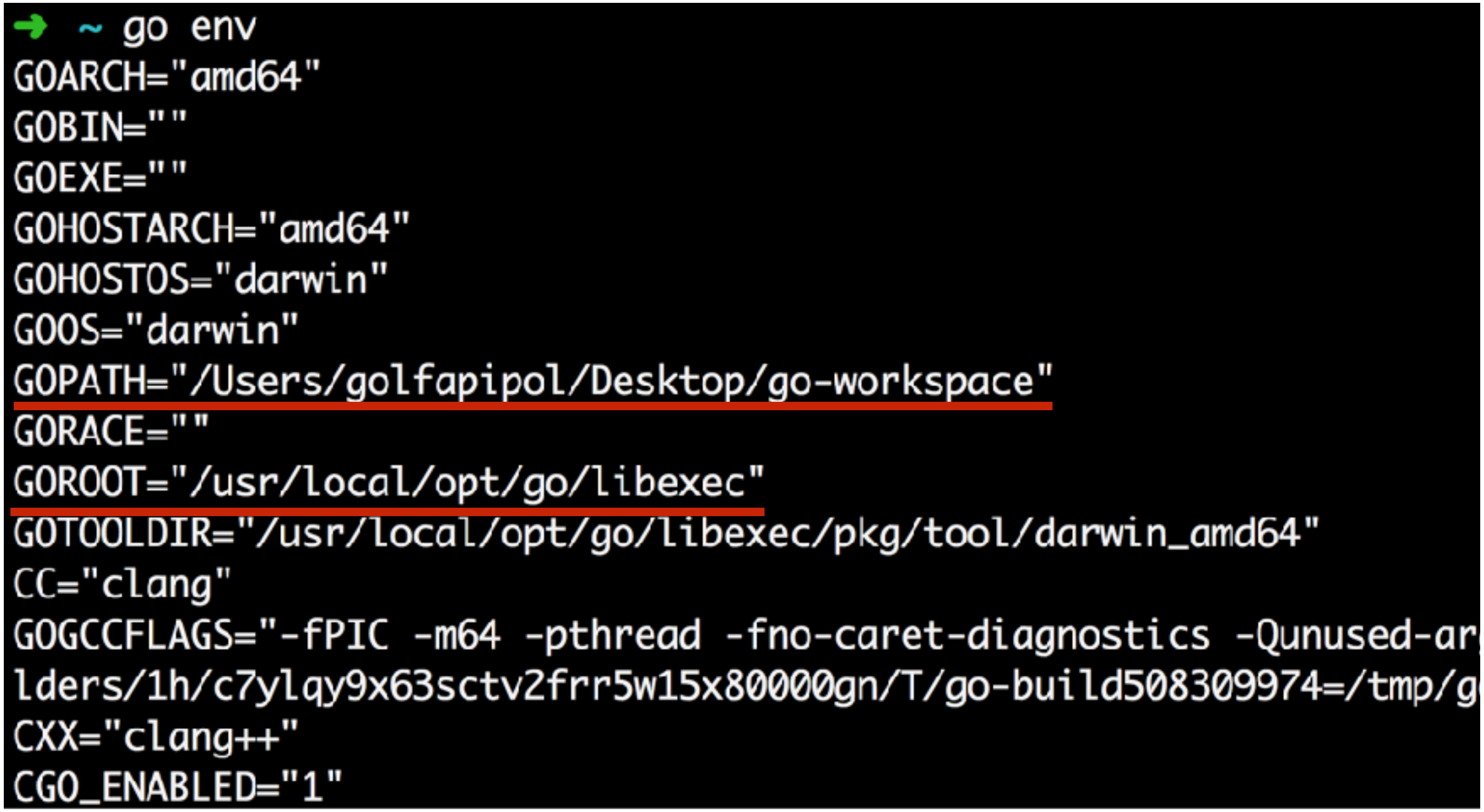
```
export GOPATH=
```

Understand Go workspace



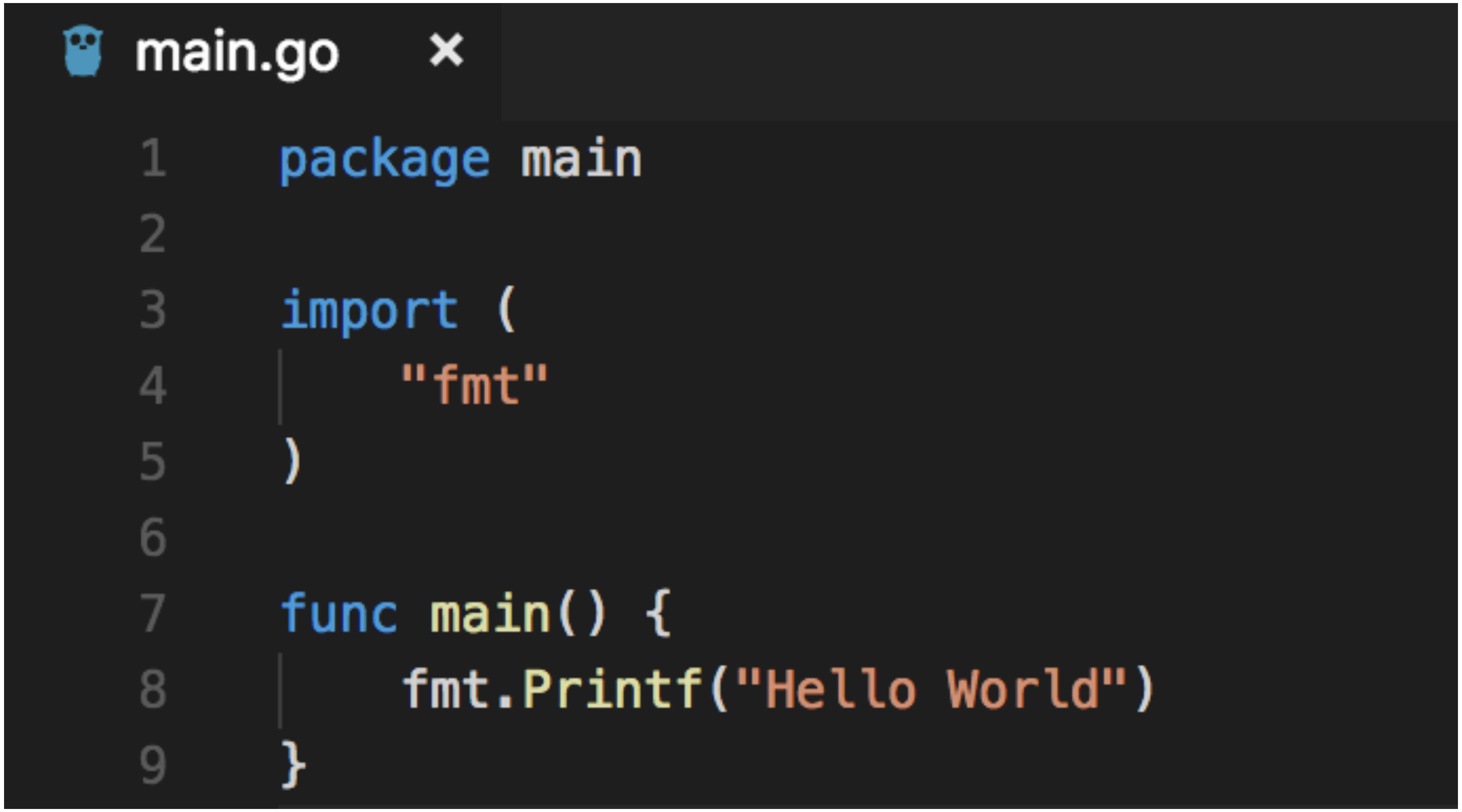
```
➜  ~ go env
```

Understand Go workspace



```
➜  ~ go env
GOARCH="amd64"
GOBIN=""
GOEXE=""
GOHOSTARCH="amd64"
GOHOSTOS="darwin"
GOOS="darwin"
GOPATH="/Users/golfapipol/Desktop/go-workspace"
GORACE=""
GOROOT="/usr/local/opt/go/libexec"
GOTOOLDIR="/usr/local/opt/go/libexec/pkg/tool/darwin_amd64"
CC="clang"
GOGCCFLAGS="-fPIC -m64 -pthread -fno-caret-diagnostics -Qunused-ar
lders/1h/c7ylqy9x63sctv2frr5w15x80000gn/T/go-build508309974=/tmp/g
CXX="clang++"
CGO_ENABLED="1"
```

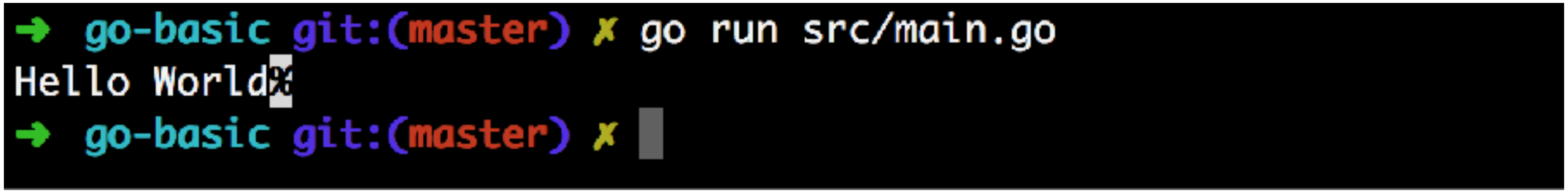
Hello World



```go
package main

import (
	"fmt"
)

func main() {
	fmt.Printf("Hello World")
}
```

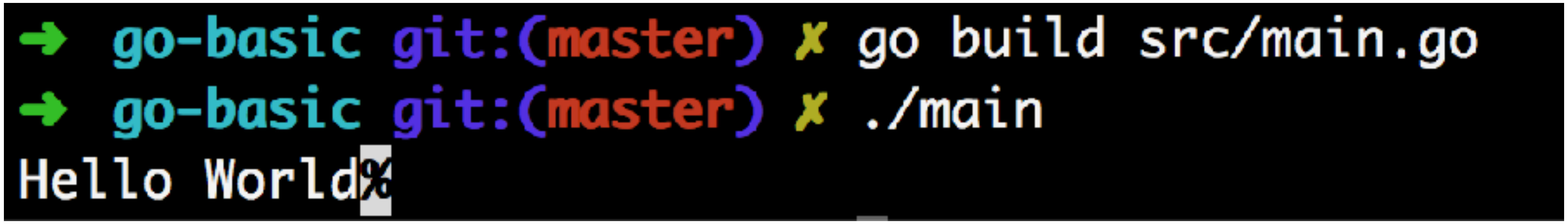
run Hello World

- run on the fly



```
➜  go-basic git:(master) ✗ go run src/main.go
Hello World%
➜  go-basic git:(master) ✗ 
```

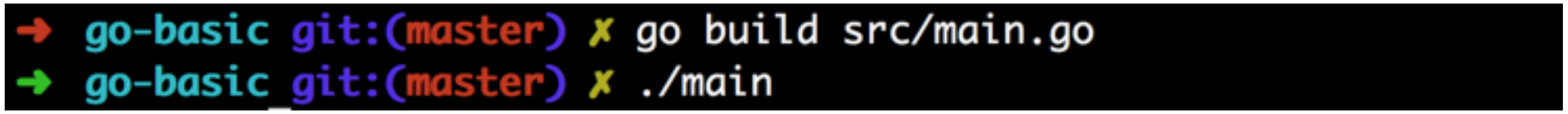
- run binary



```
➜  go-basic git:(master) ✗ go build src/main.go
➜  go-basic git:(master) ✗ ./main
Hello World%
```

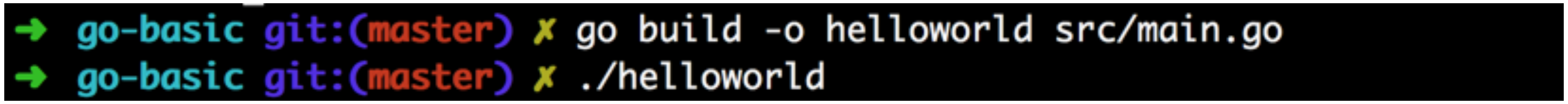
build Hello World

build



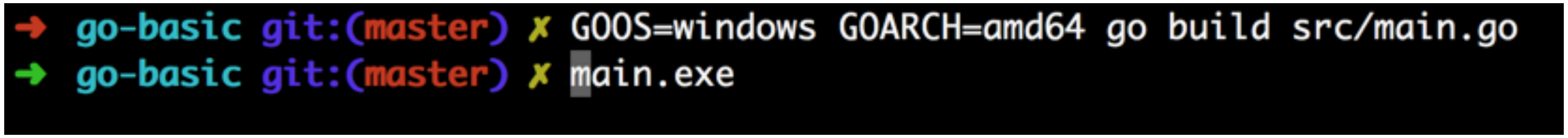
```
➜  go-basic git:(master) ✗ go build src/main.go
➜  go-basic git:(master) ✗ ./main
```

build specific output



```
➜  go-basic git:(master) ✗ go build -o helloworld src/main.go
➜  go-basic git:(master) ✗ ./helloworld
```

build for windows

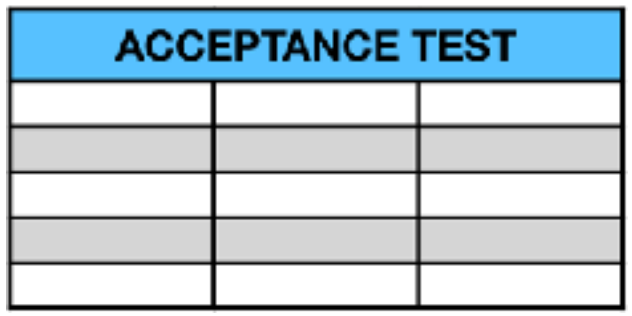


```
➜  go-basic git:(master) ✗ GOOS=windows GOARCH=amd64 go build src/main.go
➜  go-basic git:(master) ✗ main.exe
```

Start Work!

Start with Problem

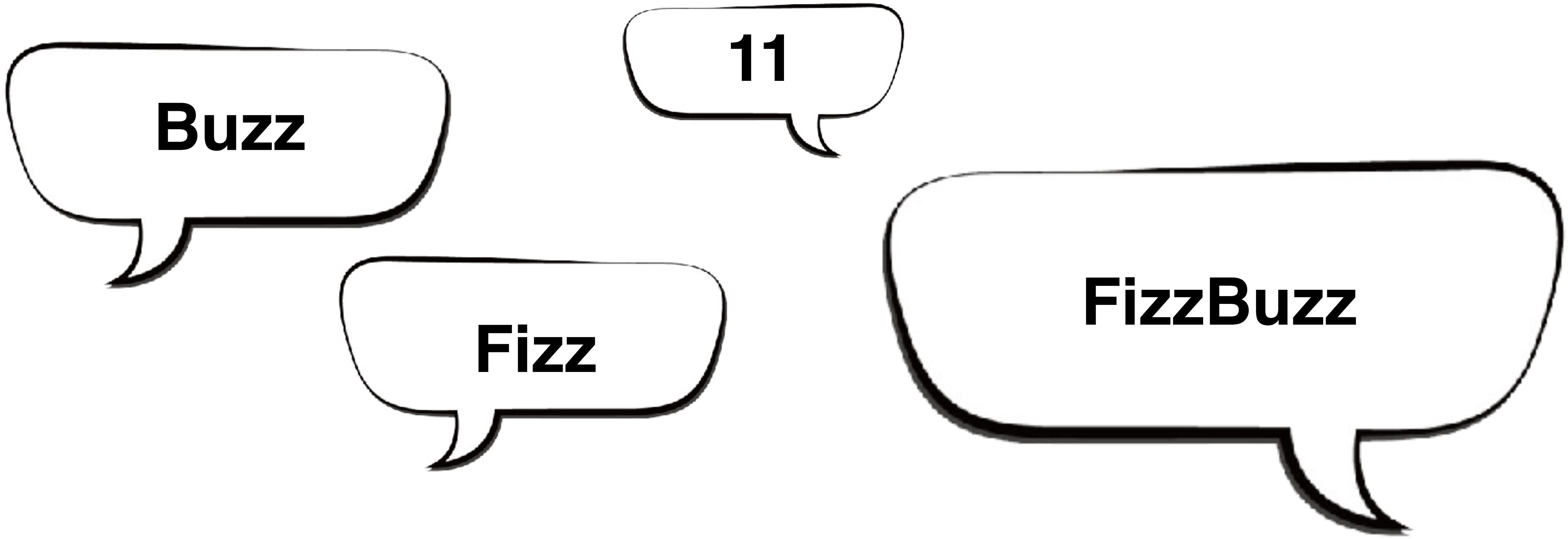
Find Business Conditions



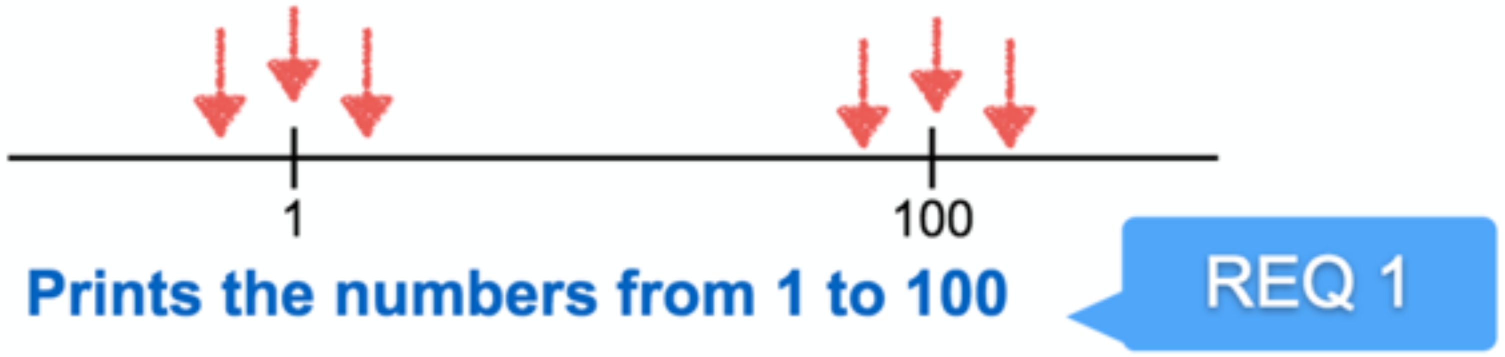
Test Design

Start with Problem

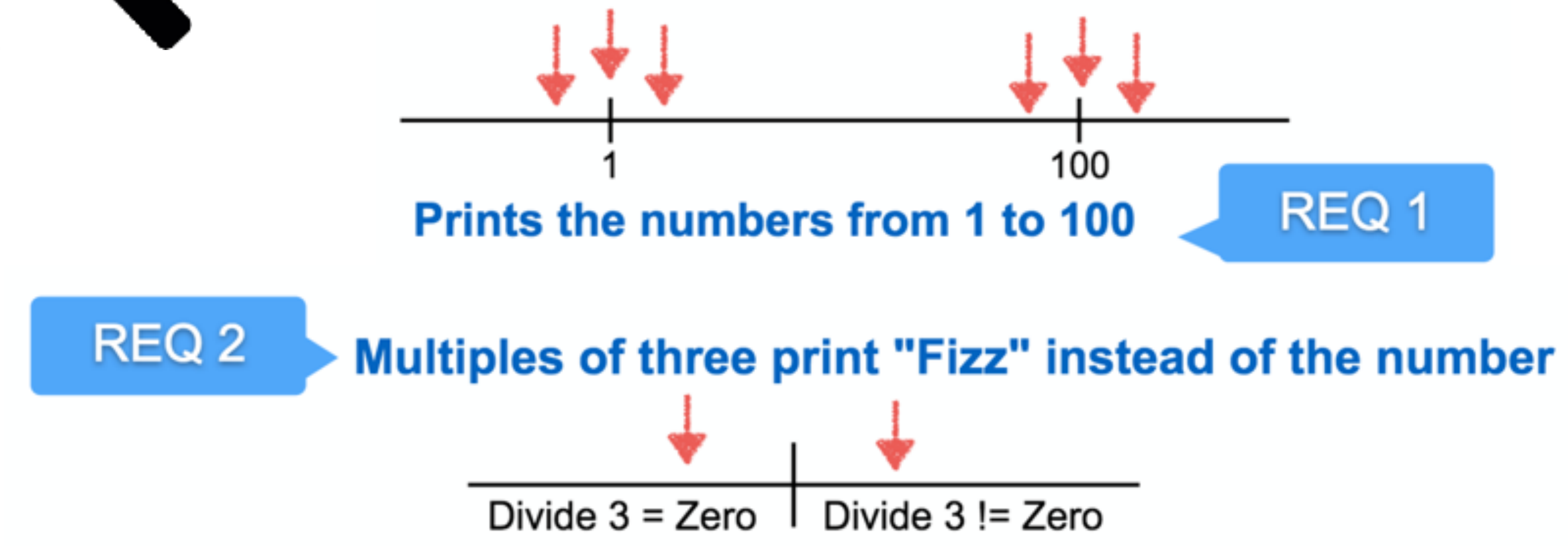
Write a program that prints the numbers 1-100 but for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz”. For numbers which are multiples of both three and five print “FizzBuzz”



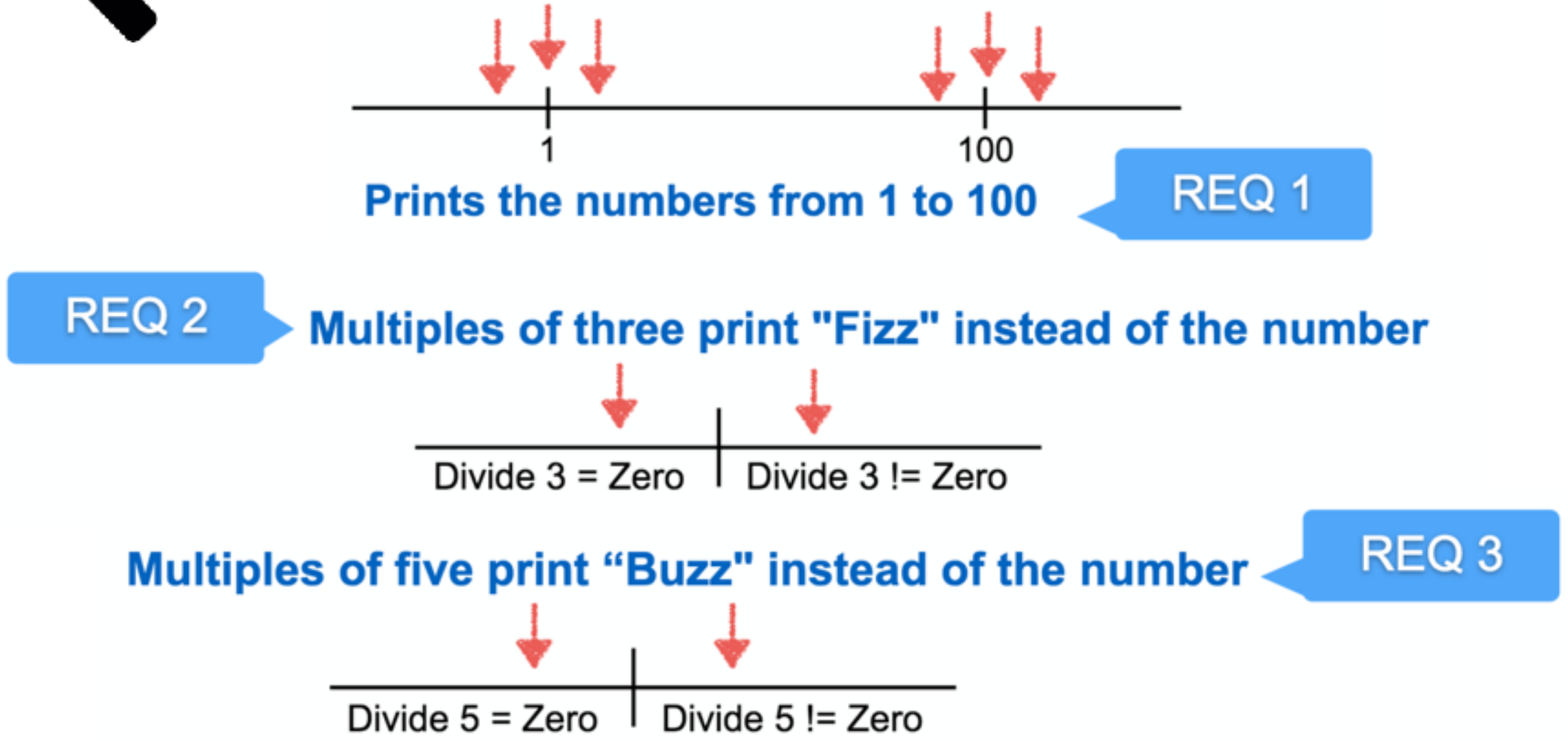
Find Business Conditions



Find Business Conditions


1
100
Prints the numbers from 1 to 100
REQ 1
REQ 2
Multiples of three print "Fizz" instead of the number
Divide 3 = Zero
Divide 3 != Zero

Find Business Conditions

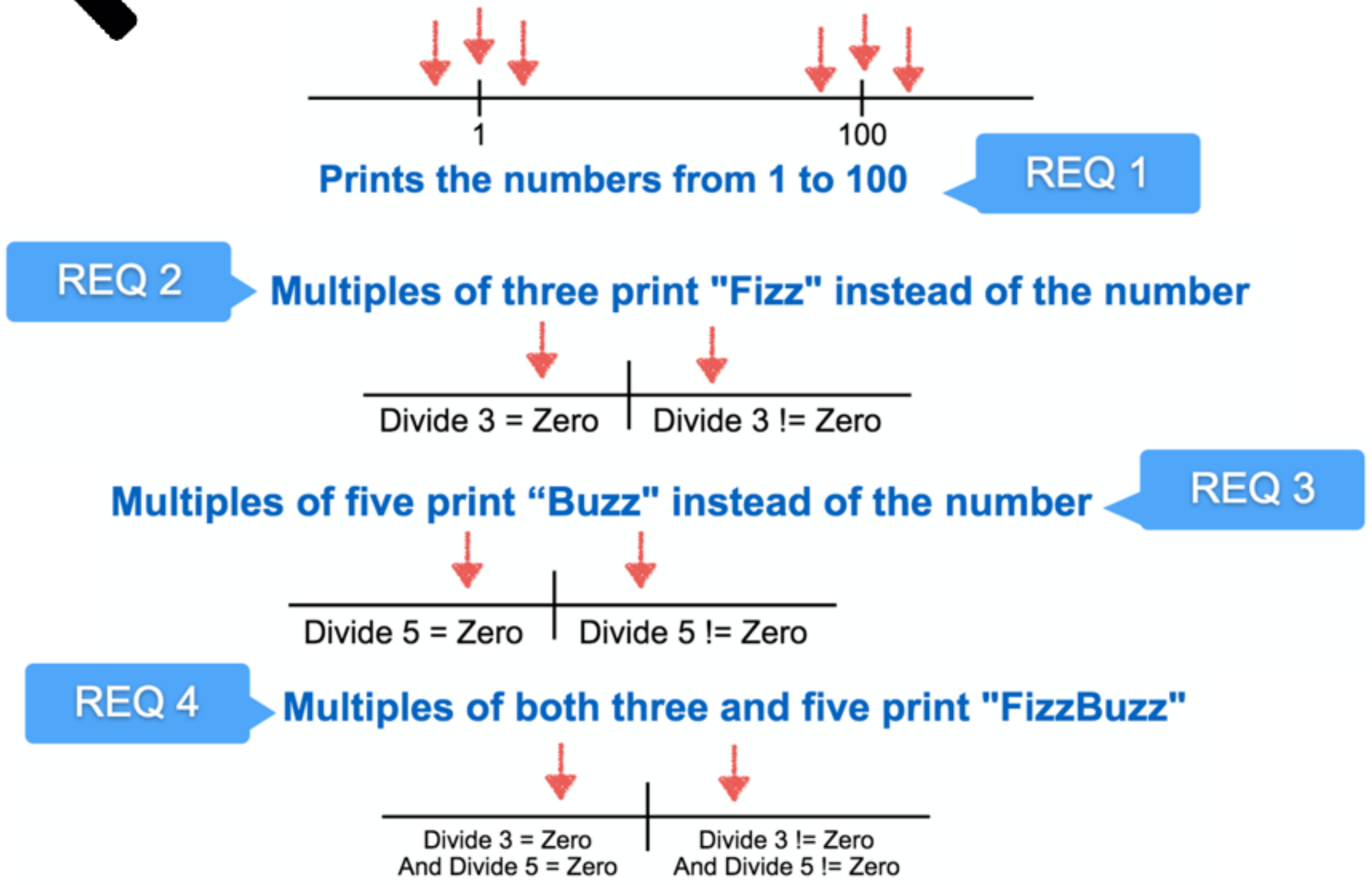


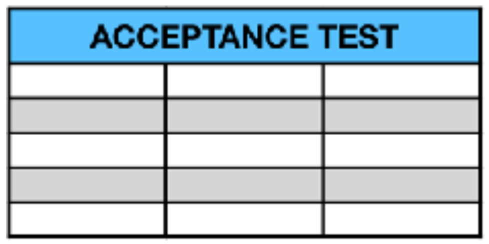
Prints the numbers from 1 to 100

Multiples of three print "Fizz" instead of the number

Multiples of five print “Buzz" instead of the number

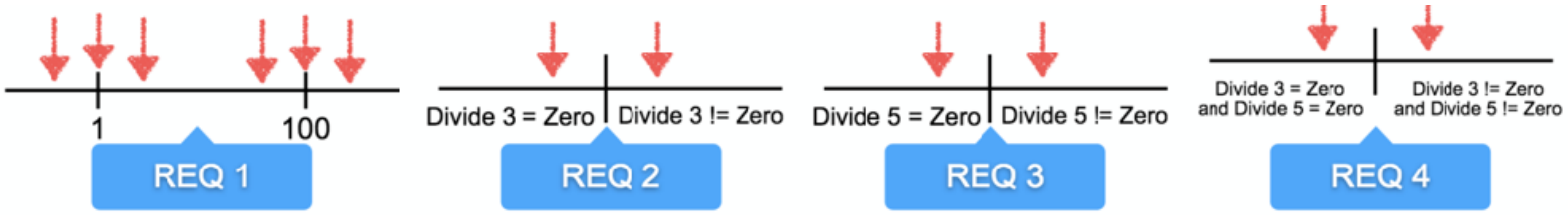
Find Business Conditions


1
100
Prints the numbers from 1 to 100
REQ 1
REQ 2
Multiples of three print "Fizz" instead of the number
Divide 3 = Zero
Divide 3 != Zero
Multiples of five print “Buzz" instead of the number
REQ 3
Divide 5 = Zero
Divide 5 != Zero
REQ 4
Multiples of both three and five print "FizzBuzz"
Divide 3 = Zero And Divide 5 = Zero
Divide 3 != Zero And Divide 5 != Zero



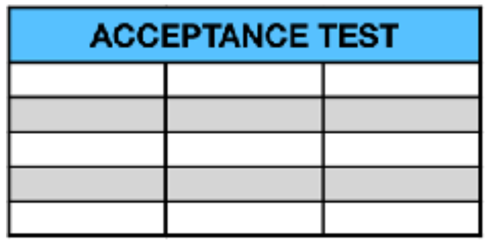
ACCEPTANCE TEST

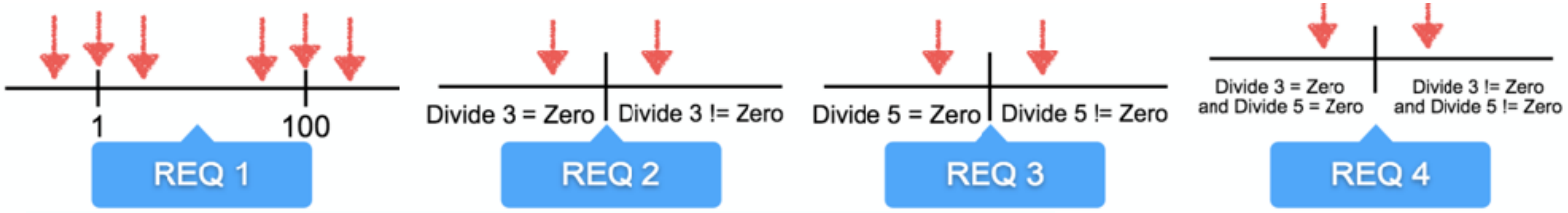
Test Design



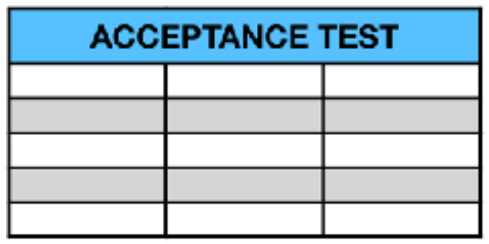
CASE	CONDITIONS	EXPECTED	REQ			
			REQ 1	REQ 2	REQ 3	REQ 4
1	Equal 1	Display Number	/			
2	More than 1 divide 3 = zero	Display Fizz	/	/		
3	More than 1 divide 3 != zero	Display Number	/	/		
4	More than 1 divide 5 = zero	Display Buzz	/		/	
5	More than 1 divide 5 != zero	Display Number	/		/	
6	More than 1 divide 3 and 5 = zero	Display FizzBuzz	/			/
7	More than 1 divide 3 and 5 != zero	Display Number	/			/
8	Less than 100 divide 3 = zero	Display Fizz	/	/		
9	Less than 100 divide 3 != zero	Display Number	/	/		
10	Less than 100 divide 5 = zero	Display Buzz	/		/	
11	Less than 100 divide 5 = zero	Display Number	/		/	
12	Less than 100 divide 3 and 5 = zero	Display FizzBuzz	/			/
13	Less than 100 divide 3 and 5 != zero	Display Number	/			/
14	Equal 100	Display Buzz	/		/	
15	Less then 1	Not Display	/			
16	More then 100	Not Display	/			

Test Design

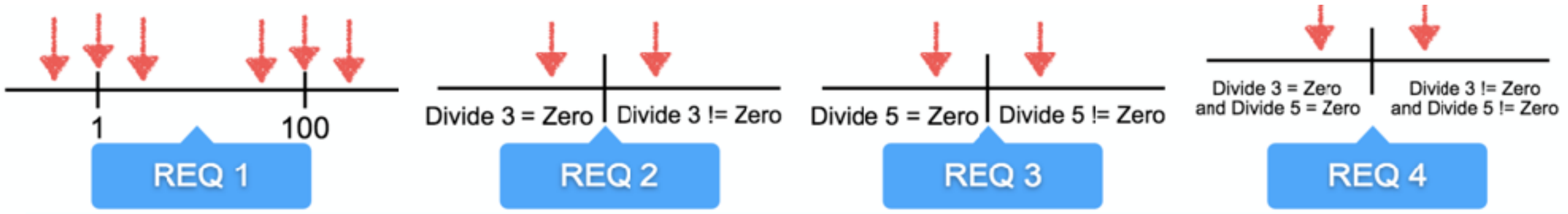




CASE	CONDITIONS	EXPECTED
1	Equal 1	Display Number
2	More than 1 divide 3 = zero	Display Fizz
3	More than 1 divide 3 != zero	Display Number
4	More than 1 divide 5 = zero	Display Buzz
5	More than 1 divide 5 != zero	Display Number
6	More than 1 divide 3 and 5 = zero	Display FizzBuzz
7	More than 1 divide 3 and 5 != zero	Display Number
8	Less than 100 divide 3 = zero	Display Fizz
9	Less than 100 divide 3 != zero	Display Number
10	Less than 100 divide 5 = zero	Display Buzz
11	Less than 100 divide 5 = zero	Display Number
12	Less than 100 divide 3 and 5 = zero	Display FizzBuzz
13	Less than 100 divide 3 and 5 != zero	Display Number
14	Equal 100	Display Buzz
15	Less then 1	Not Display
16	More then 100	Not Display



Test Design



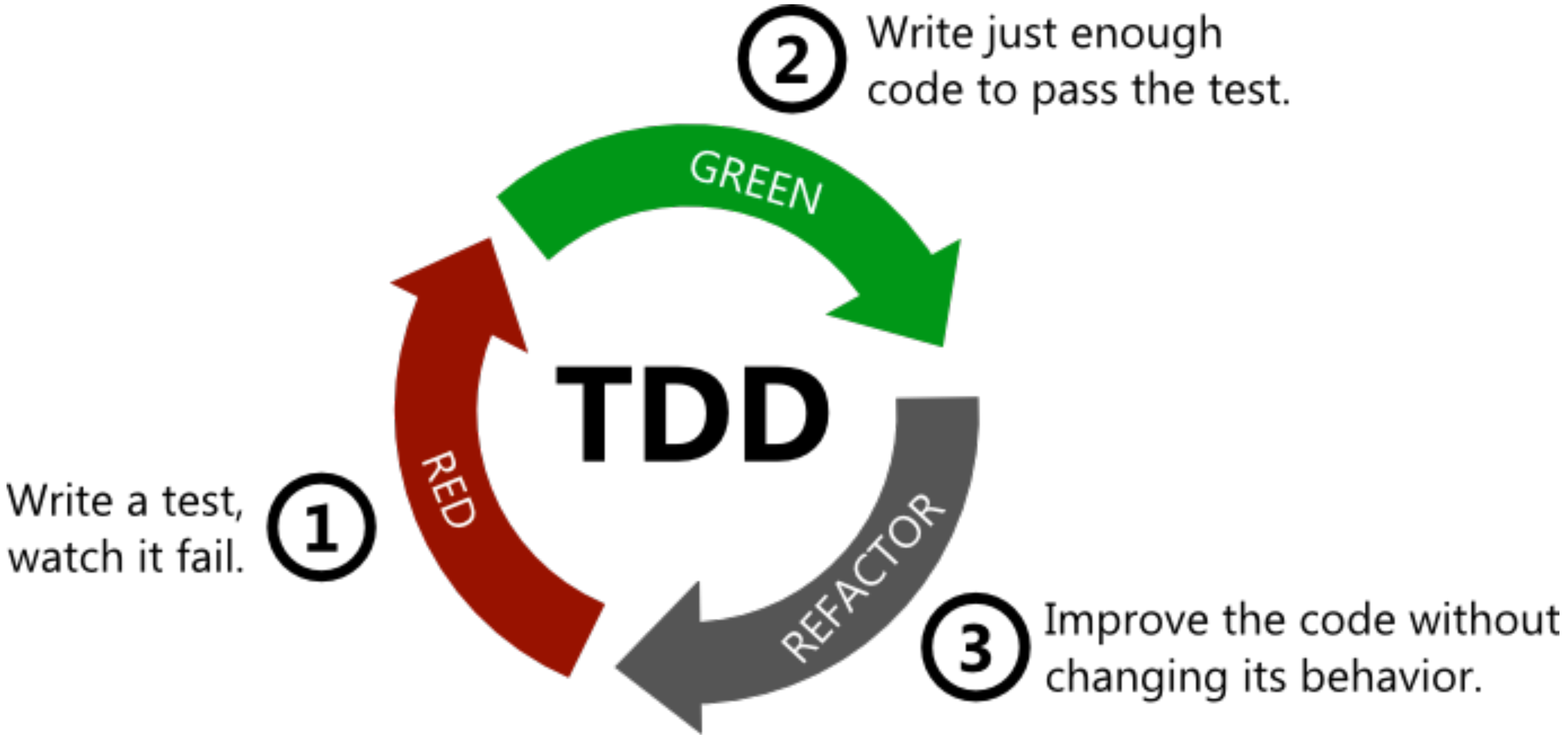
CASE	CONDITIONS	EXPECTED	DATA	EXPECTED
1	Equal 1	Display Number	1	1
2	More than 1 divide 3 = zero	Display Fizz	3	Fizz
3	More than 1 divide 3 != zero	Display Number	4	4
4	More than 1 divide 5 = zero	Display Buzz	5	Buzz
5	More than 1 divide 5 != zero	Display Number	7	7
6	More than 1 divide 3 and 5 = zero	Display FizzBuzz	15	FizzBuzz
7	More than 1 divide 3 and 5 != zero	Display Number	17	17
8	Less than 100 divide 3 = zero	Display Fizz	99	Fizz
9	Less than 100 divide 3 != zero	Display Number	98	98
10	Less than 100 divide 5 = zero	Display Buzz	95	Buzz
11	Less than 100 divide 5 = zero	Display Number	94	94
12	Less than 100 divide 3 and 5 = zero	Display FizzBuzz	90	FizzBuzz
13	Less than 100 divide 3 and 5 != zero	Display Number	98	98
14	Equal 100	Display Buzz	100	Buzz
15	Less then 1	Not Display	0	Not Display
16	More then 100	Not Display	101	Not Display

ACCEPTANCE TEST

Test Design

CASE	CONDITIONS	DATA	EXPECTED
1	Equal 1	1	1
2	More than 1 divide 3 = zero	3	Fizz
3	More than 1 divide 3 != zero	4	4
4	More than 1 divide 5 = zero	5	Buzz
5	More than 1 divide 5 != zero	7	7
6	More than 1 divide 3 and 5 = zero	15	FizzBuzz
7	More than 1 divide 3 and 5 != zero	17	17
8	Less than 100 divide 3 = zero	99	Fizz
9	Less than 100 divide 3 != zero	98	98
10	Less than 100 divide 5 = zero	95	Buzz
11	Less than 100 divide 5 = zero	94	94
12	Less than 100 divide 3 and 5 = zero	90	FizzBuzz
13	Less than 100 divide 3 and 5 != zero	98	98
14	Equal 100	100	Buzz
15	Less then 1	0	Not Display
16	More then 100	101	Not Display

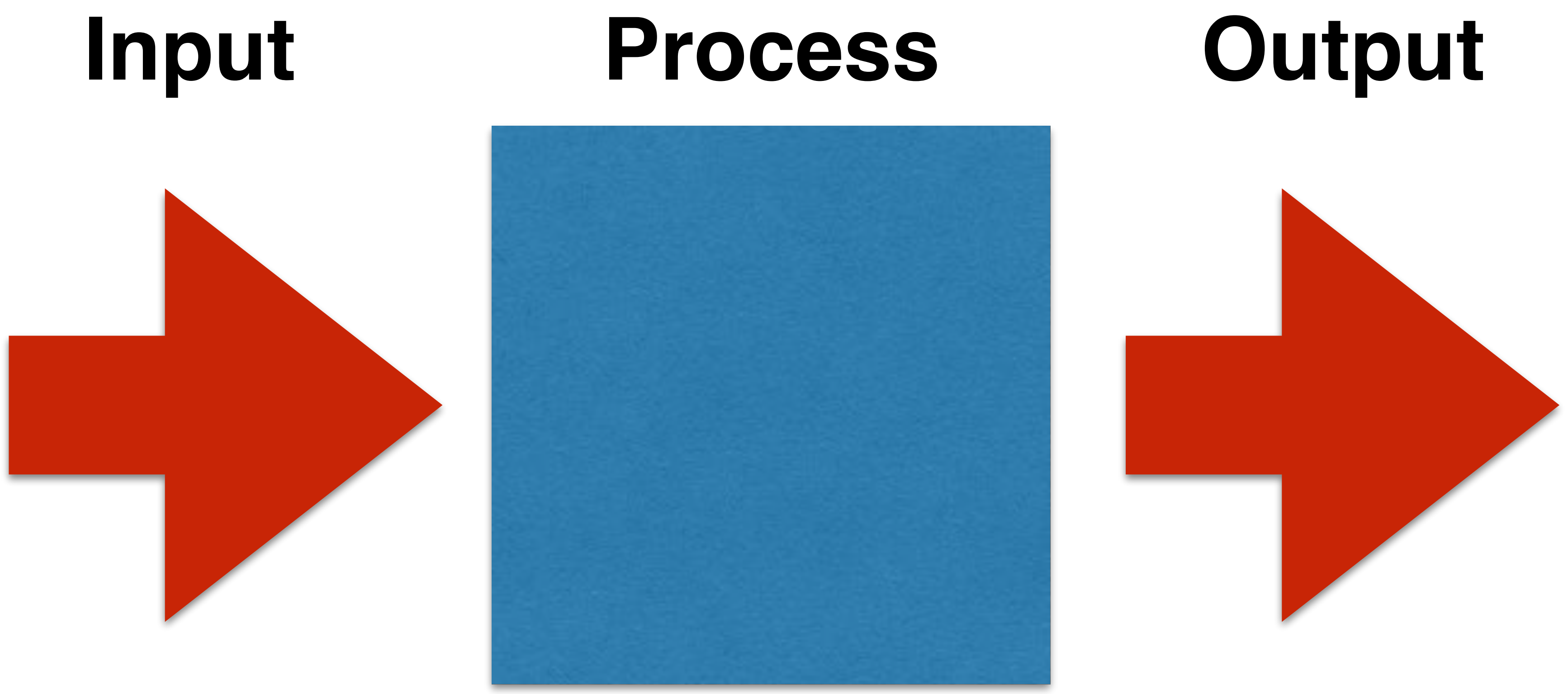
Test-Driven Development


Write just enough code to pass the test.
2
GREEN
TDD
1
Write a test, watch it fail.
RED
REFACTOR
3
Improve the code without changing its behavior.

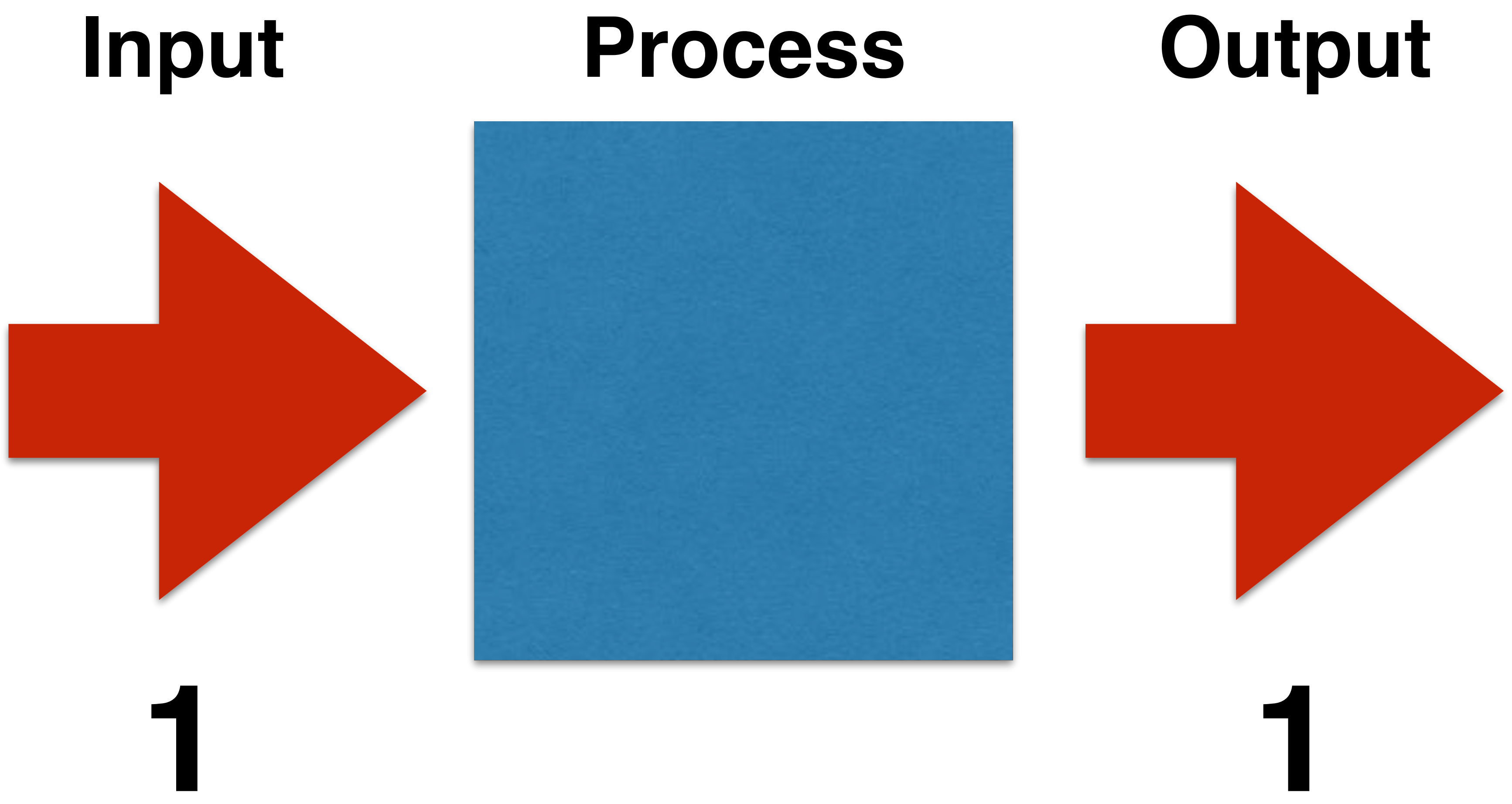
Write Test, and watch it fail

CASE	CONDITIONS	DATA	EXPECTED
1	Equal 1	1	1
2	More than 1 divide 3 = zero	3	Fizz
3	More than 1 divide 3 != zero	4	4
4	More than 1 divide 5 = zero	5	Buzz
5	More than 1 divide 5 != zero	7	7
6	More than 1 divide 3 and 5 = zero	15	FizzBuzz
7	More than 1 divide 3 and 5 != zero	17	17
8	Less than 100 divide 3 = zero	99	Fizz
9	Less than 100 divide 3 != zero	98	98
10	Less than 100 divide 5 = zero	95	Buzz
11	Less than 100 divide 5 = zero	94	94
12	Less than 100 divide 3 and 5 = zero	90	FizzBuzz
13	Less than 100 divide 3 and 5 != zero	98	98
14	Equal 100	100	Buzz
15	Less then 1	0	Not Display
16	More then 100	101	Not Display

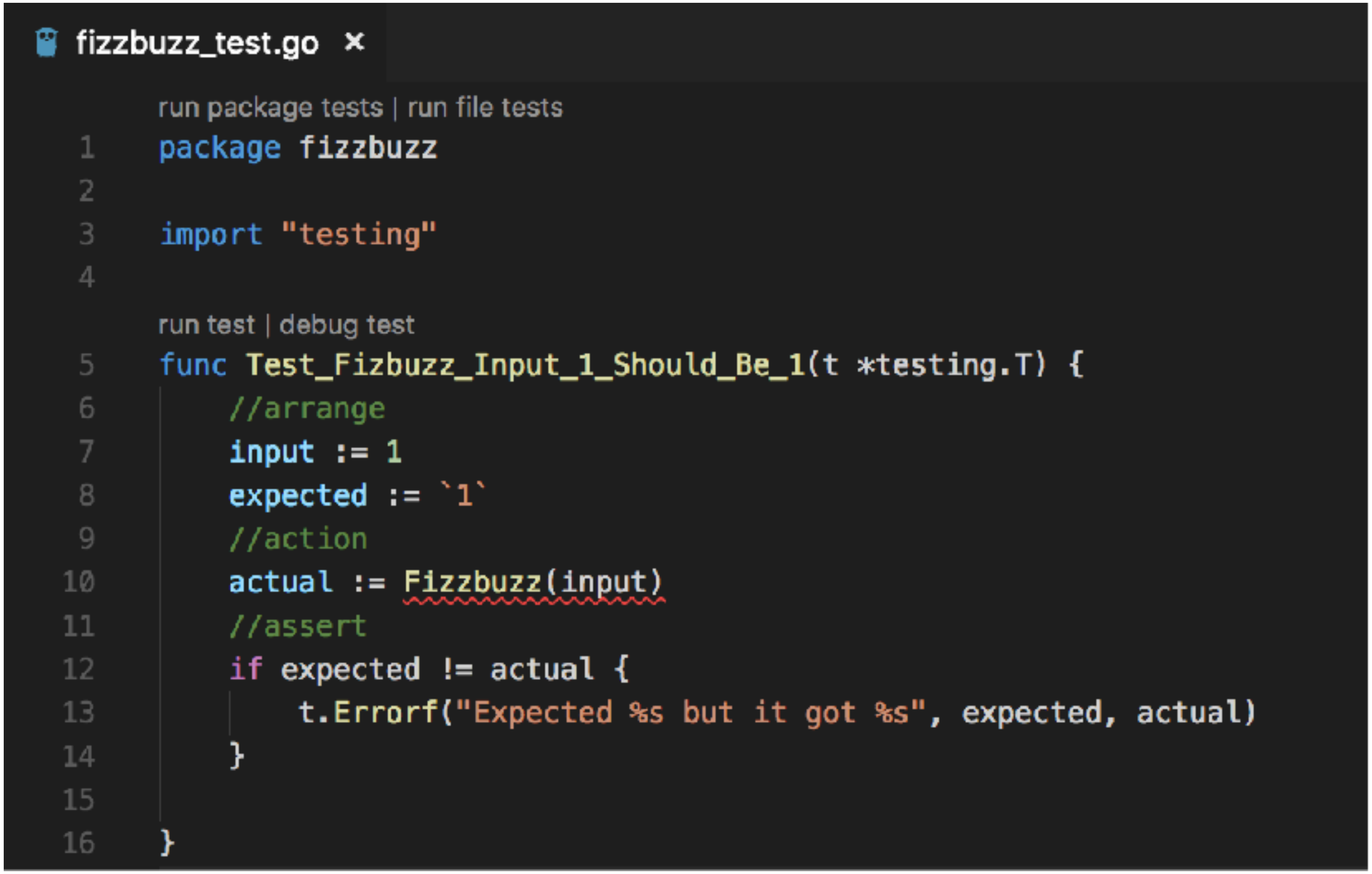
Write Test, and watch it fail



Start with case 1
input1 should be 1

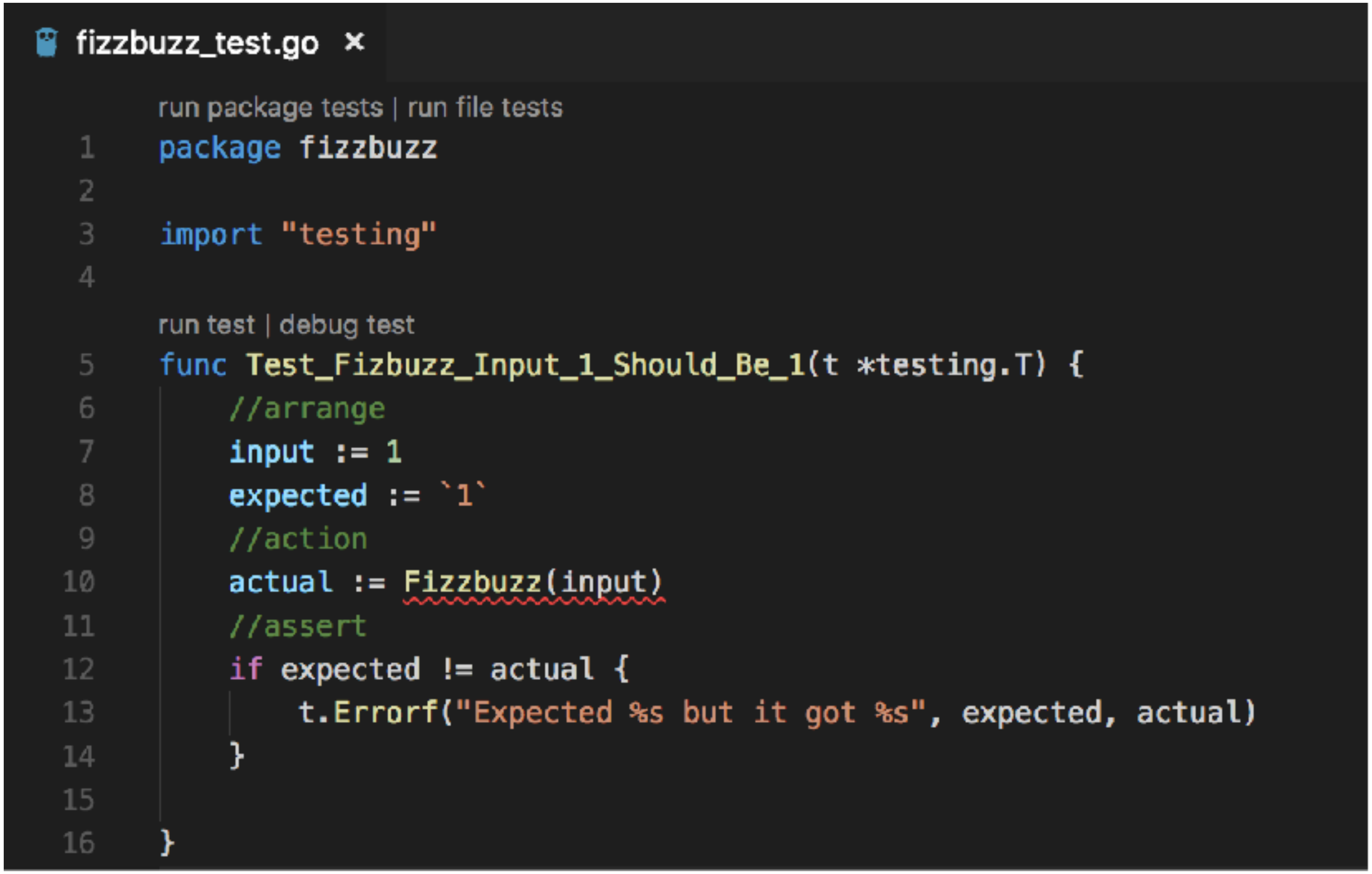


Start with case 1
input1 should be 1



```go
package fizzbuzz

import "testing"

func Test_Fizbuzz_Input_1_Should_Be_1(t *testing.T) {
	//arrange
	input := 1
	expected := `1`
	//action
	actual := Fizzbuzz(input)
	//assert
	if expected != actual {
		t.Errorf("Expected %s but it got %s", expected, actual)
	}

}
```

Start with case 1
input1 should be 1



fizzbuzz_test.go

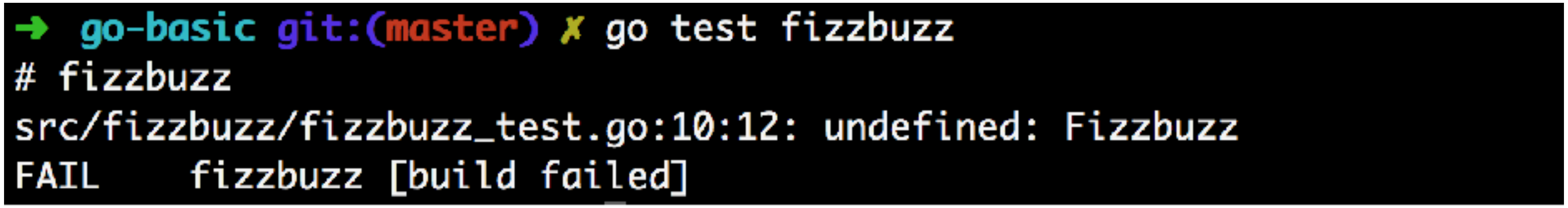
run package tests | run file tests

```go
package fizzbuzz

import "testing"
```

run test | debug test

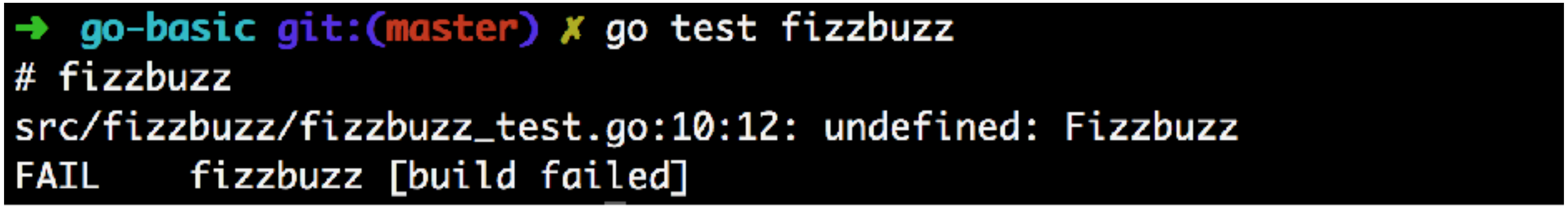
```go
func Test_Fizbuzz_Input_1_Should_Be_1(t *testing.T) {
	//arrange
	input := 1
	expected := `1`
	//action
	actual := Fizzbuzz(input)
	//assert
	if expected != actual {
		t.Errorf("Expected %s but it got %s", expected, actual)
	}

}
```

Start with case 1
input1 should be 1



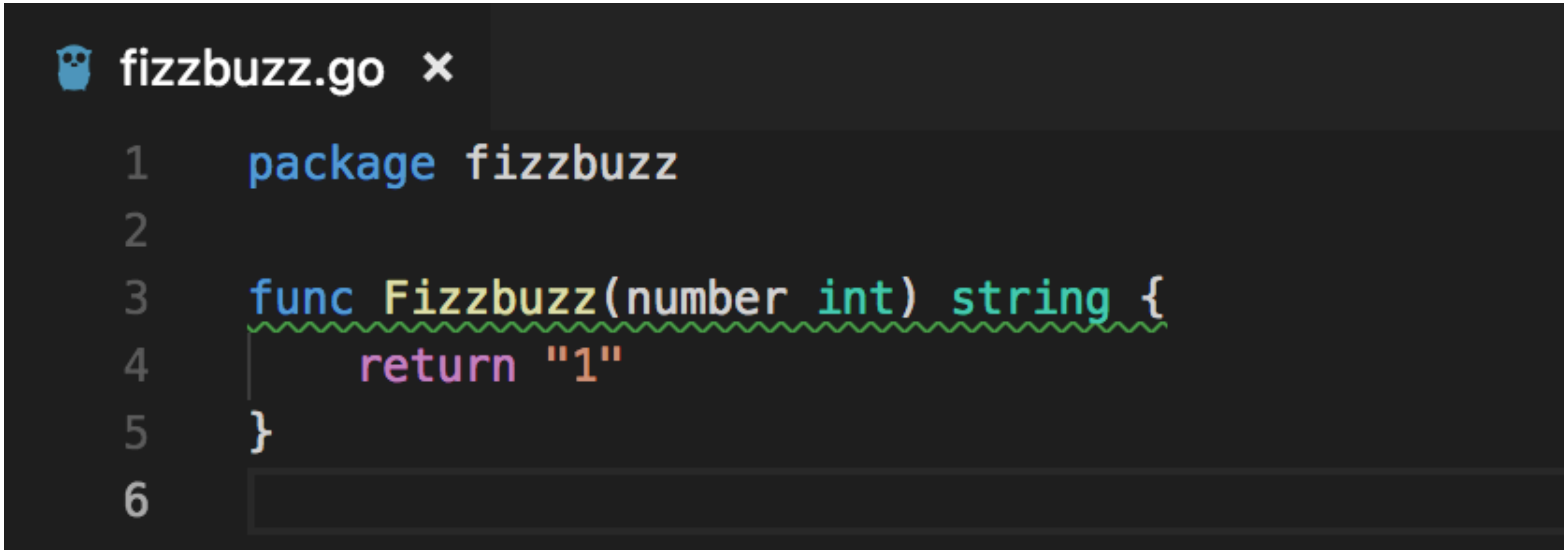
```
➜  go-basic git:(master) ✗ go test fizzbuzz
# fizzbuzz
src/fizzbuzz/fizzbuzz_test.go:10:12: undefined: Fizzbuzz
FAIL    fizzbuzz [build failed]
```

Start with case 1
input1 should be 1



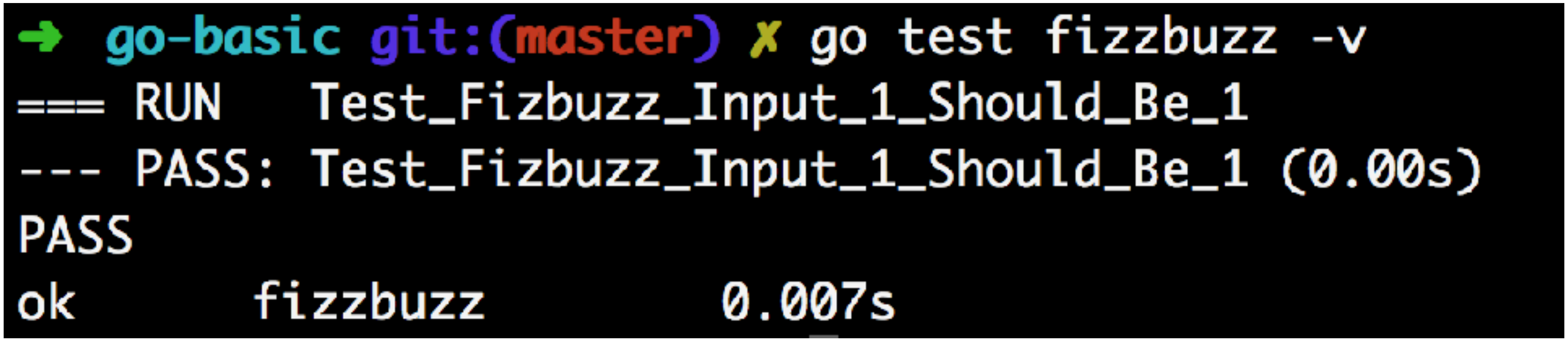
```
➜  go-basic git:(master) ✗ go test fizzbuzz
# fizzbuzz
src/fizzbuzz/fizzbuzz_test.go:10:12: undefined: Fizzbuzz
FAIL	fizzbuzz [build failed]
```

Start with case 1
input1 should be 1



```go
package fizzbuzz

func Fizzbuzz(number int) string {
	return "1"
}
```

Start with case 1
input1 should be 1

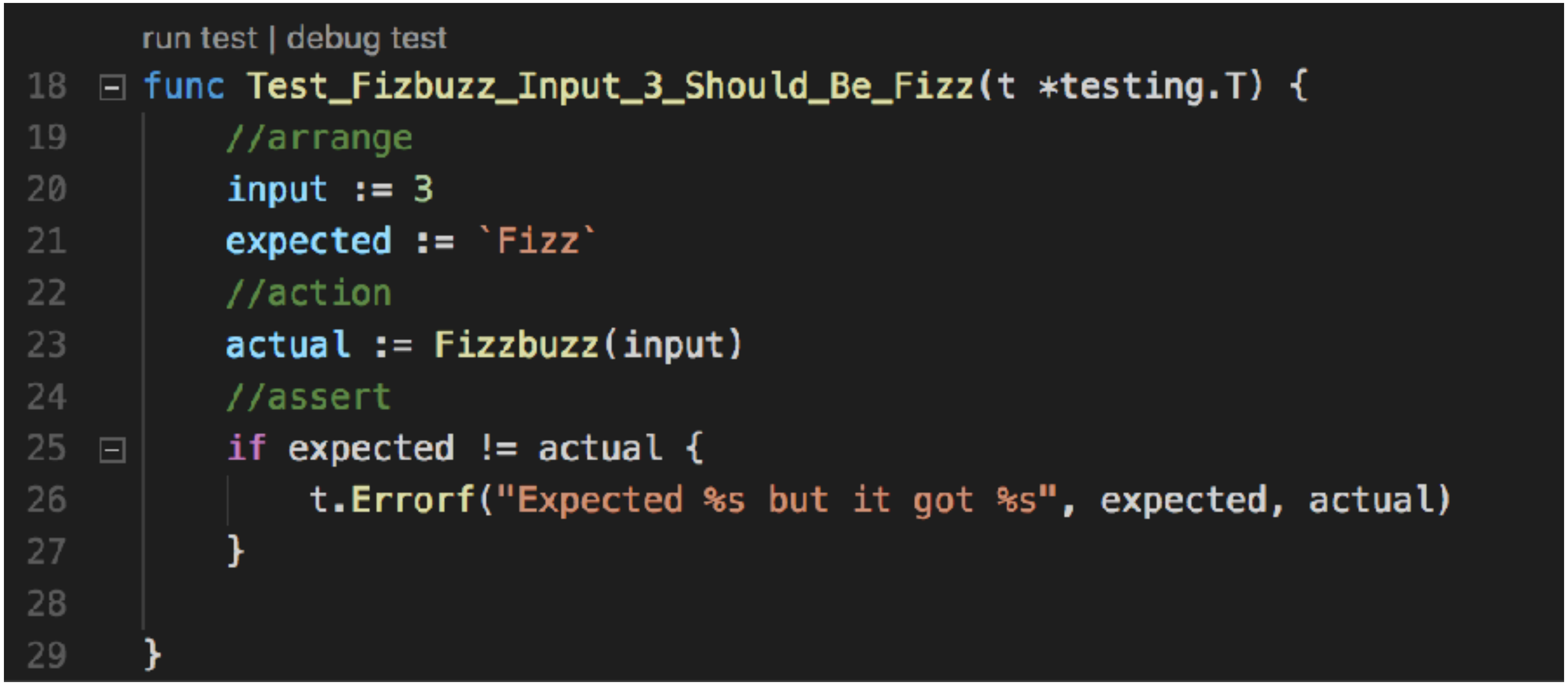


```
➜  go-basic git:(master) ✗ go test fizzbuzz -v
=== RUN   Test_Fizbuzz_Input_1_Should_Be_1
--- PASS: Test_Fizbuzz_Input_1_Should_Be_1 (0.00s)
PASS
ok      fizzbuzz        0.007s
```

case 2
input 3 should be Fizz

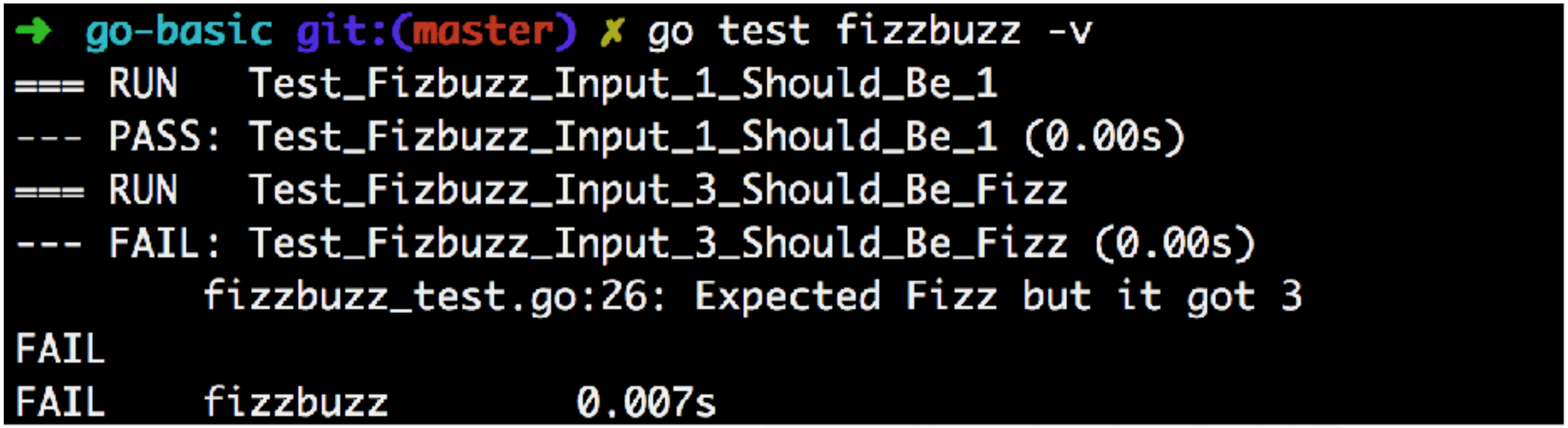
CASE	CONDITIONS	DATA	EXPECTED
1	Equal 1	1	1
2	More than 1 divide 3 = zero	3	Fizz
3	More than 1 divide 3 != zero	4	4
4	More than 1 divide 5 = zero	5	Buzz
5	More than 1 divide 5 != zero	7	7
6	More than 1 divide 3 and 5 = zero	15	FizzBuzz
7	More than 1 divide 3 and 5 != zero	17	17
8	Less than 100 divide 3 = zero	99	Fizz
9	Less than 100 divide 3 != zero	98	98
10	Less than 100 divide 5 = zero	95	Buzz
11	Less than 100 divide 5 = zero	94	94
12	Less than 100 divide 3 and 5 = zero	90	FizzBuzz
13	Less than 100 divide 3 and 5 != zero	98	98
14	Equal 100	100	Buzz
15	Less then 1	0	Not Display
16	More then 100	101	Not Display

case 2
input 3 should be Fizz



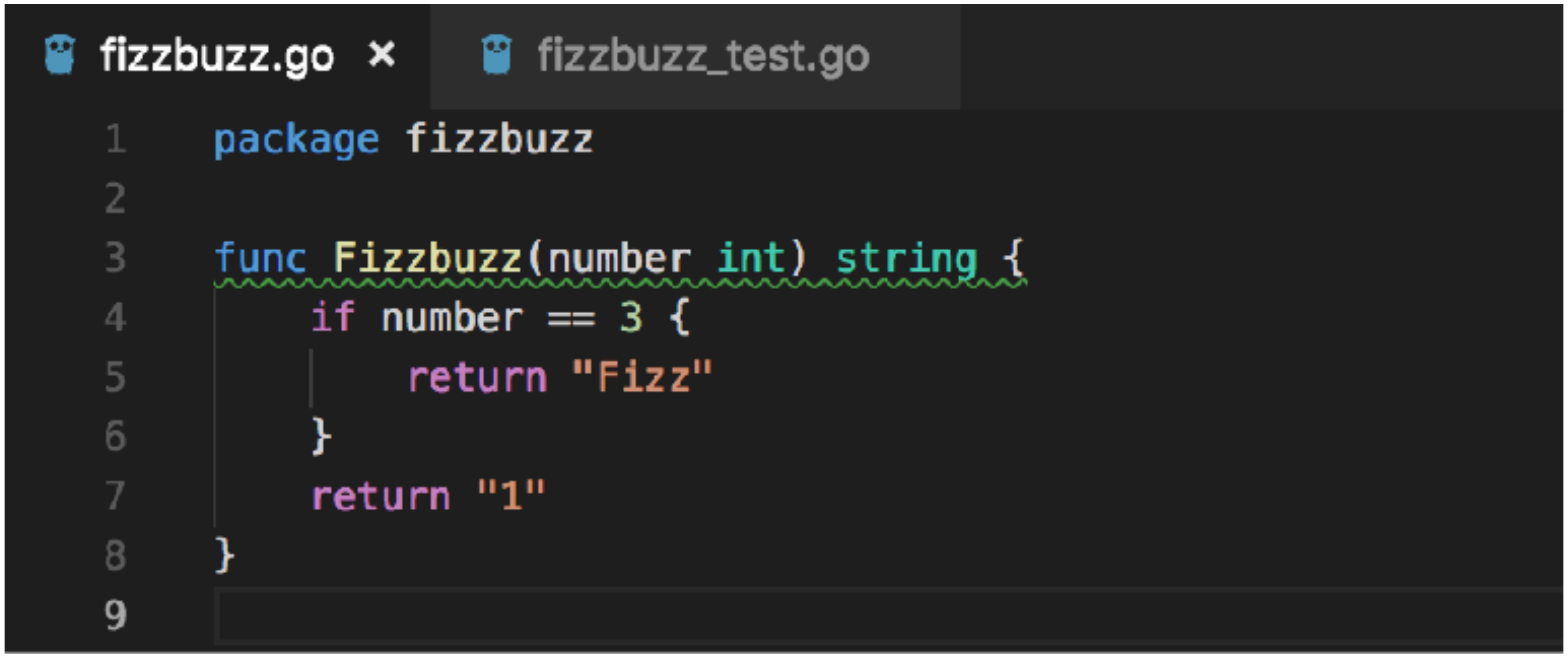
```go
func Test_Fizbuzz_Input_3_Should_Be_Fizz(t *testing.T) {
	//arrange
	input := 3
	expected := `Fizz`
	//action
	actual := Fizzbuzz(input)
	//assert
	if expected != actual {
		t.Errorf("Expected %s but it got %s", expected, actual)
	}

}
```

case 2
input 3 should be Fizz



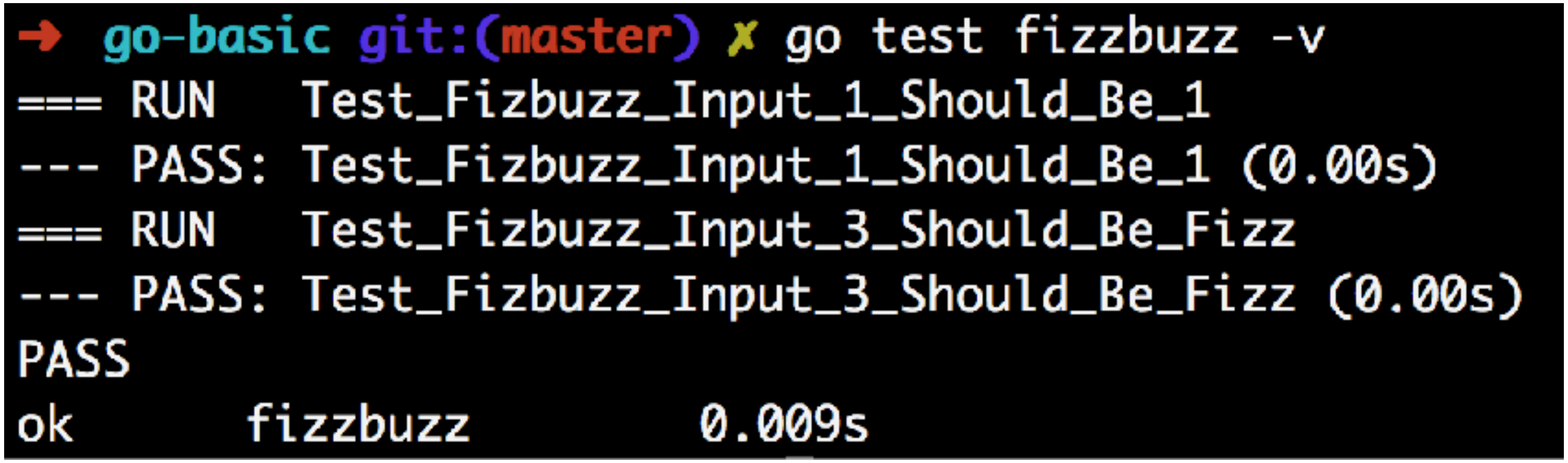
```
➜  go-basic git:(master) ✗ go test fizzbuzz -v
=== RUN   Test_Fizbuzz_Input_1_Should_Be_1
--- PASS: Test_Fizbuzz_Input_1_Should_Be_1 (0.00s)
=== RUN   Test_Fizbuzz_Input_3_Should_Be_Fizz
--- FAIL: Test_Fizbuzz_Input_3_Should_Be_Fizz (0.00s)
        fizzbuzz_test.go:26: Expected Fizz but it got 3
FAIL
FAIL    fizzbuzz        0.007s
```

case 2
input 3 should be Fizz



```go
package fizzbuzz

func Fizzbuzz(number int) string {
	if number == 3 {
		return "Fizz"
	}
	return "1"
}

```

case 2
input 3 should be Fizz

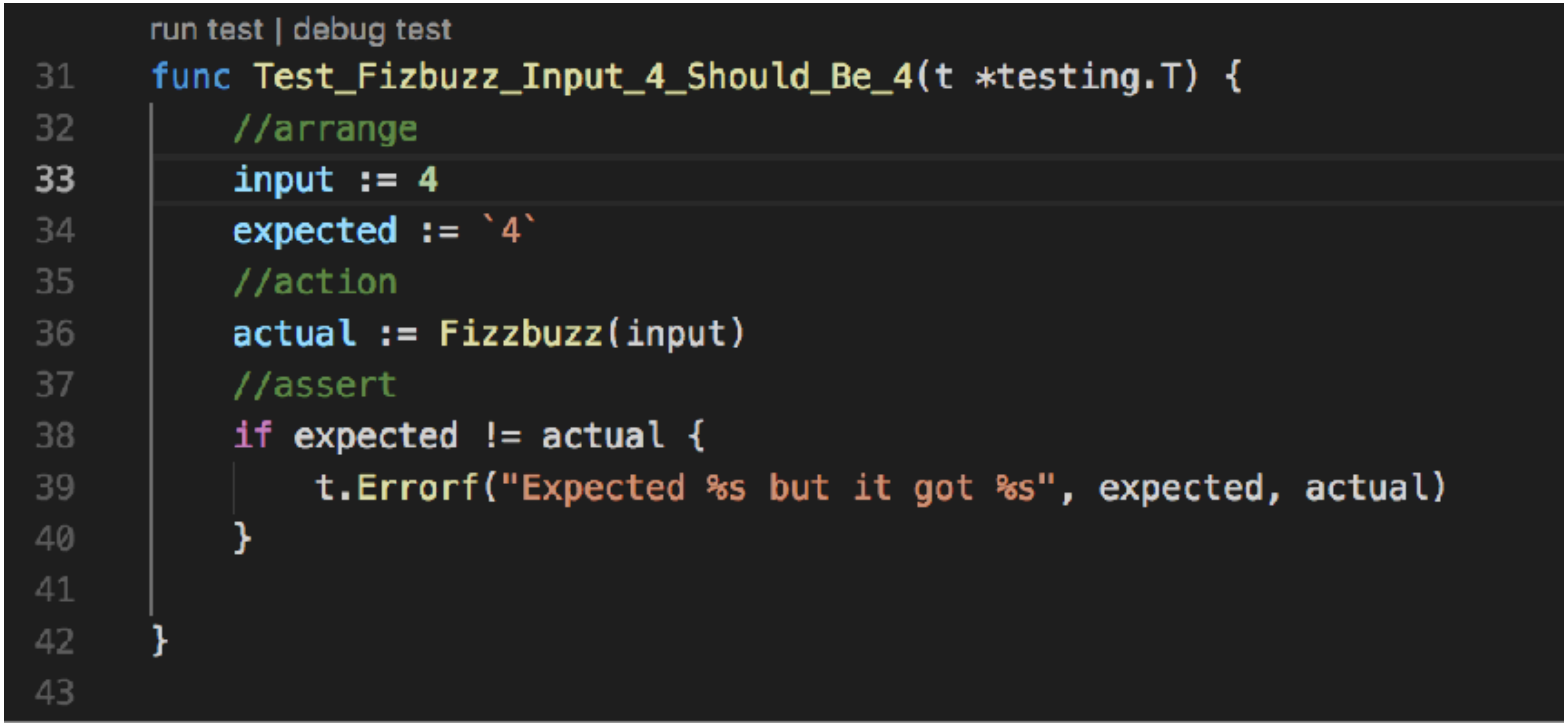


```
➜  go-basic git:(master) ✗ go test fizzbuzz -v
=== RUN   Test_Fizbuzz_Input_1_Should_Be_1
--- PASS: Test_Fizbuzz_Input_1_Should_Be_1 (0.00s)
=== RUN   Test_Fizbuzz_Input_3_Should_Be_Fizz
--- PASS: Test_Fizbuzz_Input_3_Should_Be_Fizz (0.00s)
PASS
ok      fizzbuzz        0.009s
```

case 2
input 3 should be Fizz

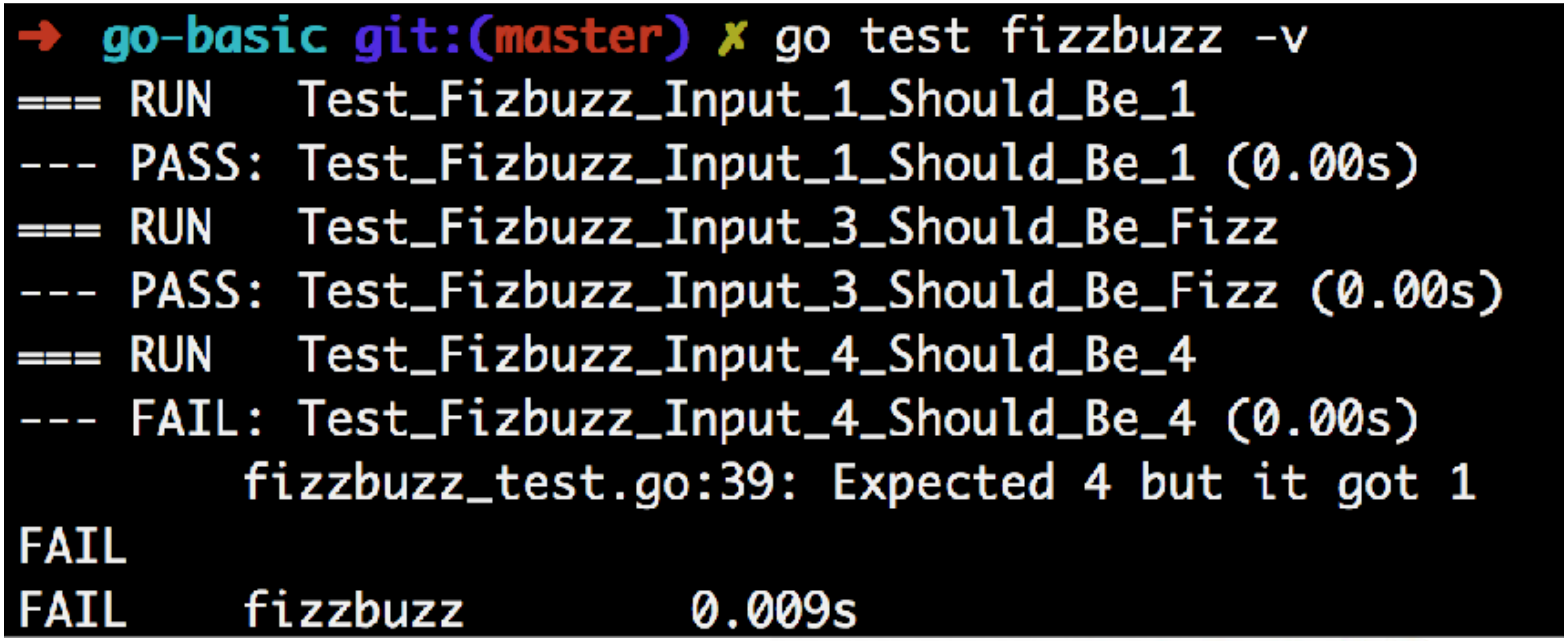
CASE	CONDITIONS	DATA	EXPECTED
1	Equal 1	1	1
2	More than 1 divide 3 = zero	3	Fizz
3	More than 1 divide 3 != zero	4	4
4	More than 1 divide 5 = zero	5	Buzz
5	More than 1 divide 5 != zero	7	7
6	More than 1 divide 3 and 5 = zero	15	FizzBuzz
7	More than 1 divide 3 and 5 != zero	17	17
8	Less than 100 divide 3 = zero	99	Fizz
9	Less than 100 divide 3 != zero	98	98
10	Less than 100 divide 5 = zero	95	Buzz
11	Less than 100 divide 5 = zero	94	94
12	Less than 100 divide 3 and 5 = zero	90	FizzBuzz
13	Less than 100 divide 3 and 5 != zero	98	98
14	Equal 100	100	Buzz
15	Less then 1	0	Not Display
16	More then 100	101	Not Display

case 3
input 4 should be 4



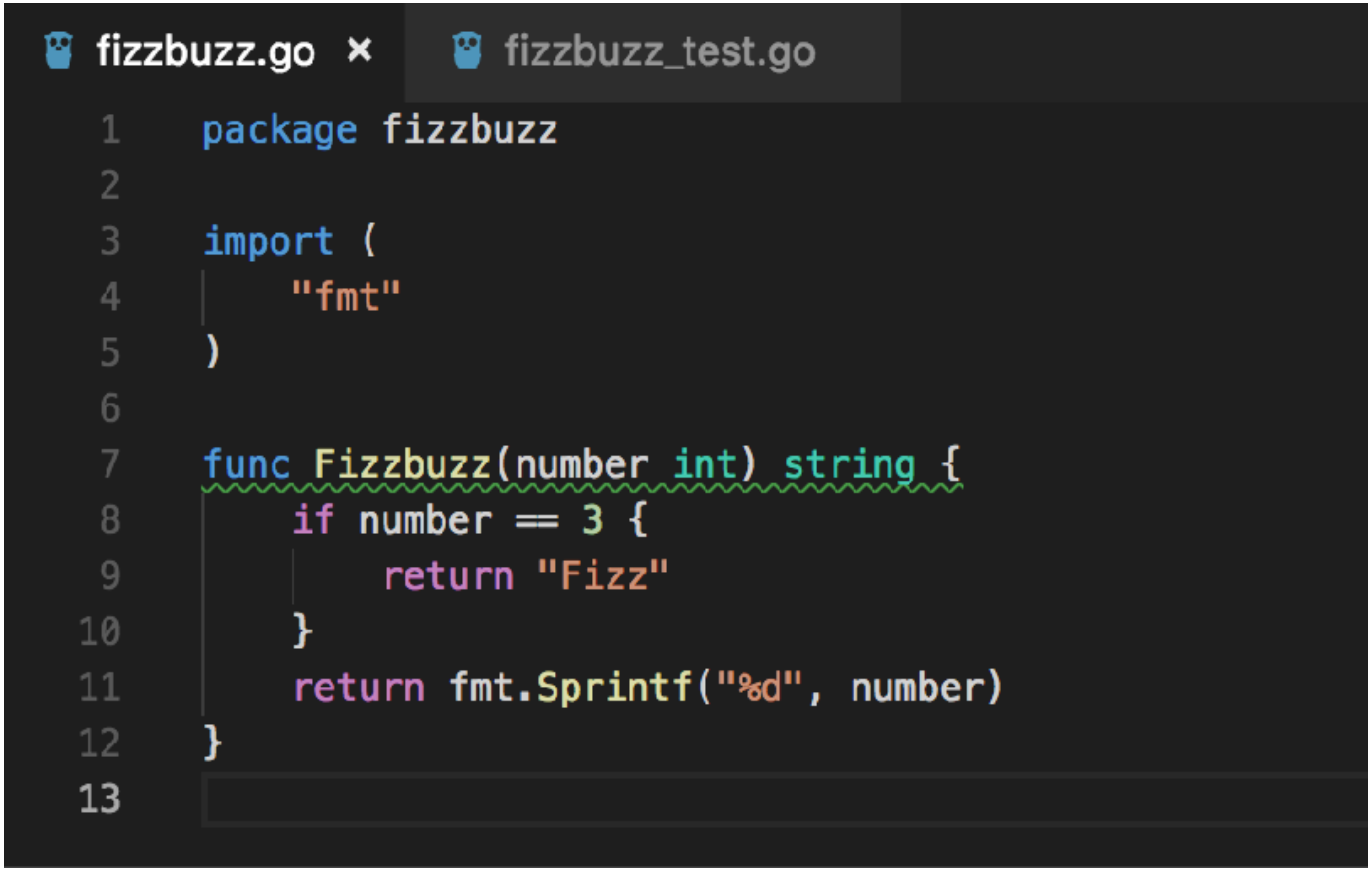
```go
func Test_Fizbuzz_Input_4_Should_Be_4(t *testing.T) {
	//arrange
	input := 4
	expected := `4`
	//action
	actual := Fizzbuzz(input)
	//assert
	if expected != actual {
		t.Errorf("Expected %s but it got %s", expected, actual)
	}

}
```

case 3
input 4 should be 4



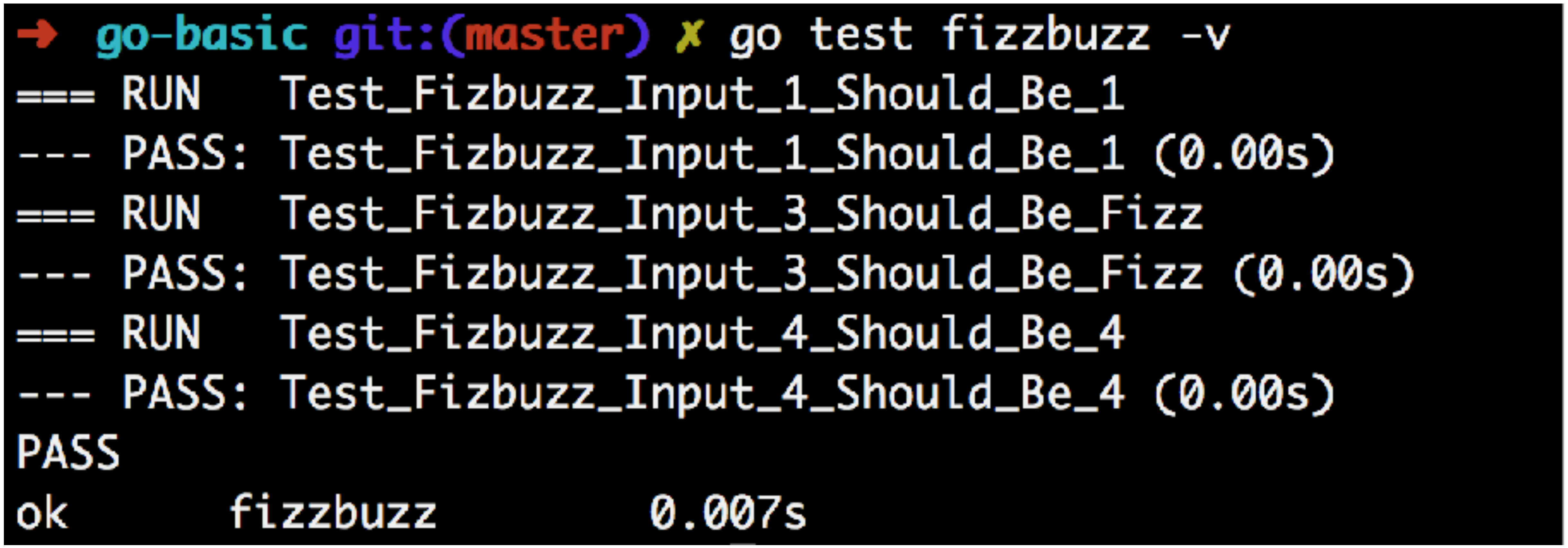
```
➜  go-basic git:(master) ✗ go test fizzbuzz -v
=== RUN   Test_Fizbuzz_Input_1_Should_Be_1
--- PASS: Test_Fizbuzz_Input_1_Should_Be_1 (0.00s)
=== RUN   Test_Fizbuzz_Input_3_Should_Be_Fizz
--- PASS: Test_Fizbuzz_Input_3_Should_Be_Fizz (0.00s)
=== RUN   Test_Fizbuzz_Input_4_Should_Be_4
--- FAIL: Test_Fizbuzz_Input_4_Should_Be_4 (0.00s)
        fizzbuzz_test.go:39: Expected 4 but it got 1
FAIL
FAIL    fizzbuzz        0.009s
```

case 3
input 4 should be 4



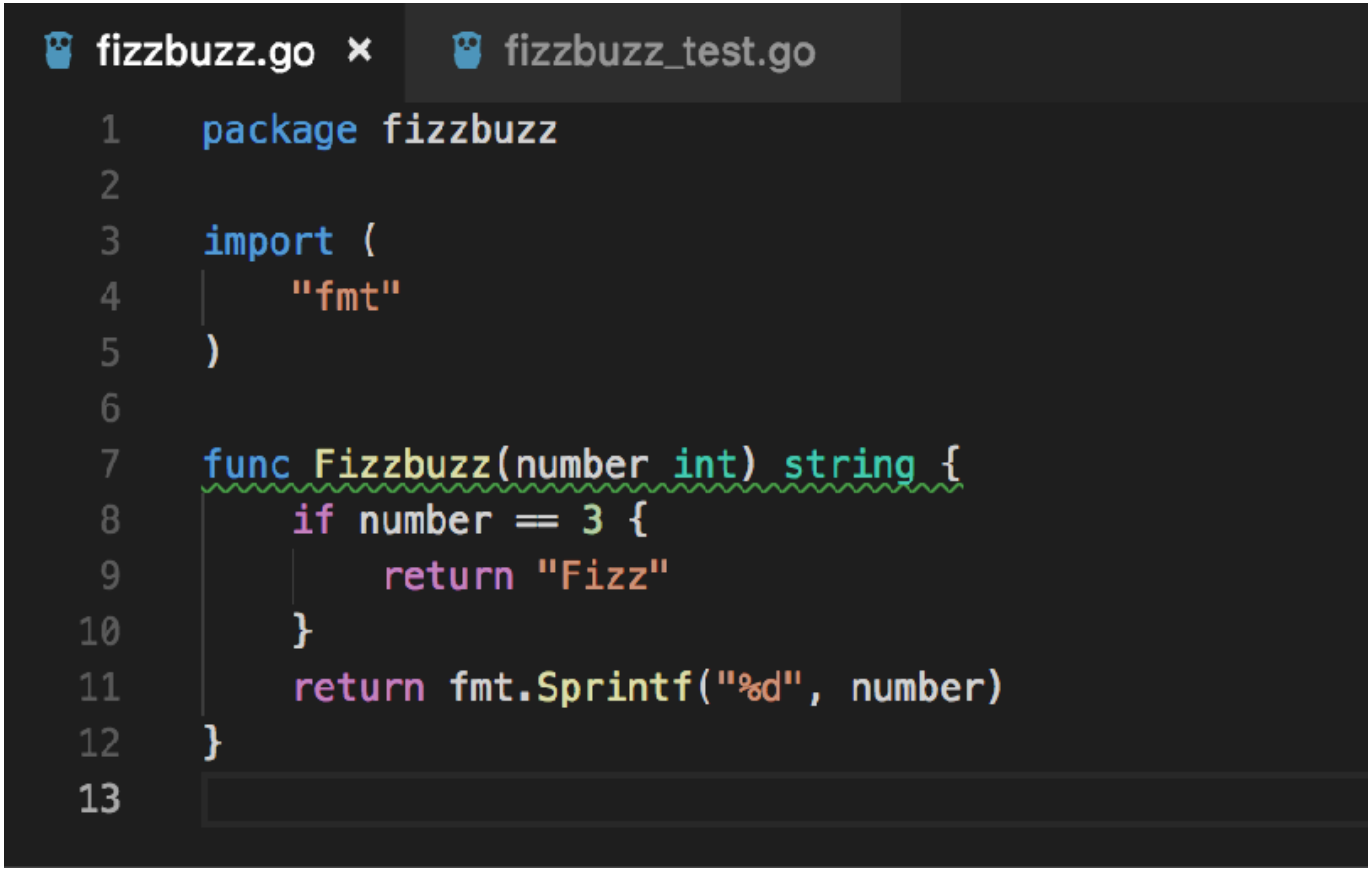
```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	if number == 3 {
		return "Fizz"
	}
	return fmt.Sprintf("%d", number)
}

```

case 3
input 4 should be 4



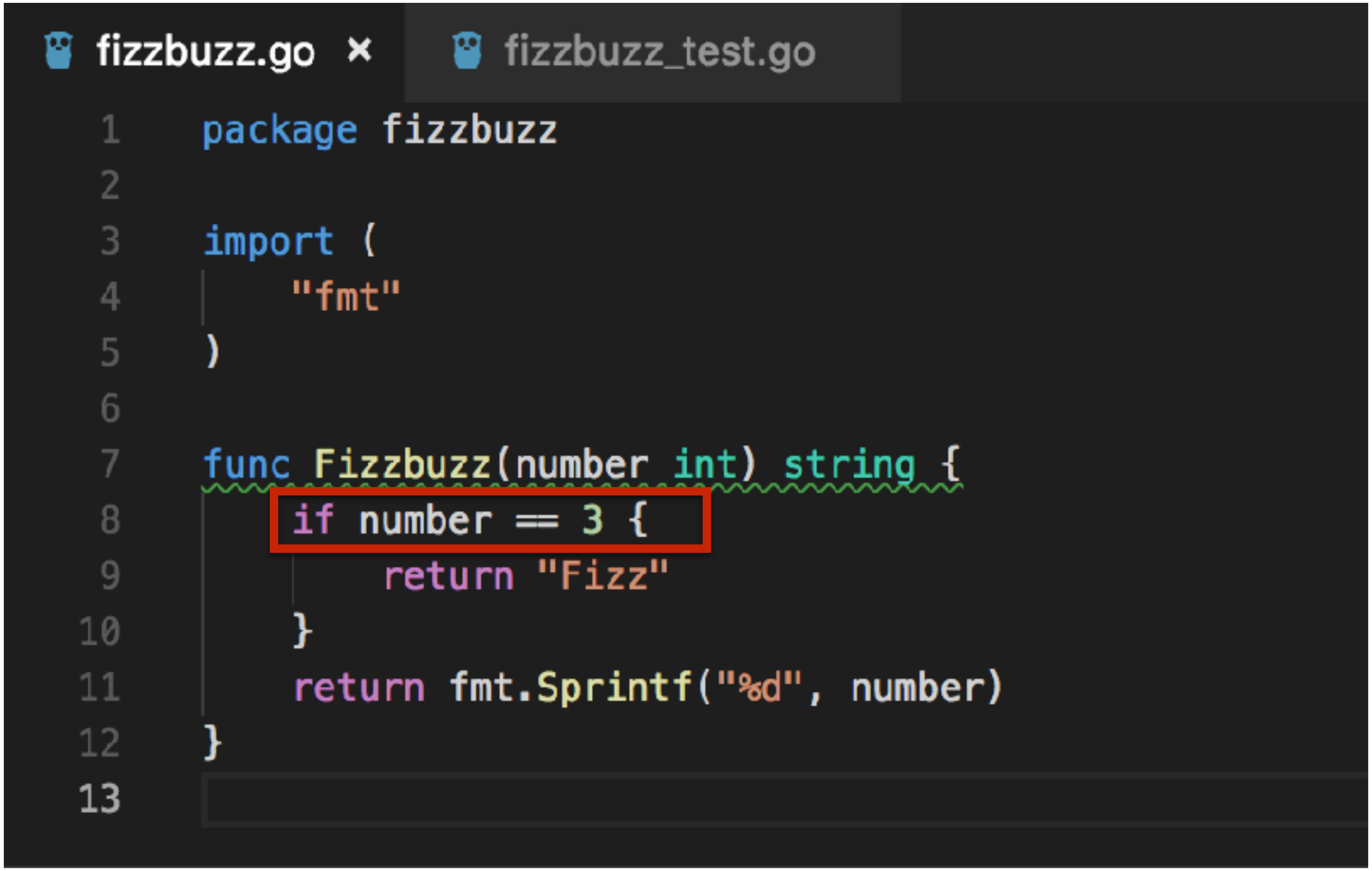
```
➜  go-basic git:(master) ✗ go test fizzbuzz -v
=== RUN   Test_Fizbuzz_Input_1_Should_Be_1
--- PASS: Test_Fizbuzz_Input_1_Should_Be_1 (0.00s)
=== RUN   Test_Fizbuzz_Input_3_Should_Be_Fizz
--- PASS: Test_Fizbuzz_Input_3_Should_Be_Fizz (0.00s)
=== RUN   Test_Fizbuzz_Input_4_Should_Be_4
--- PASS: Test_Fizbuzz_Input_4_Should_Be_4 (0.00s)
PASS
ok      fizzbuzz        0.007s
```

case 3
input 4 should be 4



```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	if number == 3 {
		return "Fizz"
	}
	return fmt.Sprintf("%d", number)
}

```

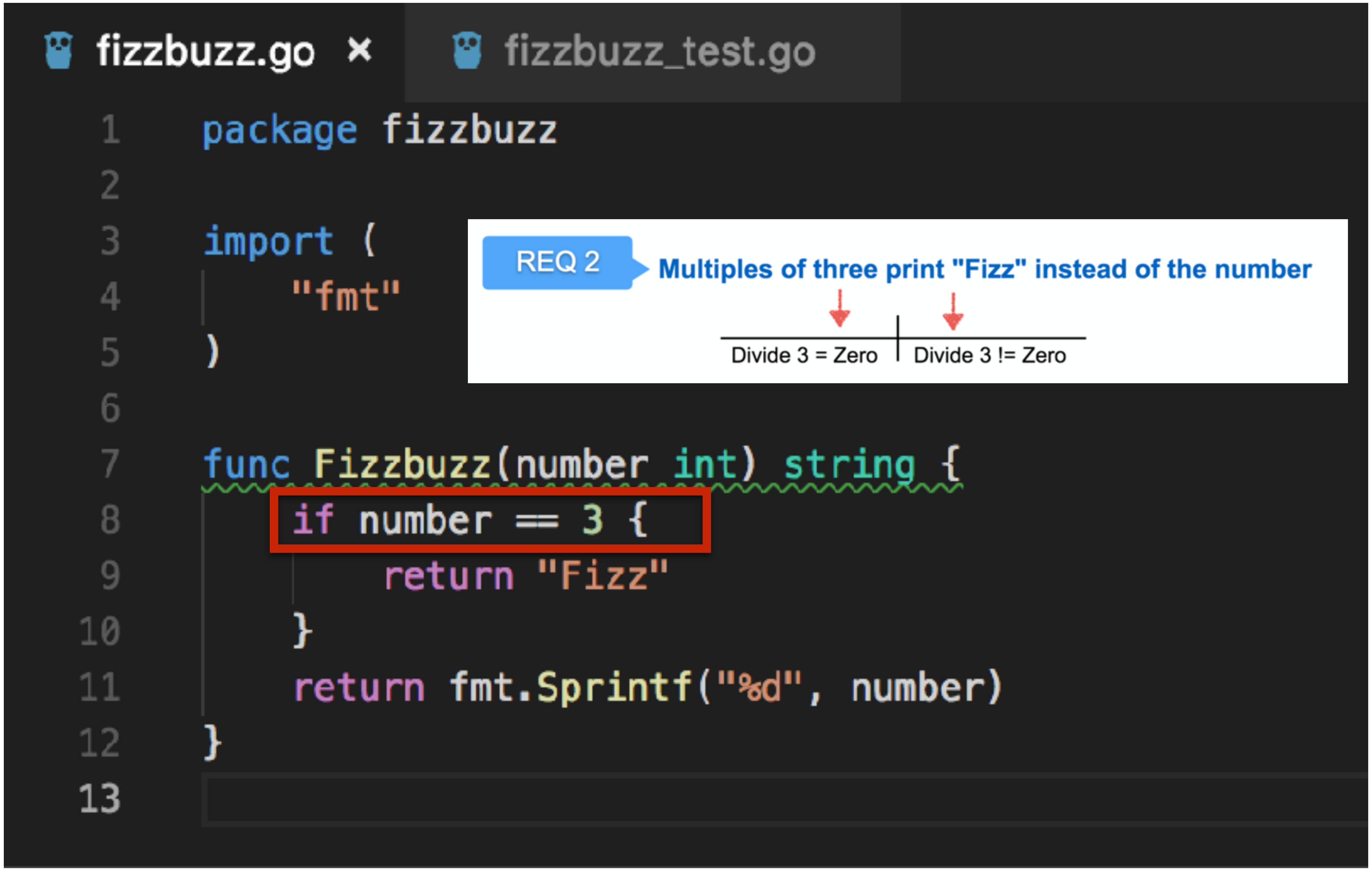
case 3
input 4 should be 4



fizzbuzz.go ×   fizzbuzz_test.go

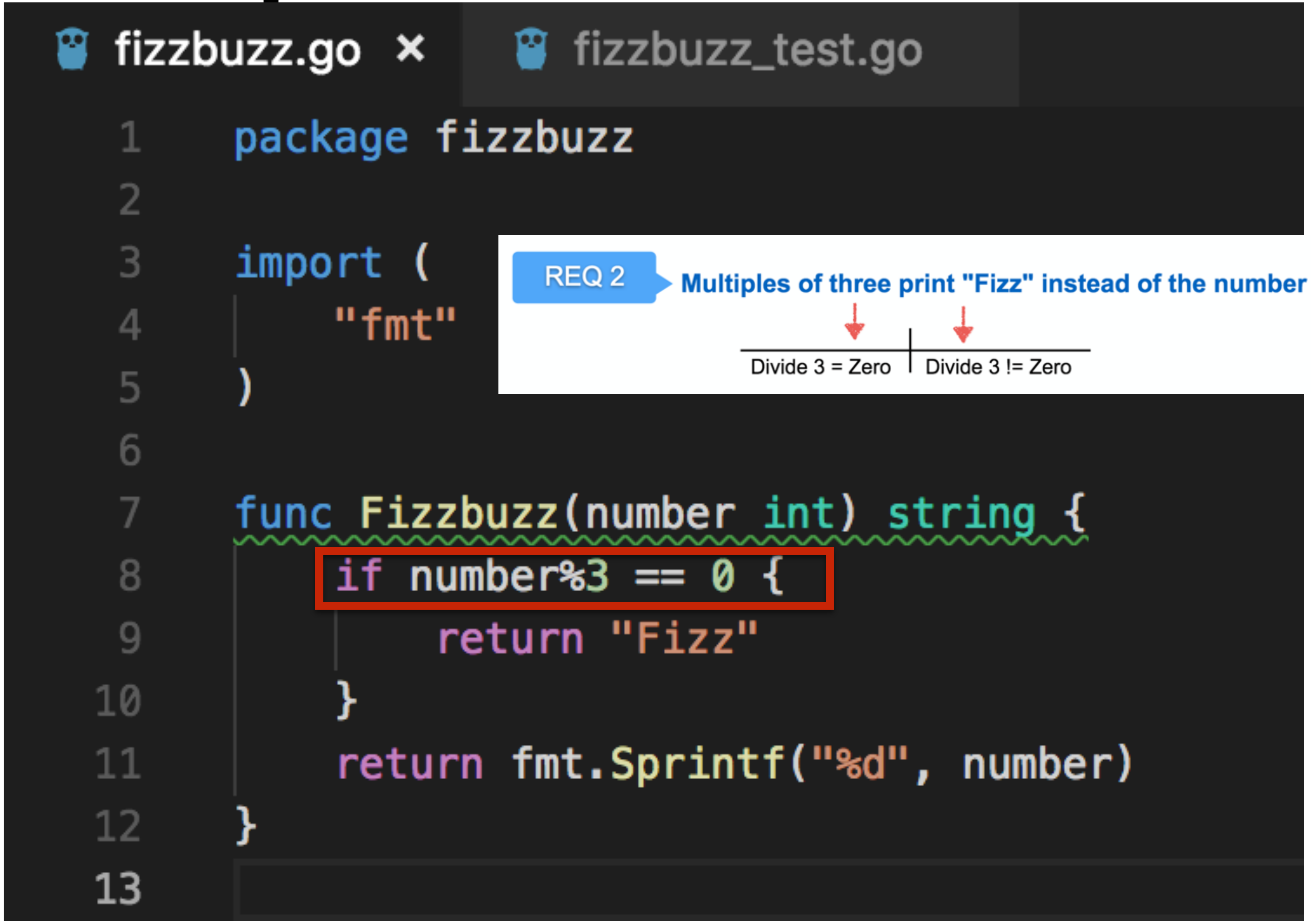
```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	if number == 3 {
		return "Fizz"
	}
	return fmt.Sprintf("%d", number)
}

```

case 3
input 4 should be 4



```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	if number == 3 {
		return "Fizz"
	}
	return fmt.Sprintf("%d", number)
}
```

case 3
input 4 should be 4



fizzbuzz.go × fizzbuzz_test.go

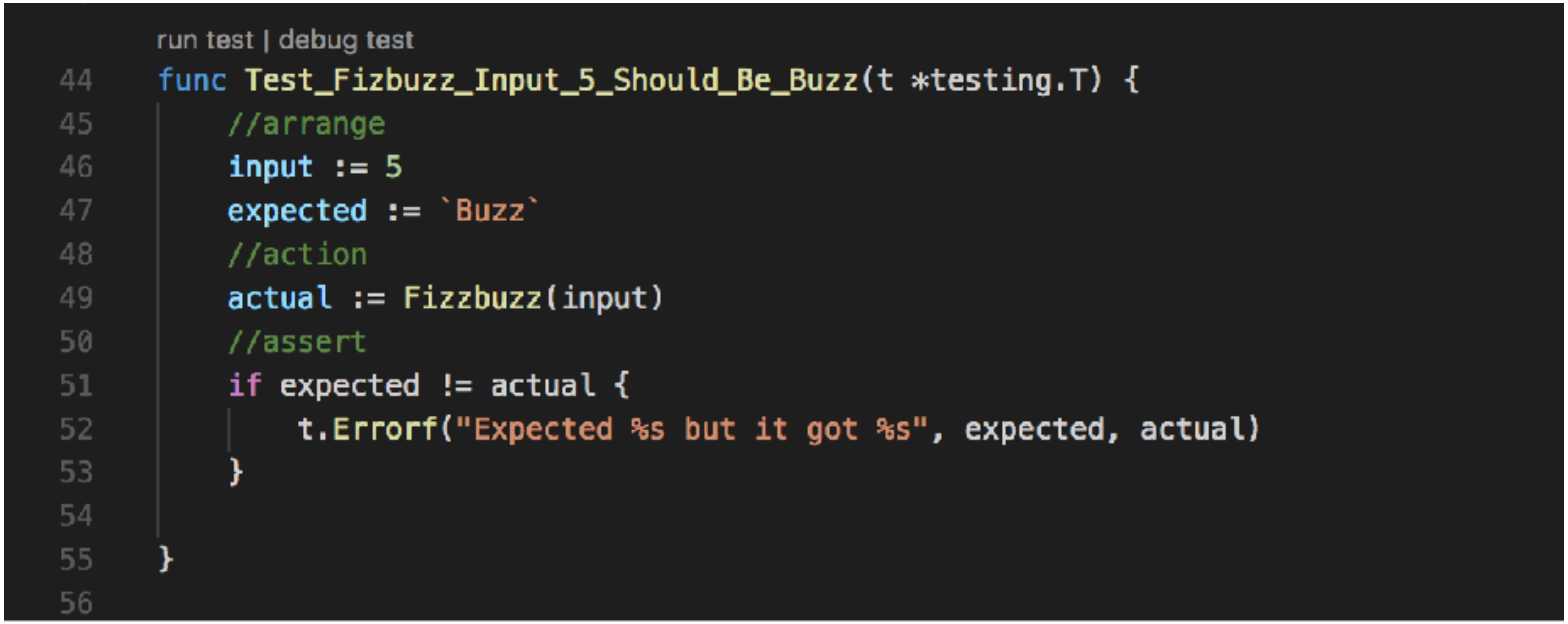
```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	if number%3 == 0 {
		return "Fizz"
	}
	return fmt.Sprintf("%d", number)
}
```

REQ 2 Multiples of three print "Fizz" instead of the number

case 4
input 5 should be Buzz

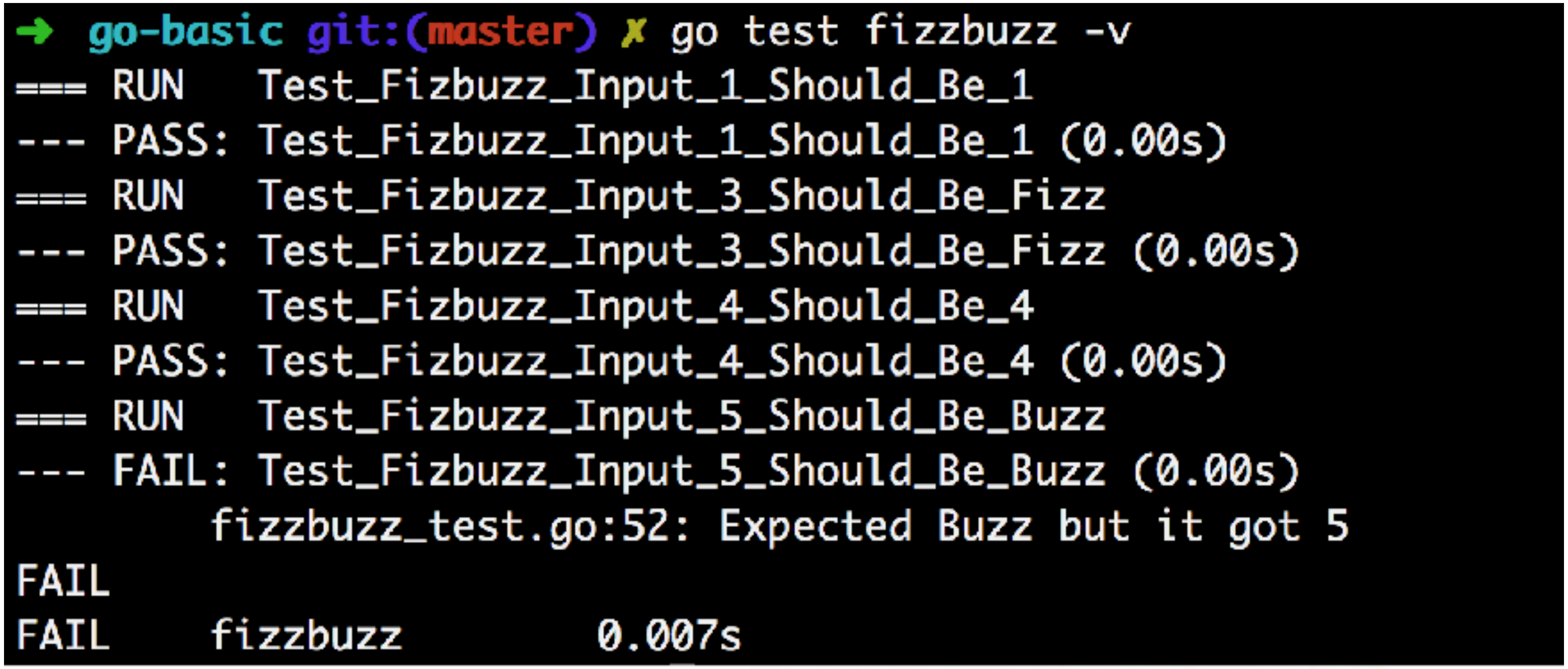
CASE	CONDITIONS	DATA	EXPECTED
1	Equal 1	1	1
2	More than 1 divide 3 = zero	3	Fizz
3	More than 1 divide 3 != zero	4	4
4	More than 1 divide 5 = zero	5	Buzz
5	More than 1 divide 5 != zero	7	7
6	More than 1 divide 3 and 5 = zero	15	FizzBuzz
7	More than 1 divide 3 and 5 != zero	17	17
8	Less than 100 divide 3 = zero	99	Fizz
9	Less than 100 divide 3 != zero	98	98
10	Less than 100 divide 5 = zero	95	Buzz
11	Less than 100 divide 5 = zero	94	94
12	Less than 100 divide 3 and 5 = zero	90	FizzBuzz
13	Less than 100 divide 3 and 5 != zero	98	98
14	Equal 100	100	Buzz
15	Less then 1	0	Not Display
16	More then 100	101	Not Display

case 4
input 5 should be Buzz



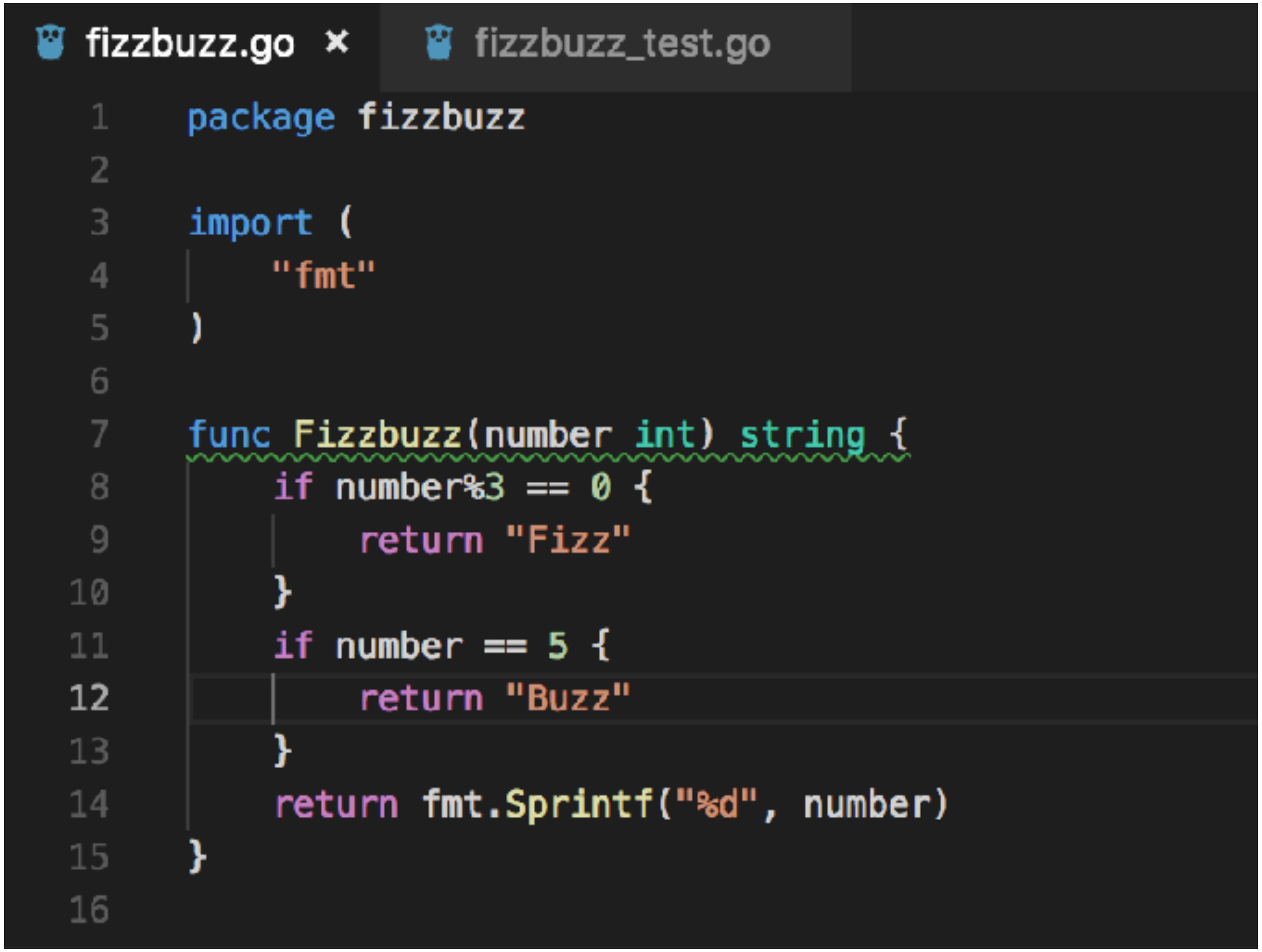
run test | debug test
```go
func Test_Fizbuzz_Input_5_Should_Be_Buzz(t *testing.T) {
	//arrange
	input := 5
	expected := `Buzz`
	//action
	actual := Fizzbuzz(input)
	//assert
	if expected != actual {
		t.Errorf("Expected %s but it got %s", expected, actual)
	}

}

```

case 4
input 5 should be Buzz



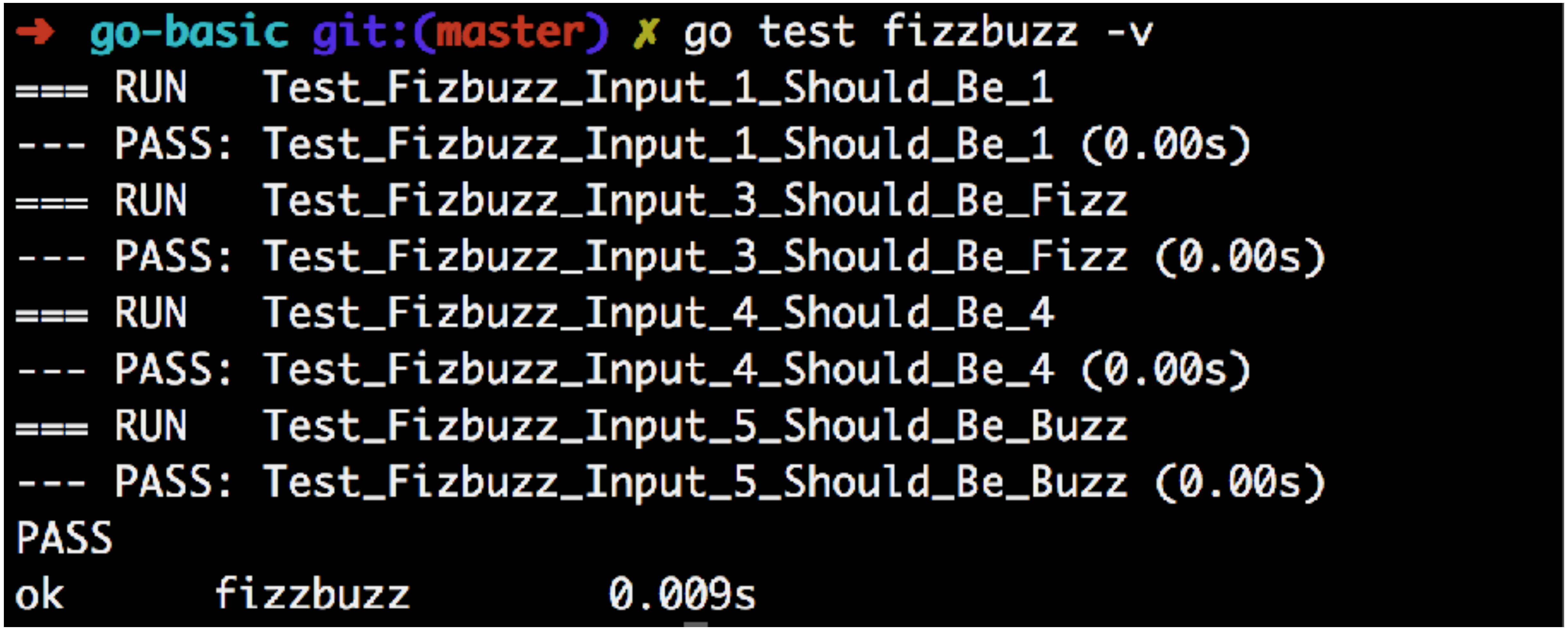
```
➜  go-basic git:(master) ✗ go test fizzbuzz -v
=== RUN   Test_Fizbuzz_Input_1_Should_Be_1
--- PASS: Test_Fizbuzz_Input_1_Should_Be_1 (0.00s)
=== RUN   Test_Fizbuzz_Input_3_Should_Be_Fizz
--- PASS: Test_Fizbuzz_Input_3_Should_Be_Fizz (0.00s)
=== RUN   Test_Fizbuzz_Input_4_Should_Be_4
--- PASS: Test_Fizbuzz_Input_4_Should_Be_4 (0.00s)
=== RUN   Test_Fizbuzz_Input_5_Should_Be_Buzz
--- FAIL: Test_Fizbuzz_Input_5_Should_Be_Buzz (0.00s)
        fizzbuzz_test.go:52: Expected Buzz but it got 5
FAIL
FAIL    fizzbuzz        0.007s
```

case 4
input 5 should be Buzz



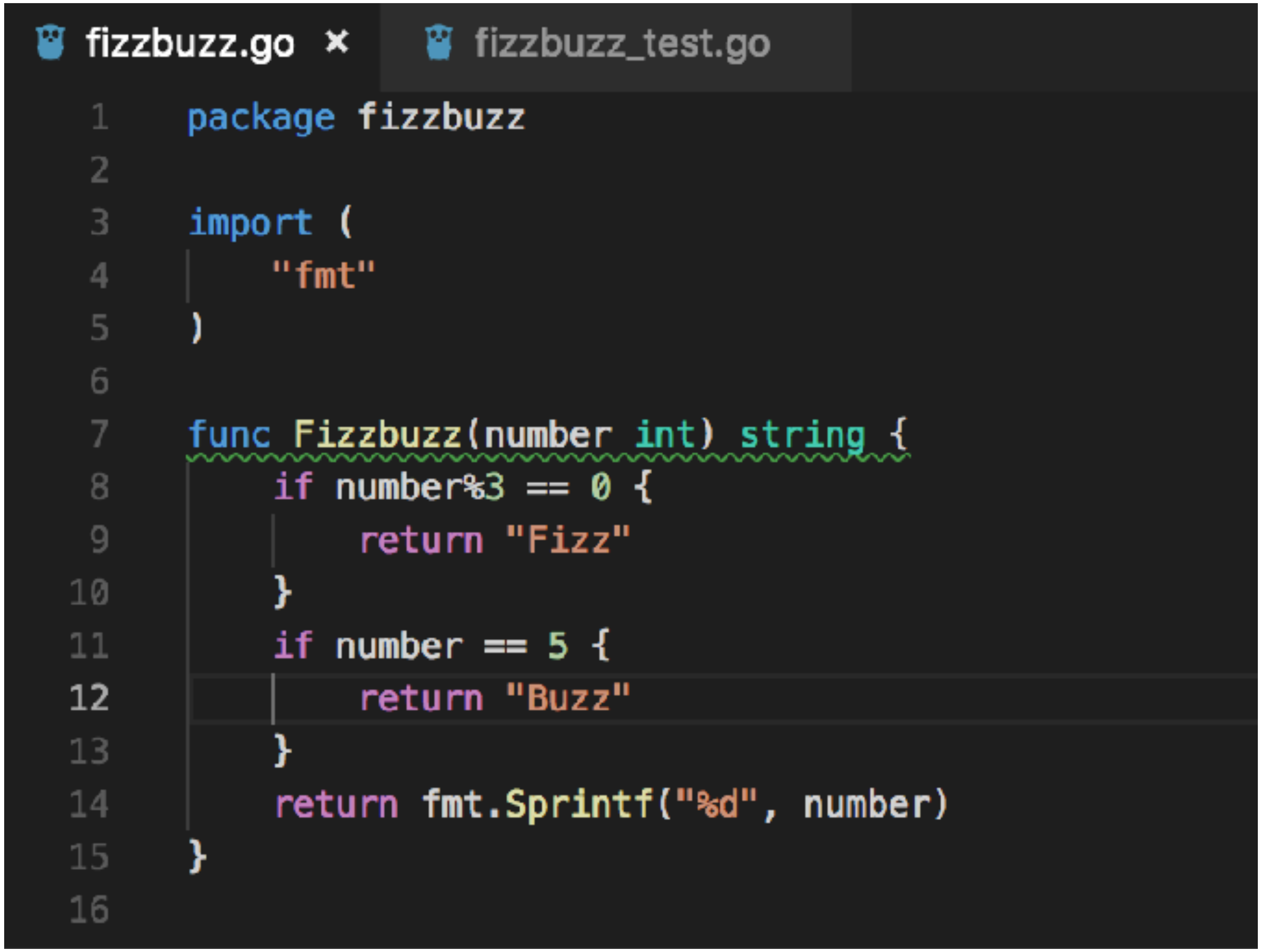
```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	if number%3 == 0 {
		return "Fizz"
	}
	if number == 5 {
		return "Buzz"
	}
	return fmt.Sprintf("%d", number)
}

```

case 4
input 5 should be Buzz



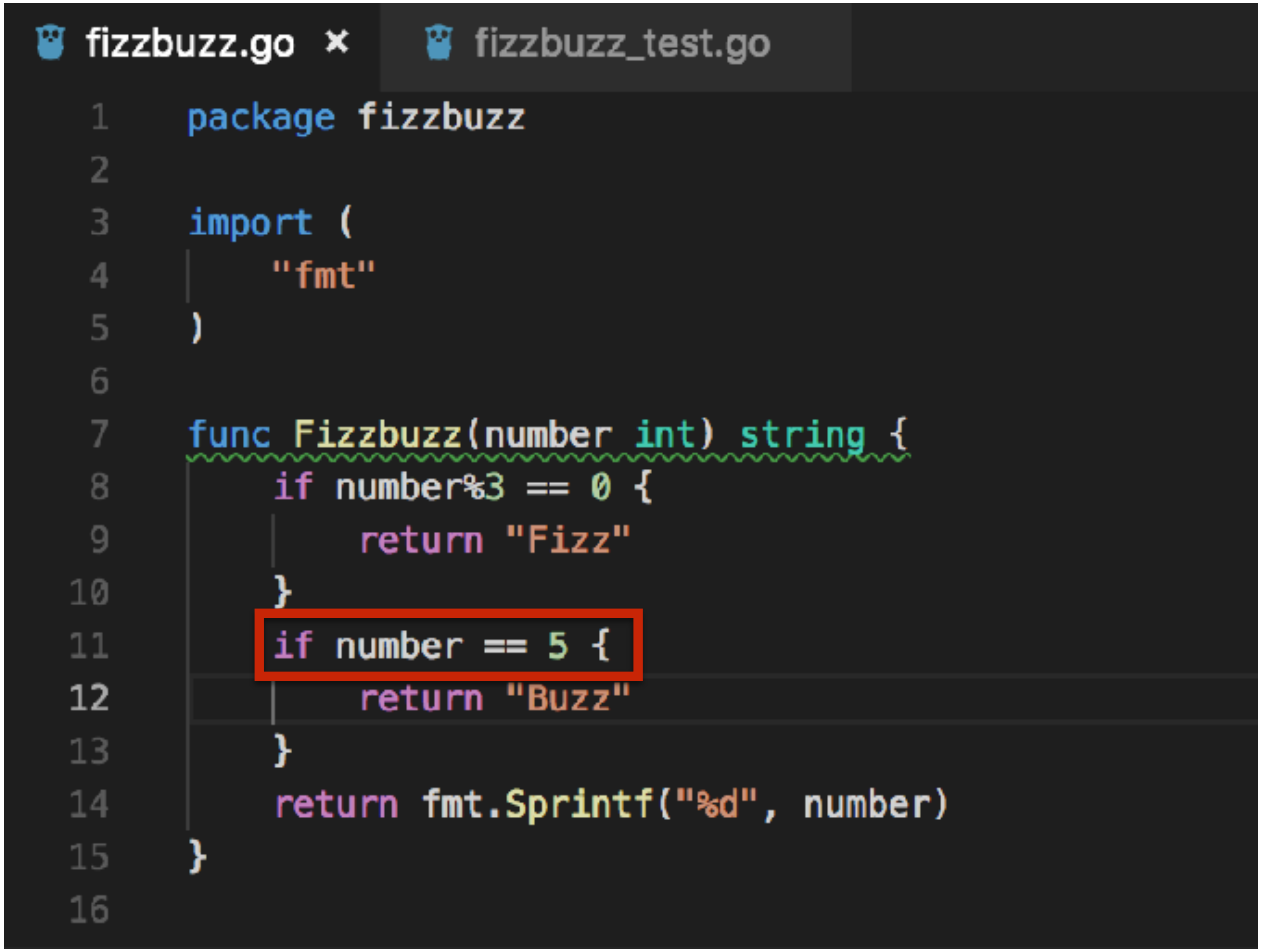
```
➜  go-basic git:(master) ✗ go test fizzbuzz -v
=== RUN   Test_Fizbuzz_Input_1_Should_Be_1
--- PASS: Test_Fizbuzz_Input_1_Should_Be_1 (0.00s)
=== RUN   Test_Fizbuzz_Input_3_Should_Be_Fizz
--- PASS: Test_Fizbuzz_Input_3_Should_Be_Fizz (0.00s)
=== RUN   Test_Fizbuzz_Input_4_Should_Be_4
--- PASS: Test_Fizbuzz_Input_4_Should_Be_4 (0.00s)
=== RUN   Test_Fizbuzz_Input_5_Should_Be_Buzz
--- PASS: Test_Fizbuzz_Input_5_Should_Be_Buzz (0.00s)
PASS
ok      fizzbuzz        0.009s
```

case 4
input 5 should be Buzz



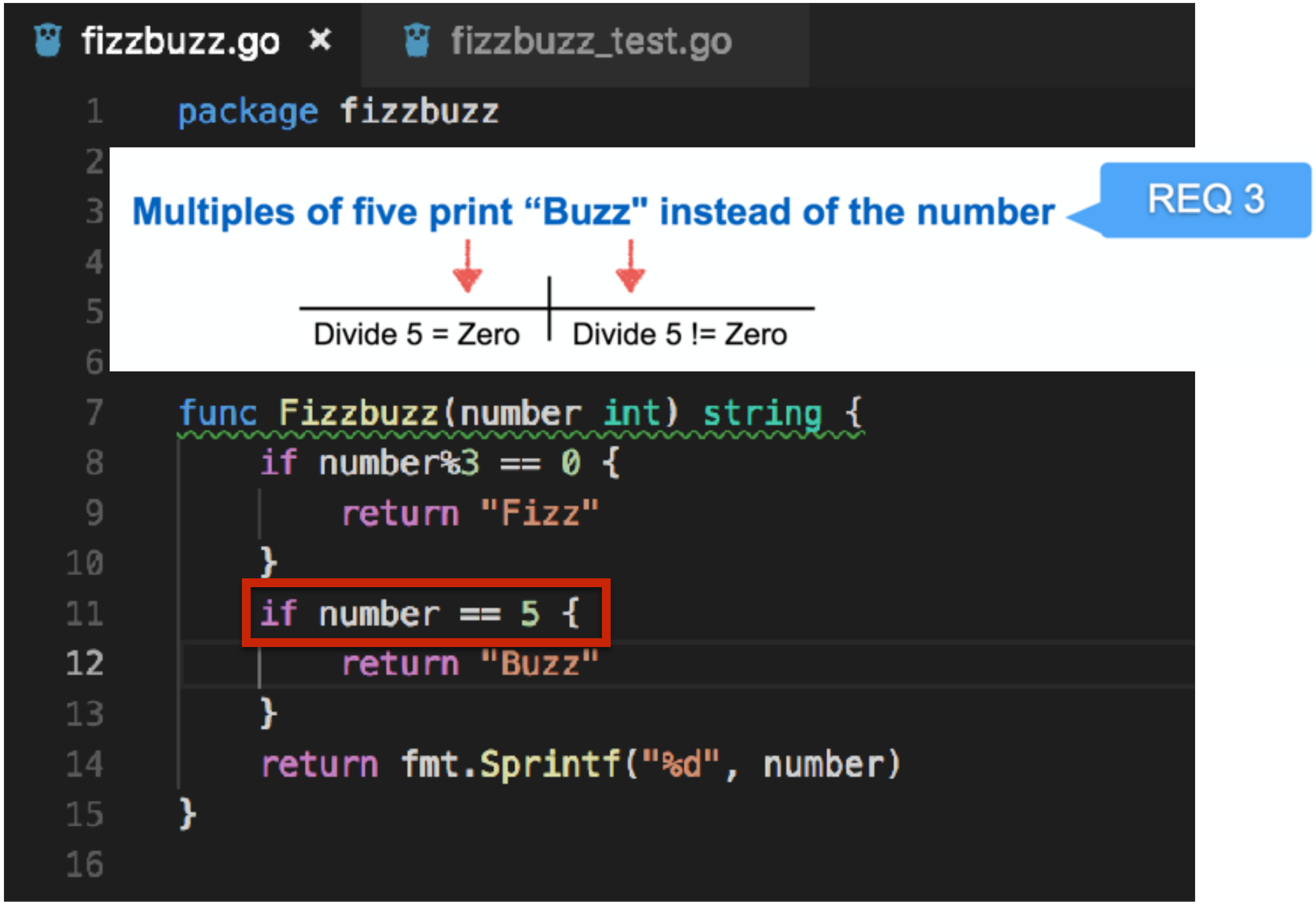
```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	if number%3 == 0 {
		return "Fizz"
	}
	if number == 5 {
		return "Buzz"
	}
	return fmt.Sprintf("%d", number)
}

```

case 4
input 5 should be Buzz



```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	if number%3 == 0 {
		return "Fizz"
	}
	if number == 5 {
		return "Buzz"
	}
	return fmt.Sprintf("%d", number)
}

```

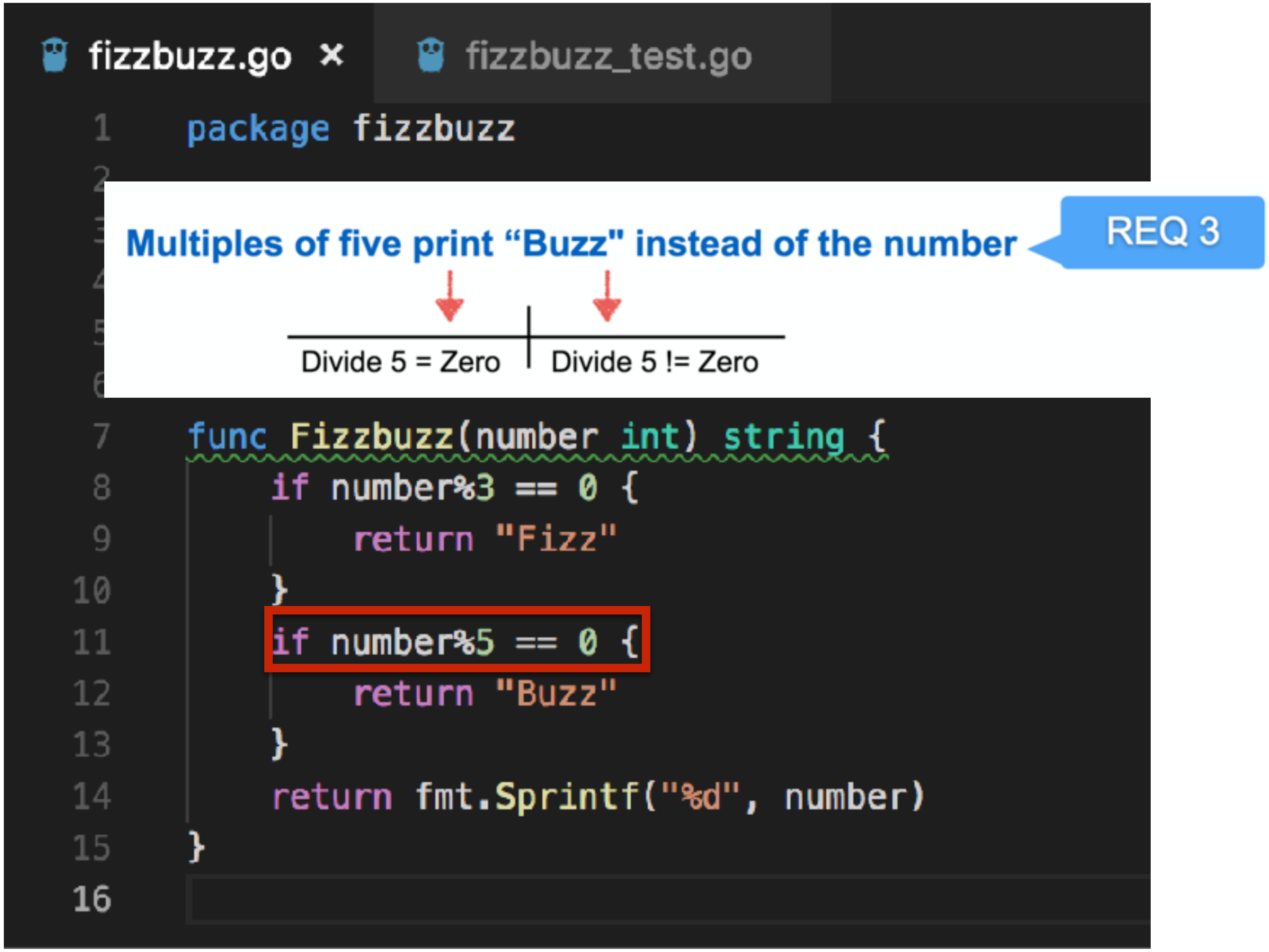
case 4
input 5 should be Buzz



Multiples of five print “Buzz" instead of the number

```go
package fizzbuzz

func Fizzbuzz(number int) string {
	if number%3 == 0 {
		return "Fizz"
	}
	if number == 5 {
		return "Buzz"
	}
	return fmt.Sprintf("%d", number)
}
```

case 4
input 5 should be Buzz



fizzbuzz.go × fizzbuzz_test.go

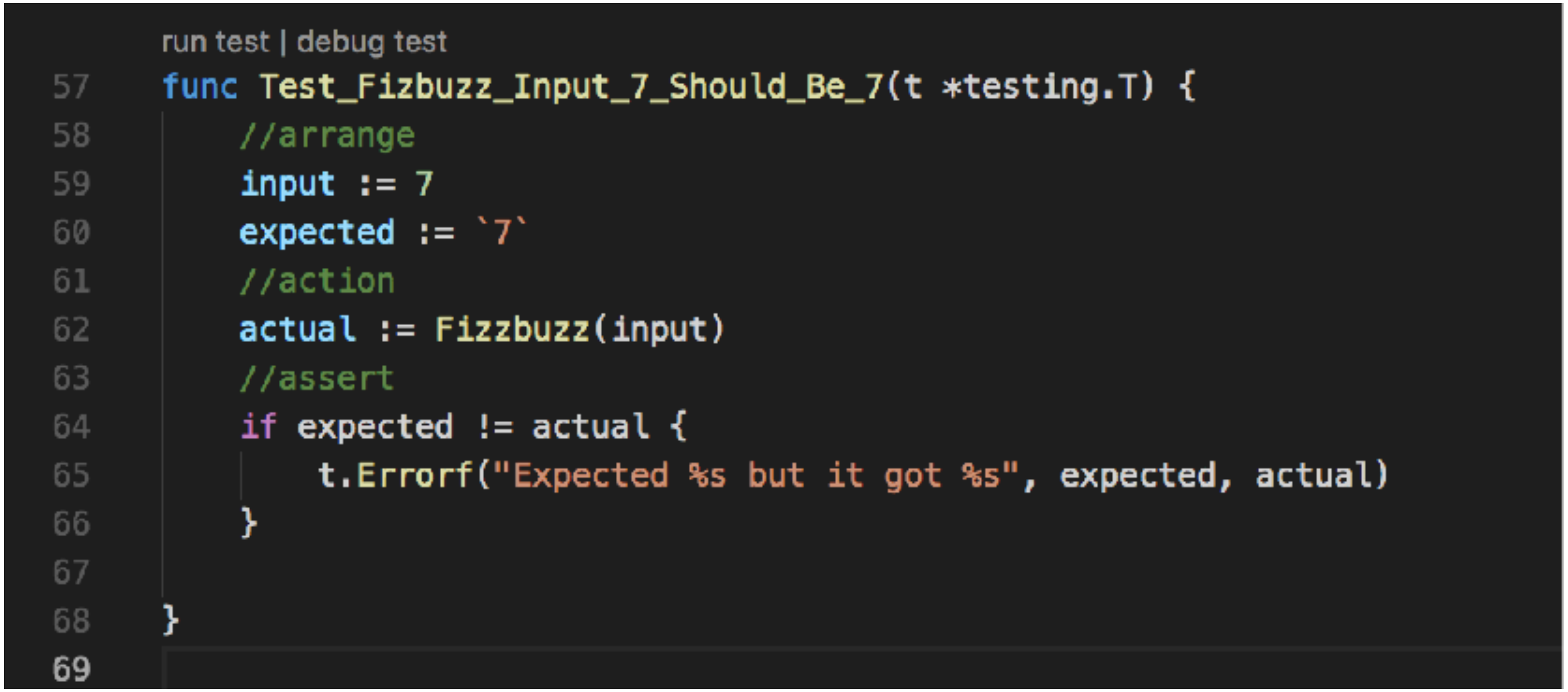
```go
package fizzbuzz

func Fizzbuzz(number int) string {
	if number%3 == 0 {
		return "Fizz"
	}
	if number%5 == 0 {
		return "Buzz"
	}
	return fmt.Sprintf("%d", number)
}
```

Multiples of five print "Buzz" instead of the number

case 5
input 7 should be 7

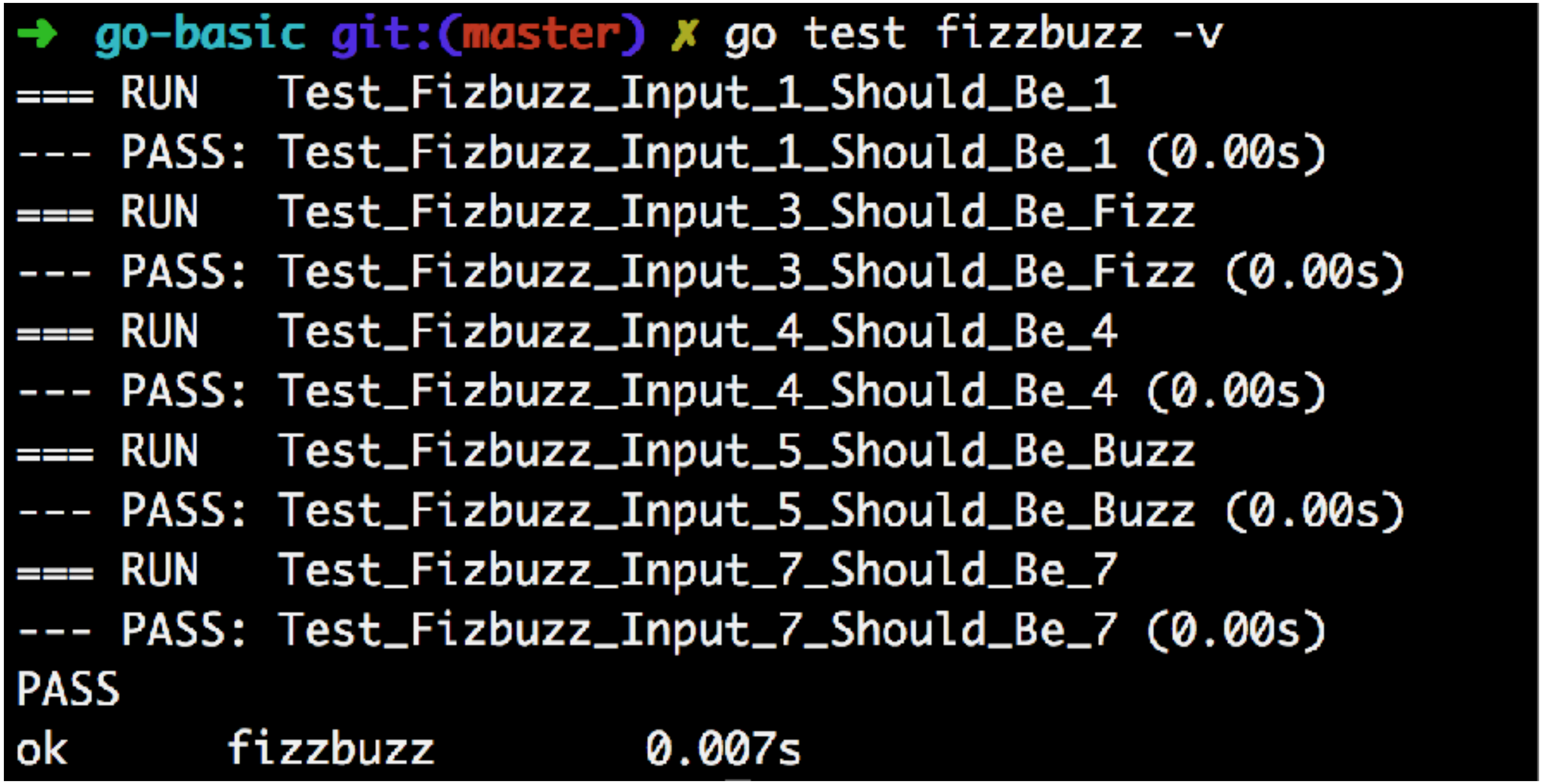
CASE	CONDITIONS	DATA	EXPECTED
1	Equal 1	1	1
2	More than 1 divide 3 = zero	3	Fizz
3	More than 1 divide 3 != zero	4	4
4	More than 1 divide 5 = zero	5	Buzz
5	More than 1 divide 5 != zero	7	7
6	More than 1 divide 3 and 5 = zero	15	FizzBuzz
7	More than 1 divide 3 and 5 != zero	17	17
8	Less than 100 divide 3 = zero	99	Fizz
9	Less than 100 divide 3 != zero	98	98
10	Less than 100 divide 5 = zero	95	Buzz
11	Less than 100 divide 5 = zero	94	94
12	Less than 100 divide 3 and 5 = zero	90	FizzBuzz
13	Less than 100 divide 3 and 5 != zero	98	98
14	Equal 100	100	Buzz
15	Less then 1	0	Not Display
16	More then 100	101	Not Display

case 5
input 7 should be 7



```go
func Test_Fizbuzz_Input_7_Should_Be_7(t *testing.T) {
	//arrange
	input := 7
	expected := `7`
	//action
	actual := Fizzbuzz(input)
	//assert
	if expected != actual {
		t.Errorf("Expected %s but it got %s", expected, actual)
	}

}
```

case 5
input 7 should be 7

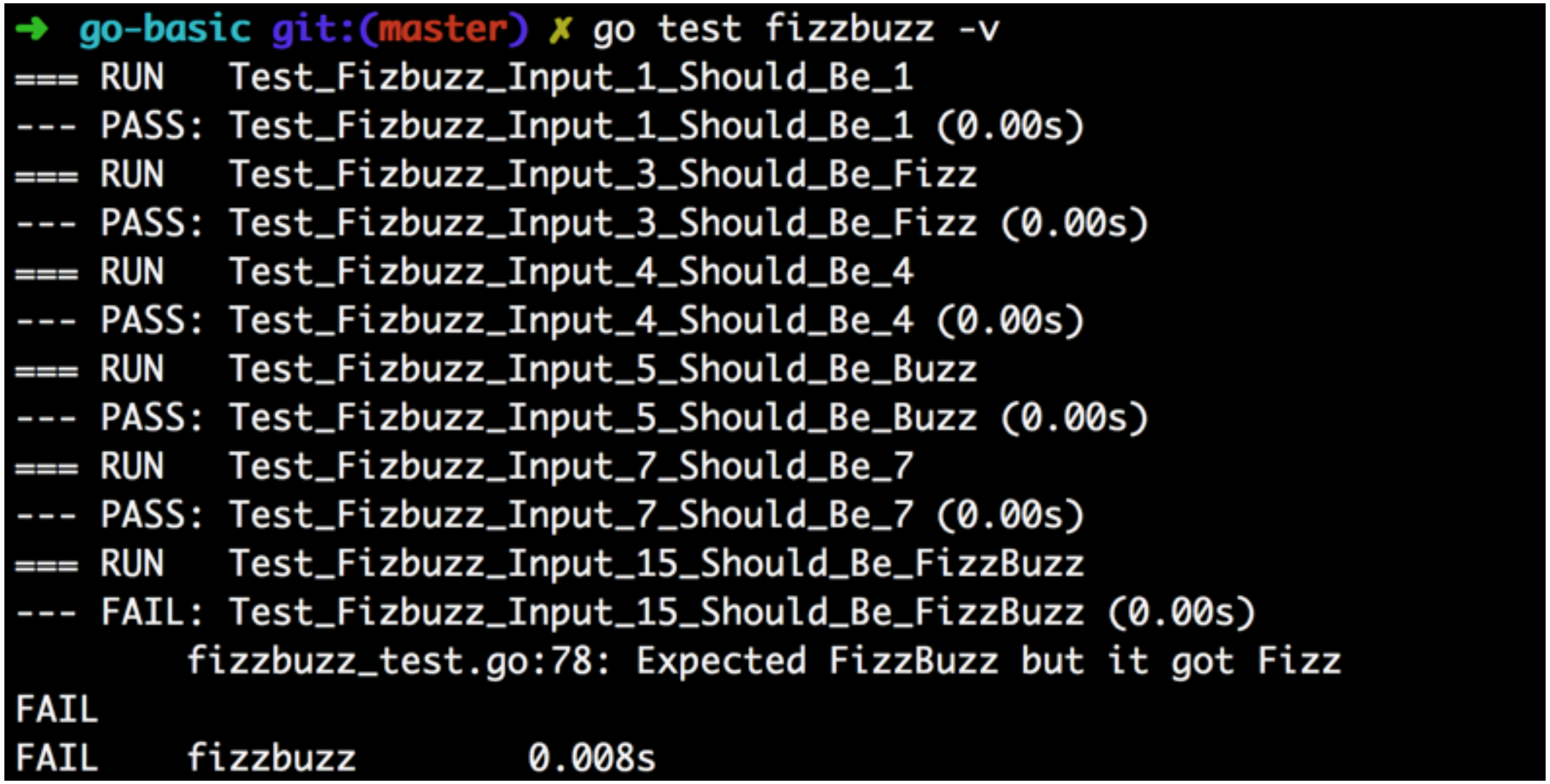


```
➜  go-basic git:(master) ✗ go test fizzbuzz -v
=== RUN   Test_Fizbuzz_Input_1_Should_Be_1
--- PASS: Test_Fizbuzz_Input_1_Should_Be_1 (0.00s)
=== RUN   Test_Fizbuzz_Input_3_Should_Be_Fizz
--- PASS: Test_Fizbuzz_Input_3_Should_Be_Fizz (0.00s)
=== RUN   Test_Fizbuzz_Input_4_Should_Be_4
--- PASS: Test_Fizbuzz_Input_4_Should_Be_4 (0.00s)
=== RUN   Test_Fizbuzz_Input_5_Should_Be_Buzz
--- PASS: Test_Fizbuzz_Input_5_Should_Be_Buzz (0.00s)
=== RUN   Test_Fizbuzz_Input_7_Should_Be_7
--- PASS: Test_Fizbuzz_Input_7_Should_Be_7 (0.00s)
PASS
ok      fizzbuzz        0.007s
```

case 6
input 15 should be FizzBuzz

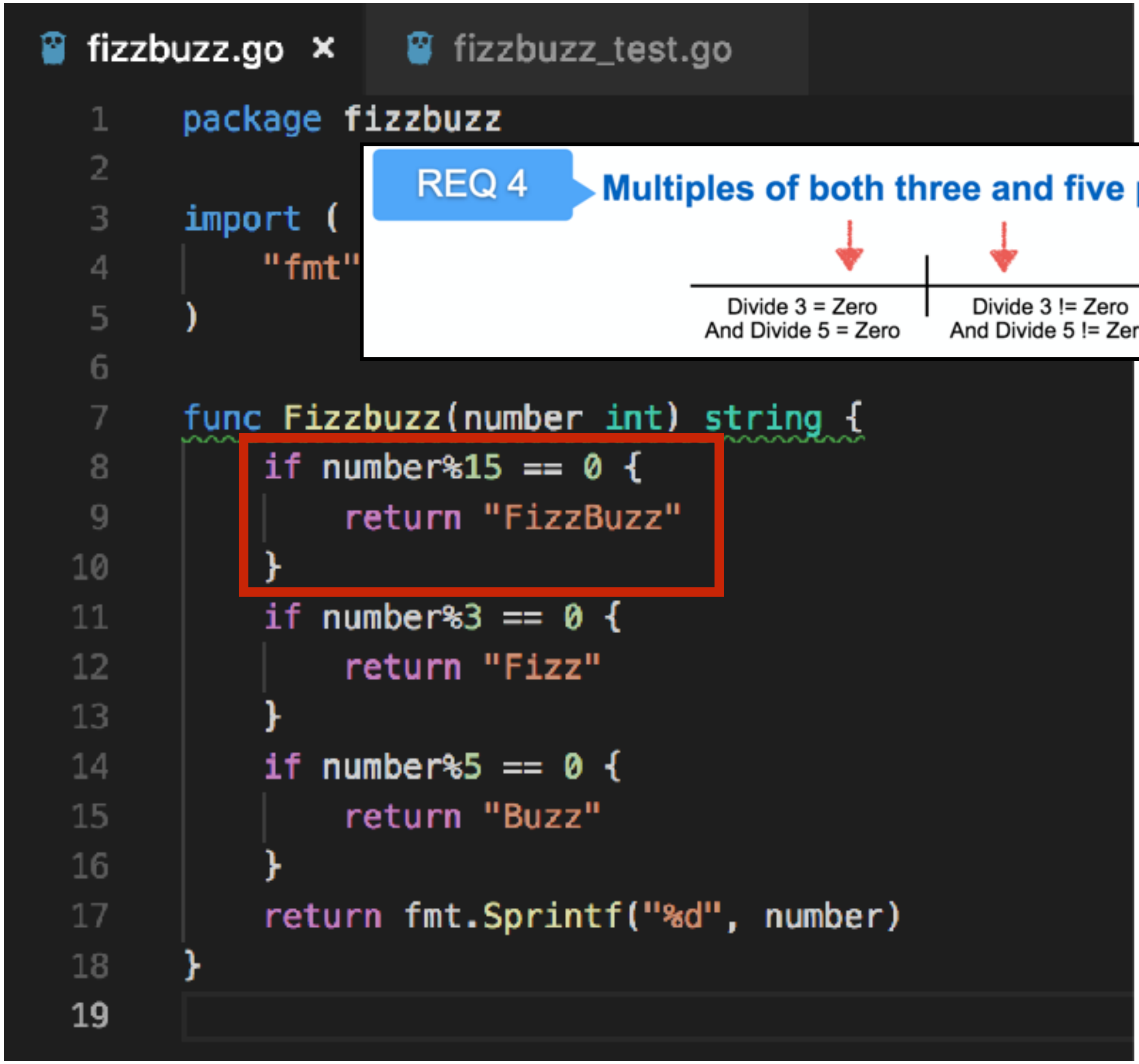
CASE	CONDITIONS	DATA	EXPECTED
1	Equal 1	1	1
2	More than 1 divide 3 = zero	3	Fizz
3	More than 1 divide 3 != zero	4	4
4	More than 1 divide 5 = zero	5	Buzz
5	More than 1 divide 5 != zero	7	7
6	More than 1 divide 3 and 5 = zero	15	FizzBuzz
7	More than 1 divide 3 and 5 != zero	17	17
8	Less than 100 divide 3 = zero	99	Fizz
9	Less than 100 divide 3 != zero	98	98
10	Less than 100 divide 5 = zero	95	Buzz
11	Less than 100 divide 5 = zero	94	94
12	Less than 100 divide 3 and 5 = zero	90	FizzBuzz
13	Less than 100 divide 3 and 5 != zero	98	98
14	Equal 100	100	Buzz
15	Less then 1	0	Not Display
16	More then 100	101	Not Display

case 6
input 15 should be FizzBuzz



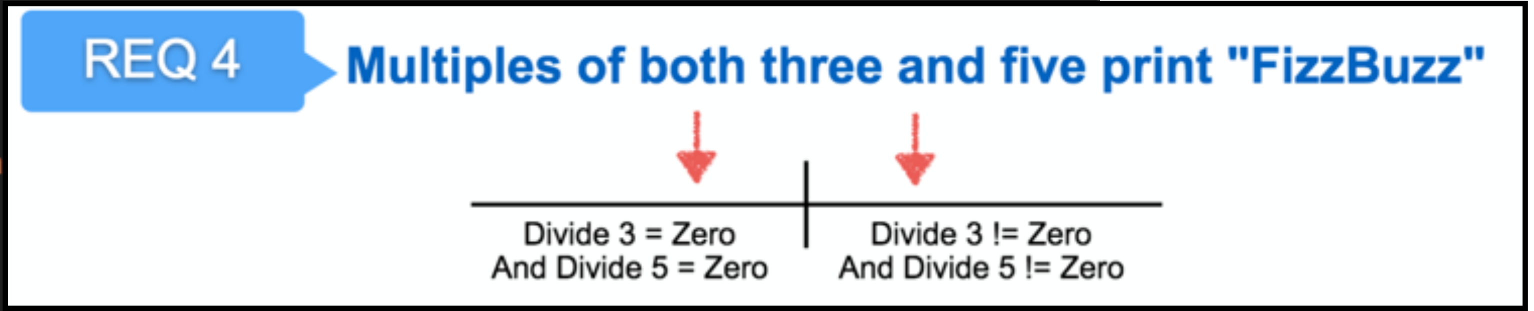
```
➜  go-basic git:(master) ✗ go test fizzbuzz -v
=== RUN   Test_Fizbuzz_Input_1_Should_Be_1
--- PASS: Test_Fizbuzz_Input_1_Should_Be_1 (0.00s)
=== RUN   Test_Fizbuzz_Input_3_Should_Be_Fizz
--- PASS: Test_Fizbuzz_Input_3_Should_Be_Fizz (0.00s)
=== RUN   Test_Fizbuzz_Input_4_Should_Be_4
--- PASS: Test_Fizbuzz_Input_4_Should_Be_4 (0.00s)
=== RUN   Test_Fizbuzz_Input_5_Should_Be_Buzz
--- PASS: Test_Fizbuzz_Input_5_Should_Be_Buzz (0.00s)
=== RUN   Test_Fizbuzz_Input_7_Should_Be_7
--- PASS: Test_Fizbuzz_Input_7_Should_Be_7 (0.00s)
=== RUN   Test_Fizbuzz_Input_15_Should_Be_FizzBuzz
--- FAIL: Test_Fizbuzz_Input_15_Should_Be_FizzBuzz (0.00s)
        fizzbuzz_test.go:78: Expected FizzBuzz but it got Fizz
FAIL
FAIL    fizzbuzz        0.008s
```

case 6
input 15 should be FizzBuzz



fizzbuzz.go × fizzbuzz_test.go

```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	if number%15 == 0 {
		return "FizzBuzz"
	}
	if number%3 == 0 {
		return "Fizz"
	}
	if number%5 == 0 {
		return "Buzz"
	}
	return fmt.Sprintf("%d", number)
}

```



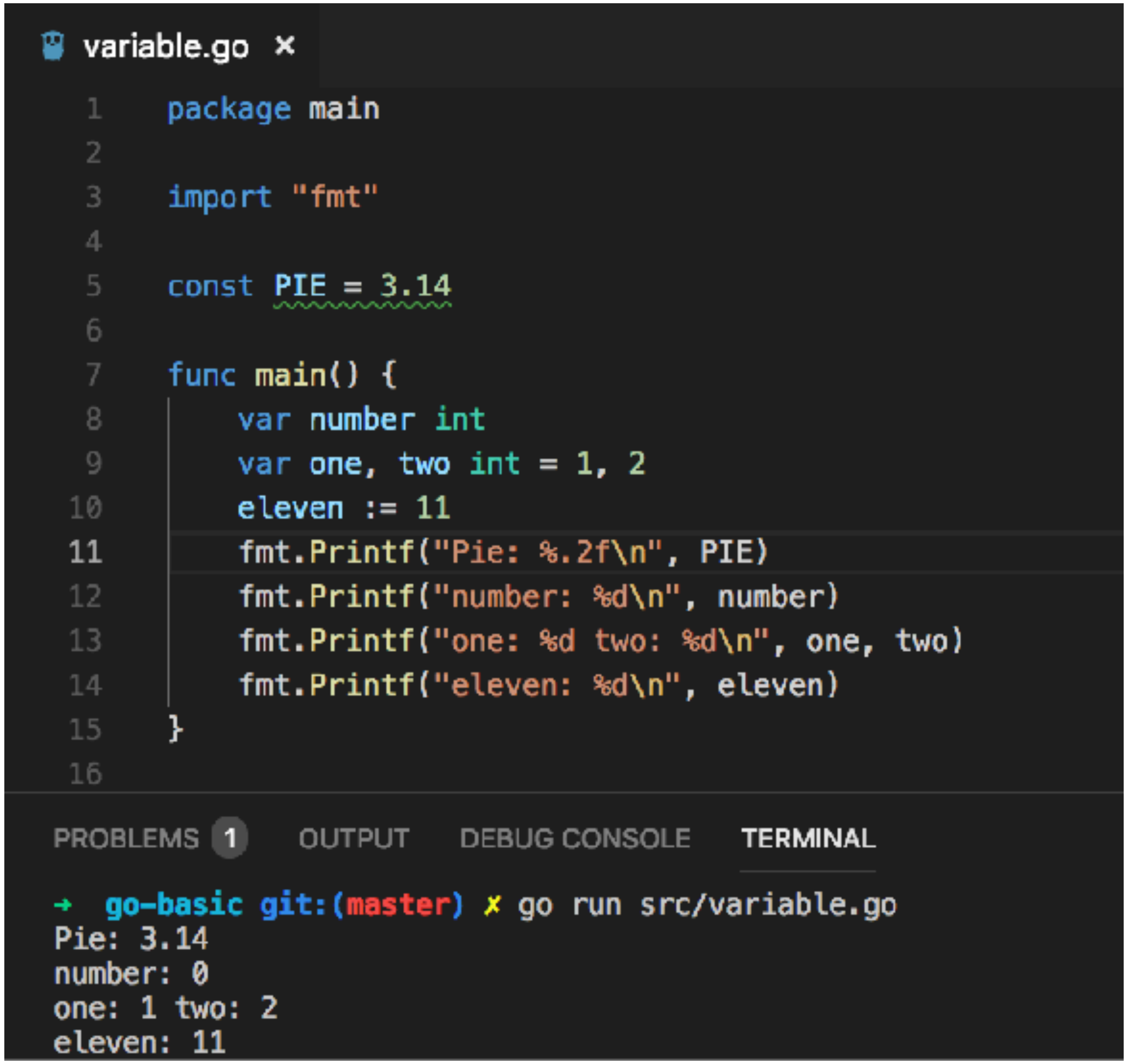
REQ 4
Multiples of both three and five print "FizzBuzz"
Divide 3 = Zero And Divide 5 = Zero
Divide 3 != Zero And Divide 5 != Zero

case 6
input 15 should be FizzBuzz

CASE	CONDITIONS	DATA	EXPECTED
1	Equal 1	1	1
2	More than 1 divide 3 = zero	3	Fizz
3	More than 1 divide 3 != zero	4	4
4	More than 1 divide 5 = zero	5	Buzz
5	More than 1 divide 5 != zero	7	7
6	More than 1 divide 3 and 5 = zero	15	FizzBuzz
7	More than 1 divide 3 and 5 != zero	17	17
8	Less than 100 divide 3 = zero	99	Fizz
9	Less than 100 divide 3 != zero	98	98
10	Less than 100 divide 5 = zero	95	Buzz
11	Less than 100 divide 5 = zero	94	94
12	Less than 100 divide 3 and 5 = zero	90	FizzBuzz
13	Less than 100 divide 3 and 5 != zero	98	98
14	Equal 100	100	Buzz
15	Less then 1	0	Not Display
16	More then 100	101	Not Display

Basic Syntax

Variable

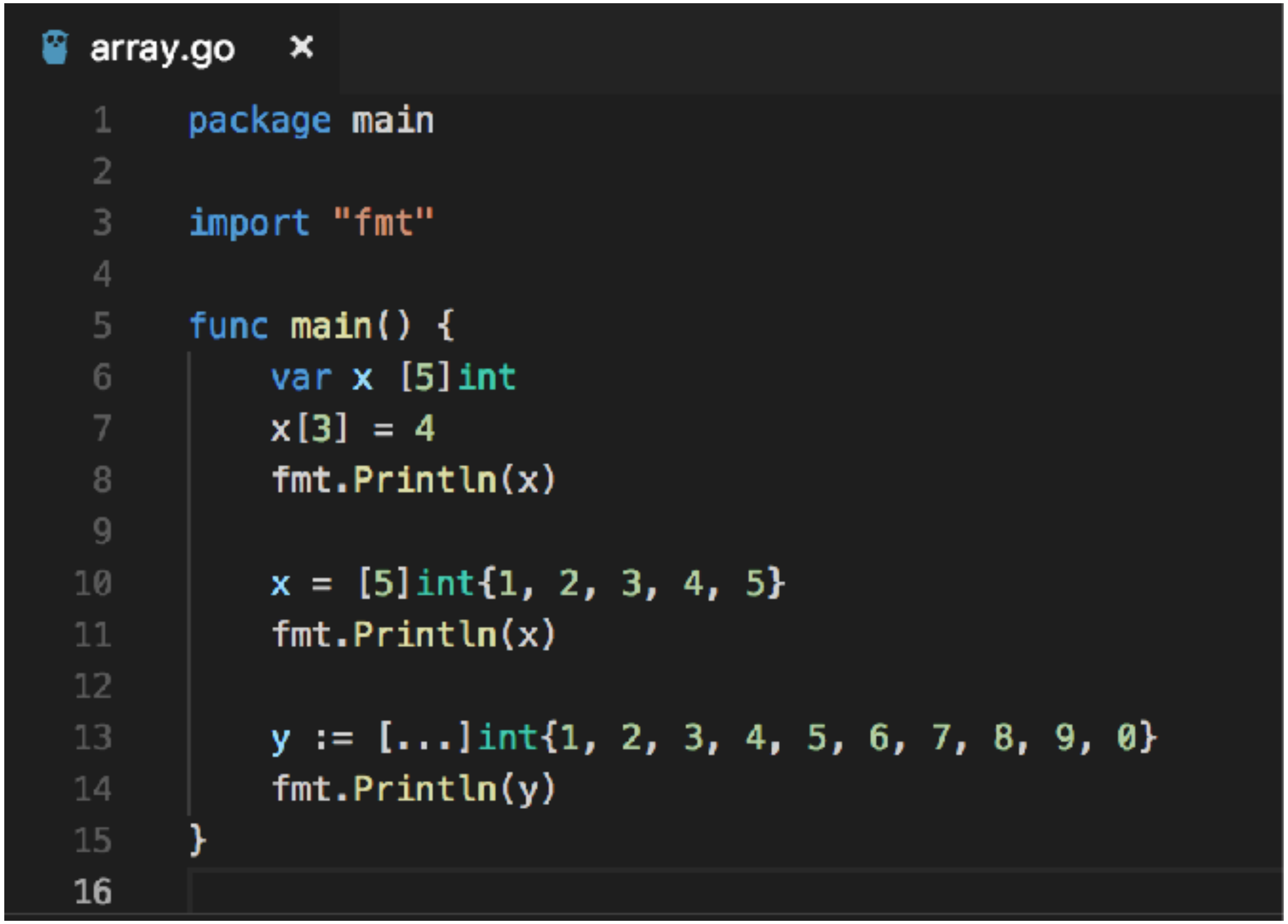


variable.go ×

```go
package main

import "fmt"

const PIE = 3.14

func main() {
	var number int
	var one, two int = 1, 2
	eleven := 11
	fmt.Printf("Pie: %.2f\n", PIE)
	fmt.Printf("number: %d\n", number)
	fmt.Printf("one: %d two: %d\n", one, two)
	fmt.Printf("eleven: %d\n", eleven)
}
```

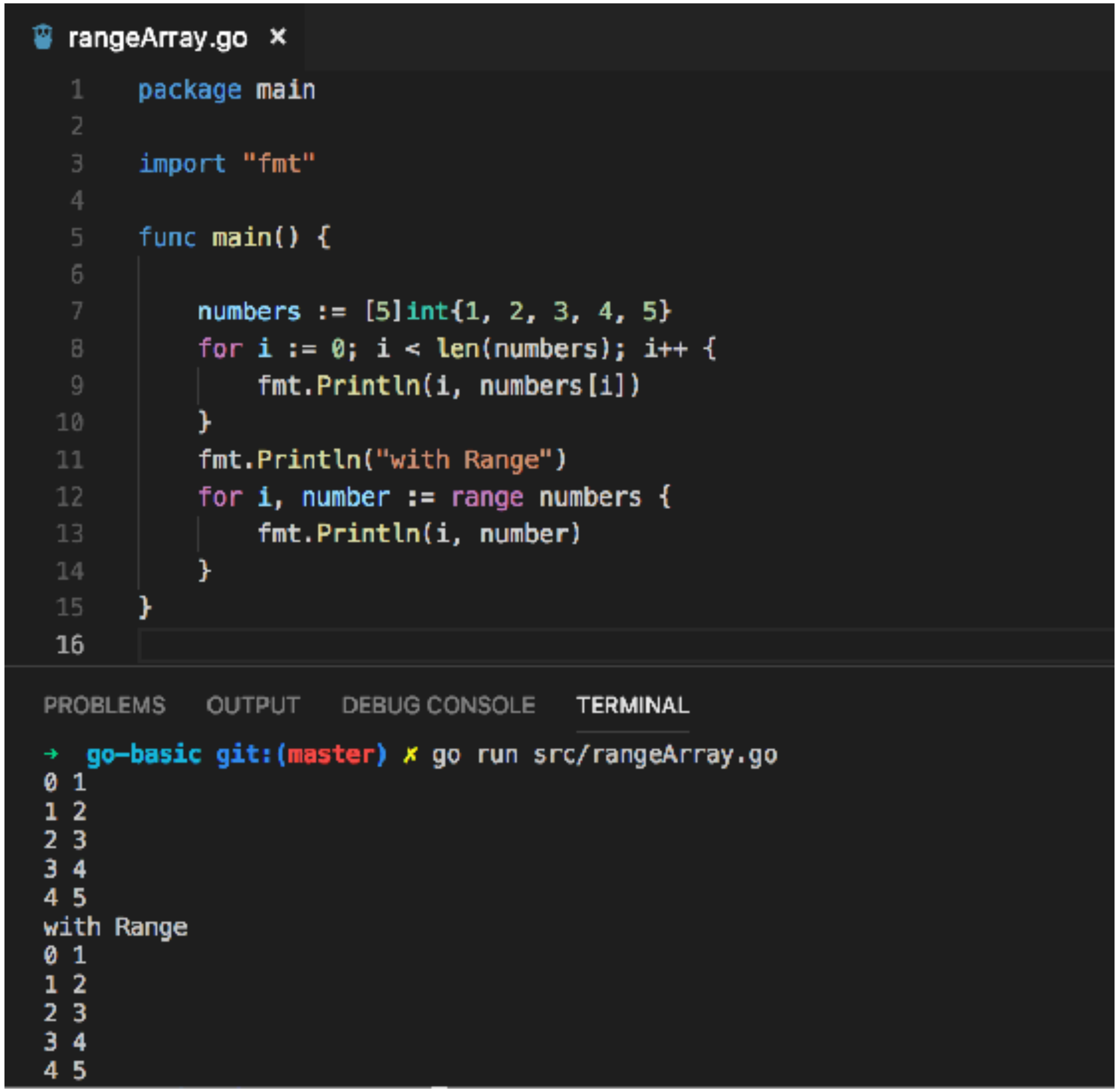
```
→  go-basic git:(master) ✗ go run src/variable.go
Pie: 3.14
number: 0
one: 1 two: 2
eleven: 11
```

Array


array.go ×

```go
package main

import "fmt"

func main() {
	var x [5]int
	x[3] = 4
	fmt.Println(x)

	x = [5]int{1, 2, 3, 4, 5}
	fmt.Println(x)

	y := [...]int{1, 2, 3, 4, 5, 6, 7, 8, 9, 0}
	fmt.Println(y)
}
```

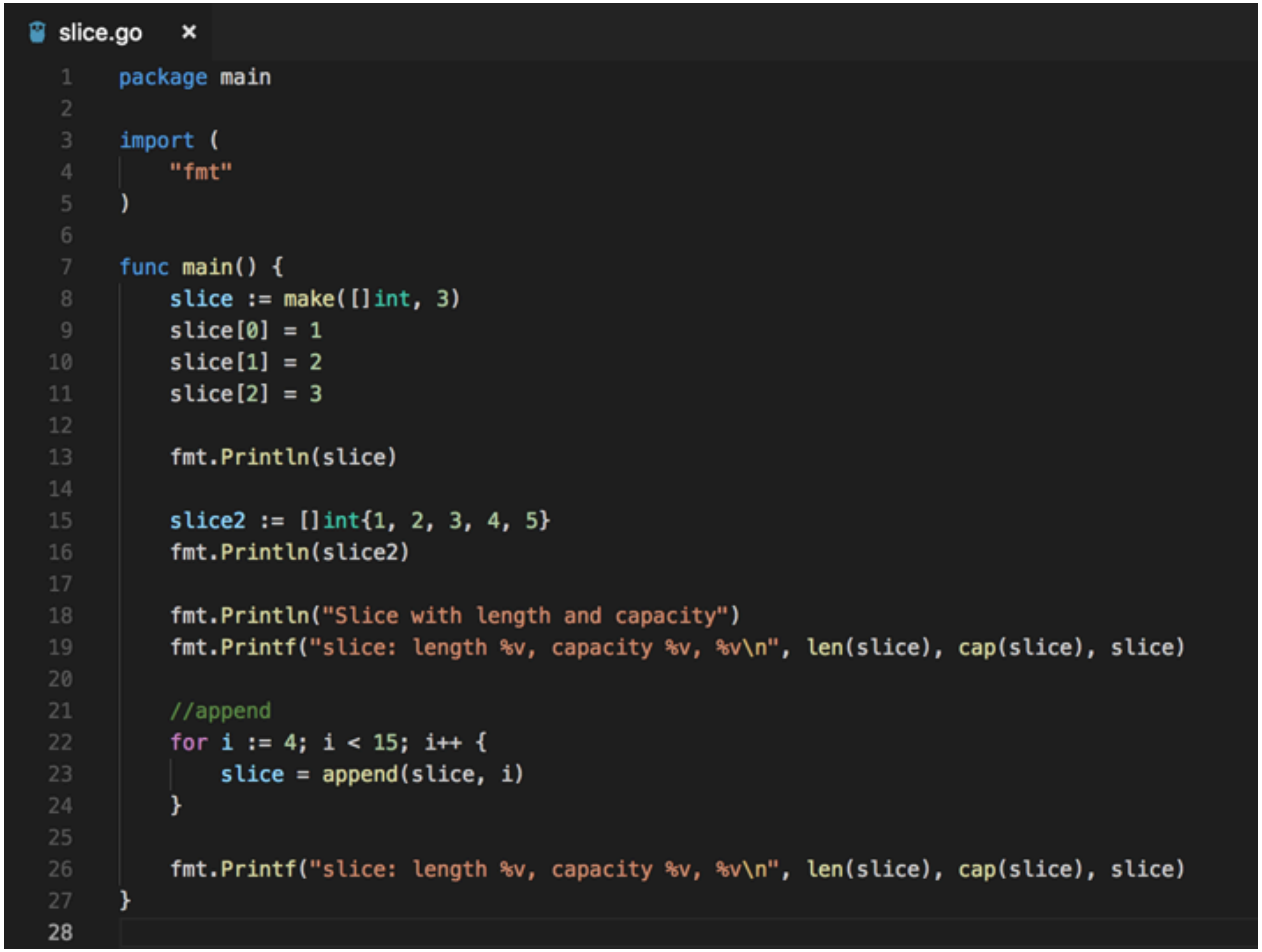
Array (2)



```go
package main

import "fmt"

func main() {

	numbers := [5]int{1, 2, 3, 4, 5}
	for i := 0; i < len(numbers); i++ {
		fmt.Println(i, numbers[i])
	}
	fmt.Println("with Range")
	for i, number := range numbers {
		fmt.Println(i, number)
	}
}

```

```
→ go-basic git:(master) ✗ go run src/rangeArray.go
0 1
1 2
2 3
3 4
4 5
with Range
0 1
1 2
2 3
3 4
4 5
```

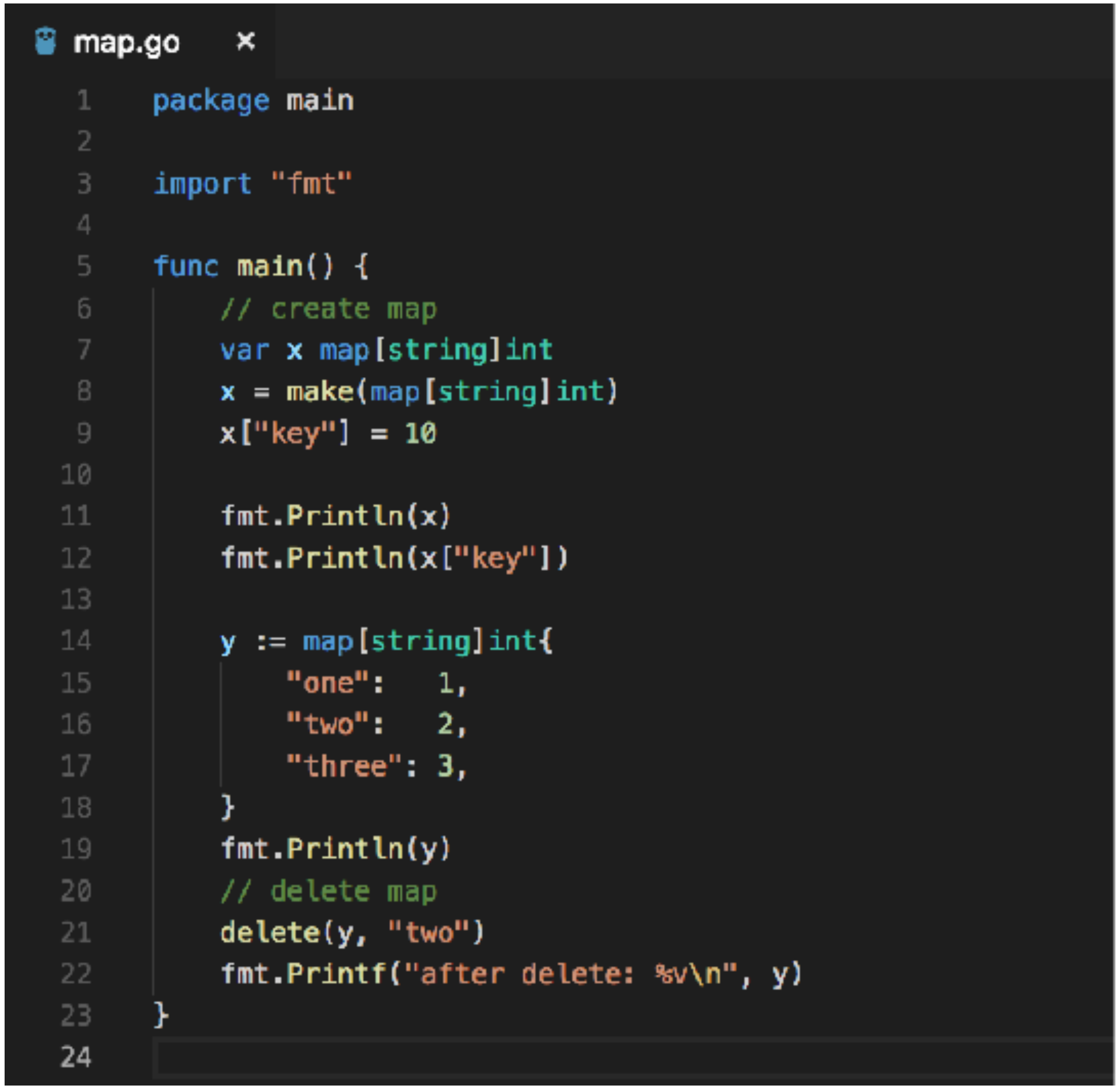
Slice



slice.go

```go
package main

import (
    "fmt"
)

func main() {
    slice := make([]int, 3)
    slice[0] = 1
    slice[1] = 2
    slice[2] = 3

    fmt.Println(slice)

    slice2 := []int{1, 2, 3, 4, 5}
    fmt.Println(slice2)

    fmt.Println("Slice with length and capacity")
    fmt.Printf("slice: length %v, capacity %v, %v\n", len(slice), cap(slice), slice)

    //append
    for i := 4; i < 15; i++ {
        slice = append(slice, i)
    }

    fmt.Printf("slice: length %v, capacity %v, %v\n", len(slice), cap(slice), slice)
}
```

Map

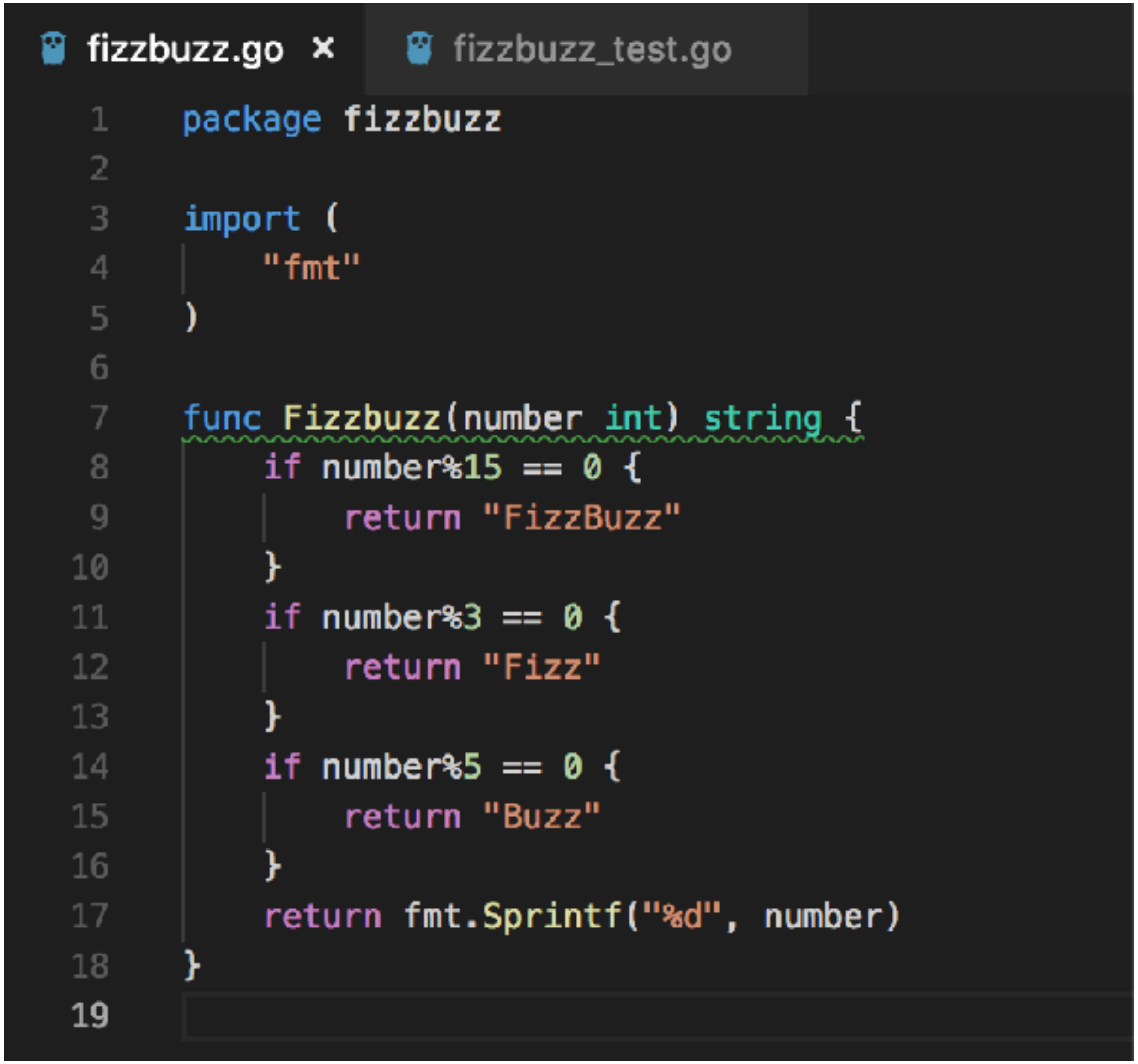


map.go ×

```go
package main

import "fmt"

func main() {
	// create map
	var x map[string]int
	x = make(map[string]int)
	x["key"] = 10

	fmt.Println(x)
	fmt.Println(x["key"])

	y := map[string]int{
		"one":   1,
		"two":   2,
		"three": 3,
	}
	fmt.Println(y)
	// delete map
	delete(y, "two")
	fmt.Printf("after delete: %v\n", y)
}
```

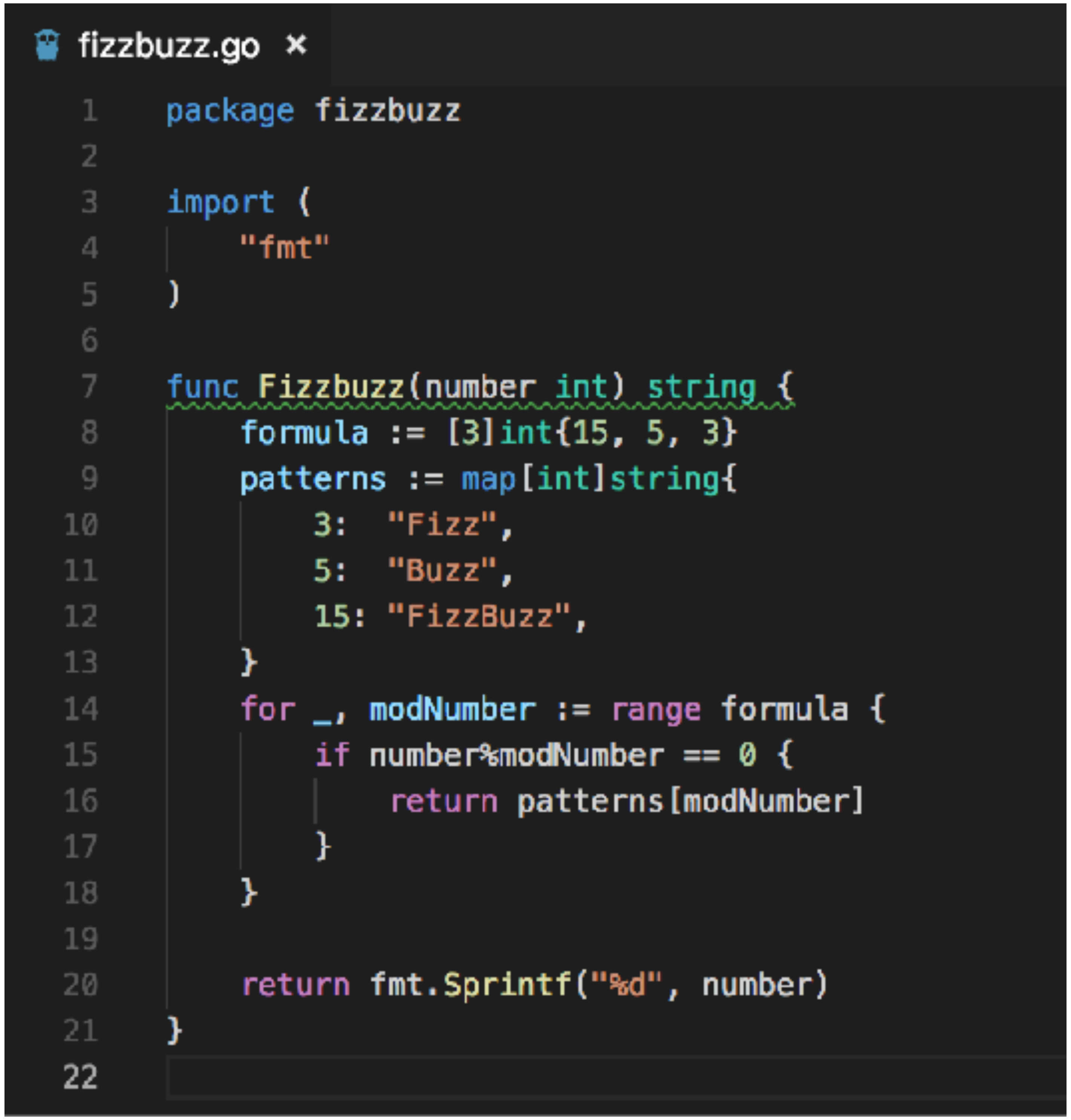
Adopt Array, Slice, Map into FizzBuzz

Remove If Duplication



```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	if number%15 == 0 {
		return "FizzBuzz"
	}
	if number%3 == 0 {
		return "Fizz"
	}
	if number%5 == 0 {
		return "Buzz"
	}
	return fmt.Sprintf("%d", number)
}

```

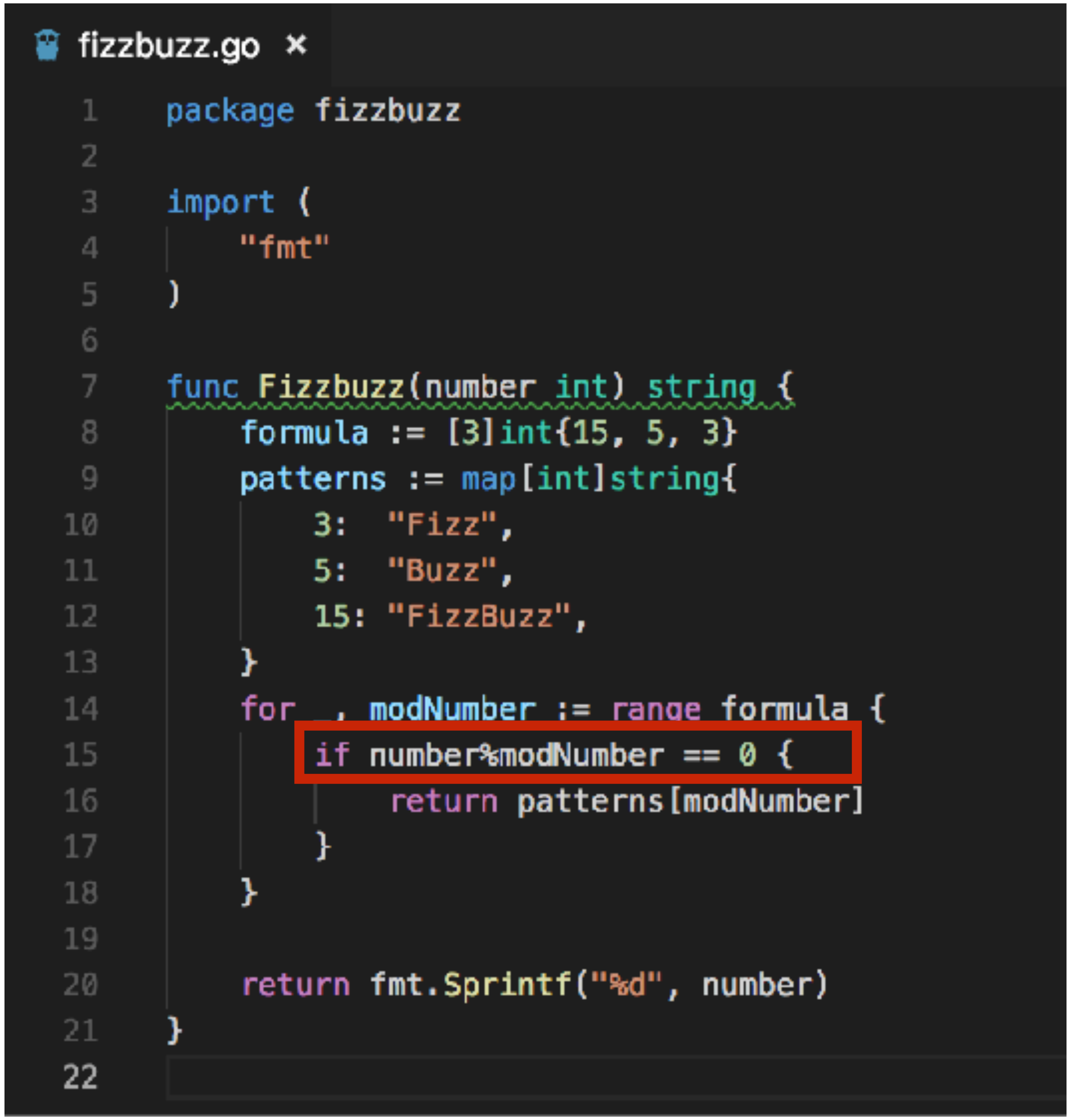
Remove If Duplication



fizzbuzz.go ×

```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	formula := [3]int{15, 5, 3}
	patterns := map[int]string{
		3:  "Fizz",
		5:  "Buzz",
		15: "FizzBuzz",
	}
	for _, modNumber := range formula {
		if number%modNumber == 0 {
			return patterns[modNumber]
		}
	}

	return fmt.Sprintf("%d", number)
}
```

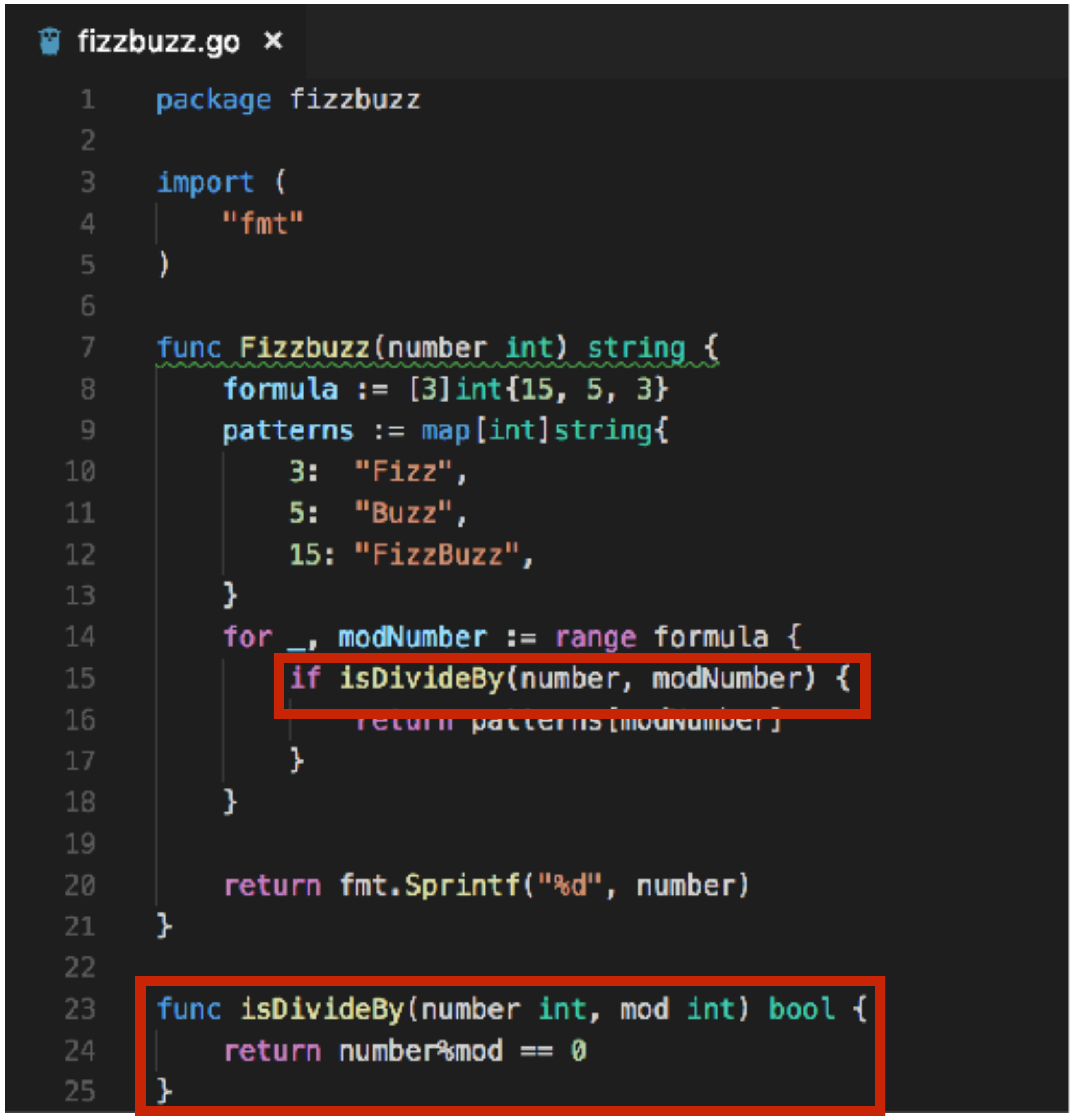
Remove If Duplication



fizzbuzz.go ×

```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	formula := [3]int{15, 5, 3}
	patterns := map[int]string{
		3:  "Fizz",
		5:  "Buzz",
		15: "FizzBuzz",
	}
	for _, modNumber := range formula {
		if number%modNumber == 0 {
			return patterns[modNumber]
		}
	}

	return fmt.Sprintf("%d", number)
}
```

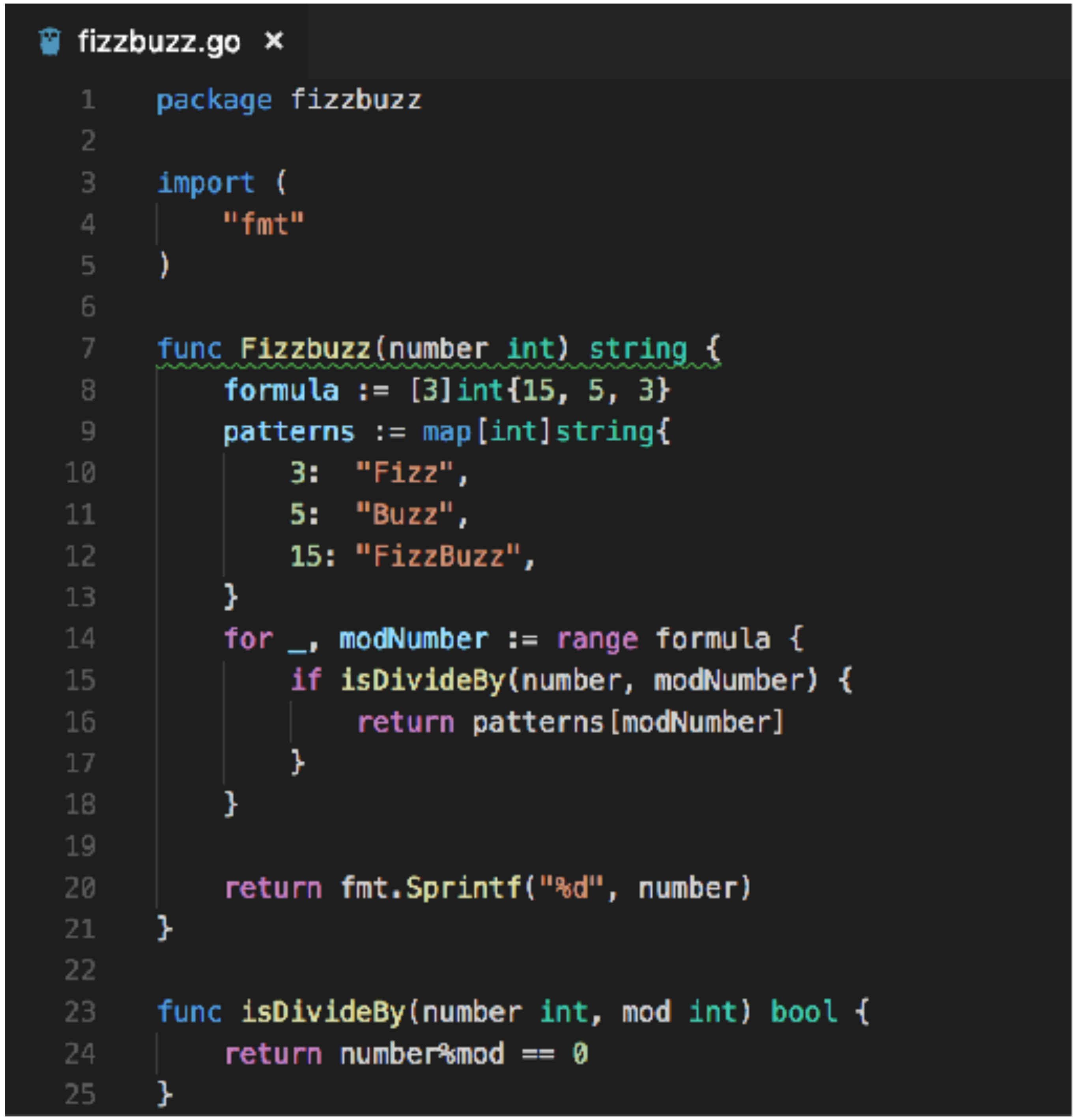
Remove If Duplication



fizzbuzz.go ×

```go
package fizzbuzz

import (
    "fmt"
)

func Fizzbuzz(number int) string {
    formula := [3]int{15, 5, 3}
    patterns := map[int]string{
        3:  "Fizz",
        5:  "Buzz",
        15: "FizzBuzz",
    }
    for _, modNumber := range formula {
        if isDivideBy(number, modNumber) {
            return patterns[modNumber]
        }
    }

    return fmt.Sprintf("%d", number)
}

func isDivideBy(number int, mod int) bool {
    return number%mod == 0
}
```

Remove If Duplication



fizzbuzz.go ×

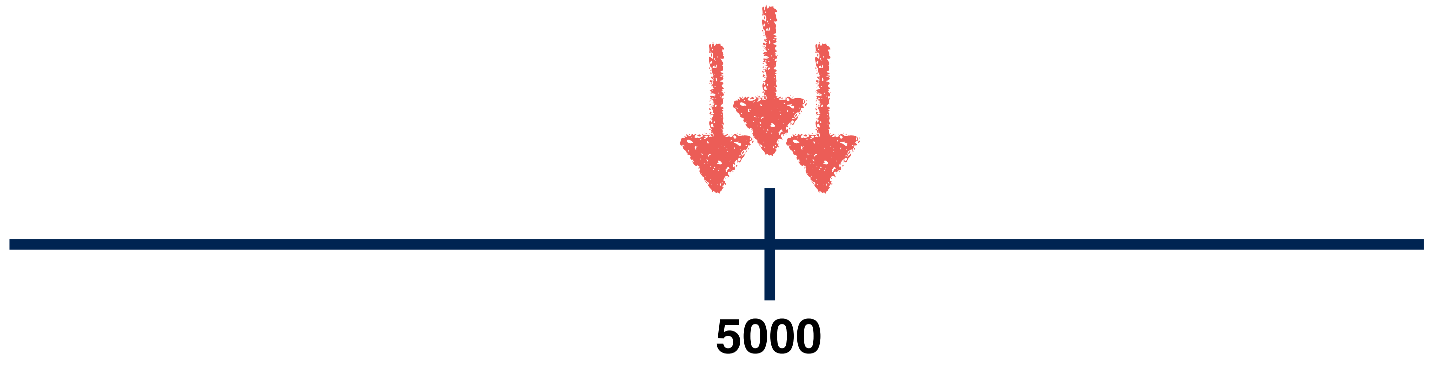
```go
package fizzbuzz

import (
	"fmt"
)

func Fizzbuzz(number int) string {
	formula := [3]int{15, 5, 3}
	patterns := map[int]string{
		3:  "Fizz",
		5:  "Buzz",
		15: "FizzBuzz",
	}
	for _, modNumber := range formula {
		if isDivideBy(number, modNumber) {
			return patterns[modNumber]
		}
	}

	return fmt.Sprintf("%d", number)
}

func isDivideBy(number int, mod int) bool {
	return number%mod == 0
}
```

Q & A

Start with Problem

Write an api that receive account info and amount of money that want to transfer, if amount of money is over 5000 cannot be transferred will prints status “can not be transfer”. if it can be transfer, will prints status “transferable” and return amount of money, balance before and after transfer.

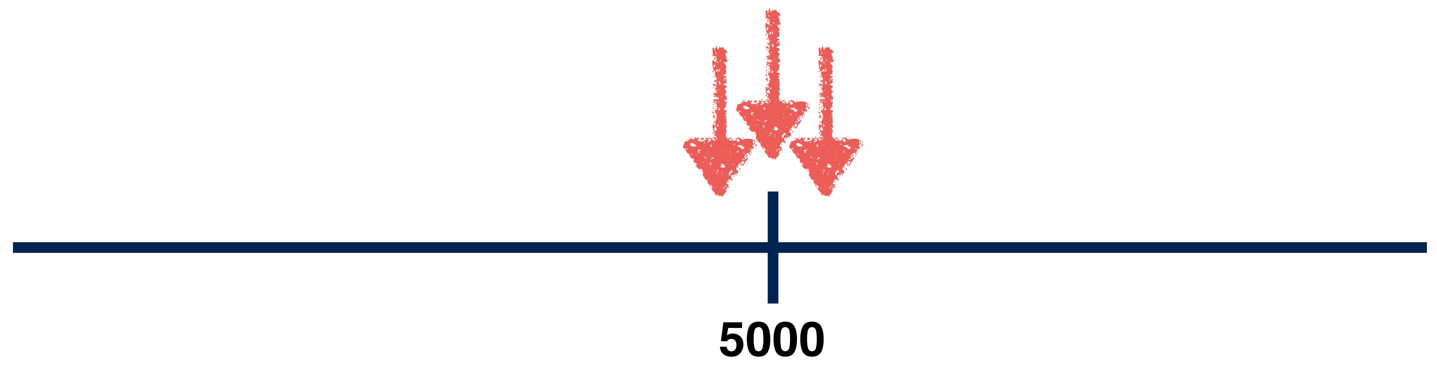
Find Business Conditions



money is over 5000 cannot be transferred

REQ 1

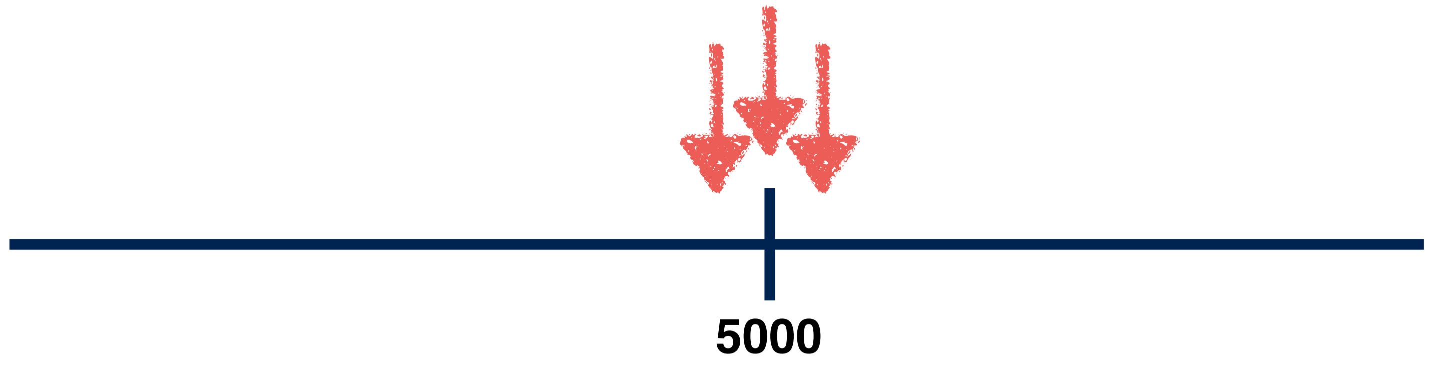
Find Business Conditions



money is over 5000 cannot be transferred

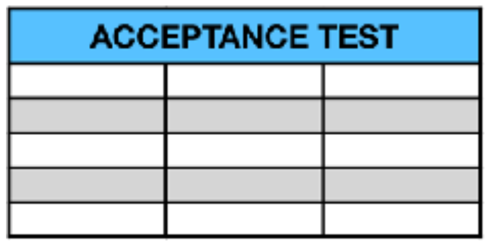
REQ 1

Find Business Conditions

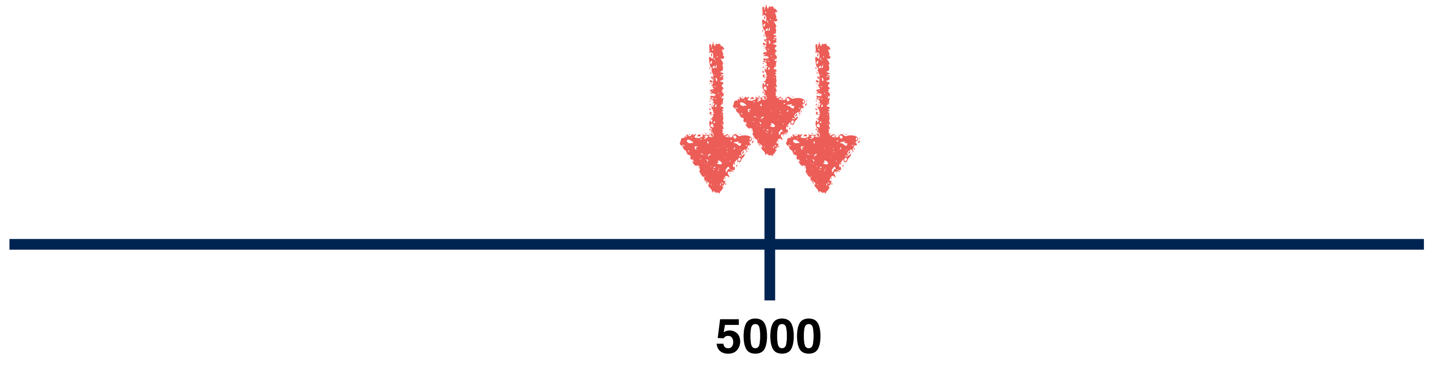


money is over 5000 cannot be transferred

REQ 1



Test Design



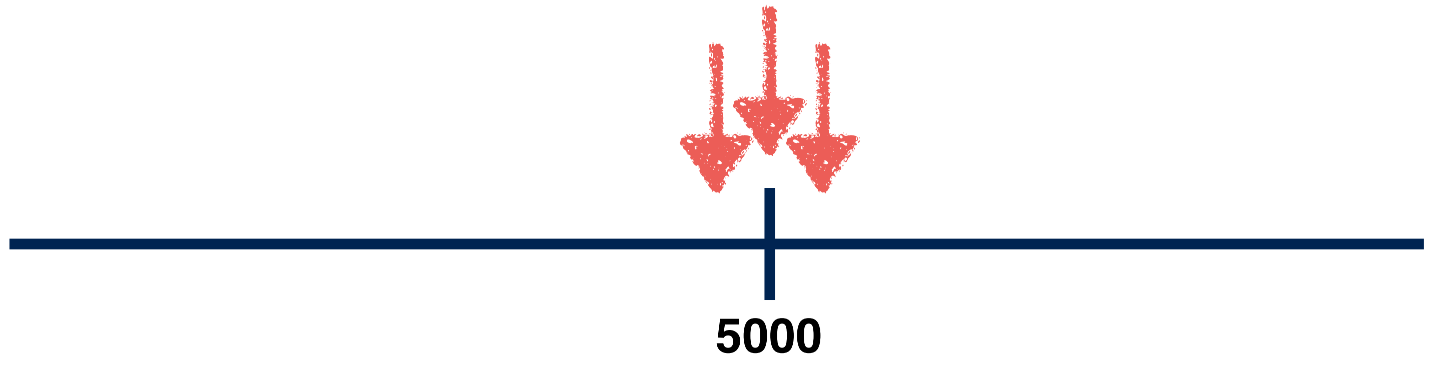
money is over 5000 cannot be transferred

REQ 1

Case	Condition		Expected		
		message	balance before	amount of money	balance after
1	Less than 5000	transferable	balance	money	balance - money
2	equal 5000	transferable	balance	money	balance - money
3	more than 5000	can not be transfer			

ACCEPTANCE TEST

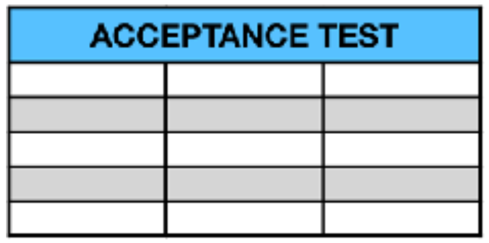
Test Design



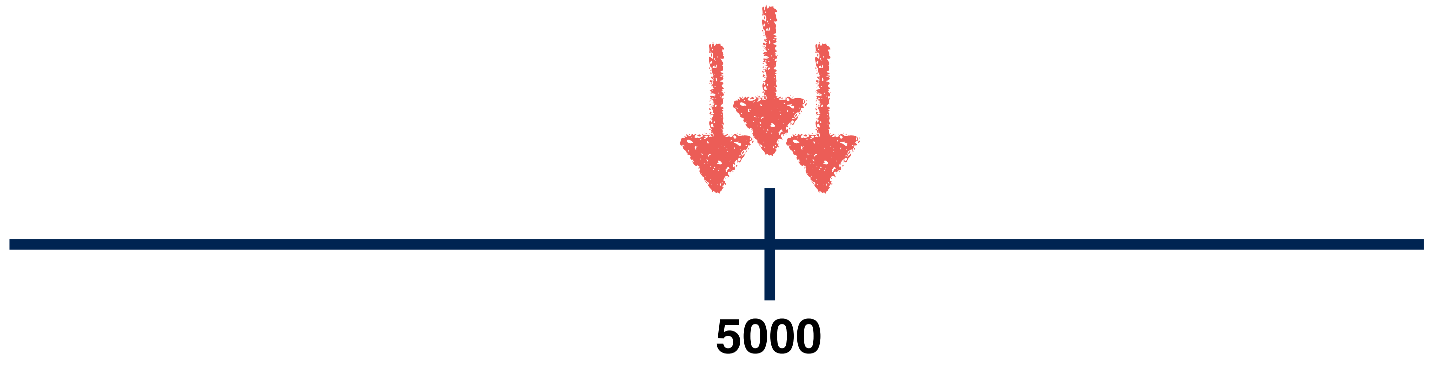
money is over 5000 cannot be transferred

REQ 1

Case	Condition	Data			Expected		
		account info	amount of money	message	balance before	amount of money	balance after
1	Less than 5000	{ balance:5000, id: "122012689", name:"Apipol Sukgler" }	4900	transferable	5000	4900	100
2	equal 5000	{ balance:5000, id: "122014627", name:"Chonnikan Toboonlarng" }	5000	transferable	5000	5000	0
3	more than 5000	{ balance:20000, id: "122014627", name:"Somkiat Puisungnoen" }	5500	can not be transfer			



Test Design



money is over 5000 cannot be transferred

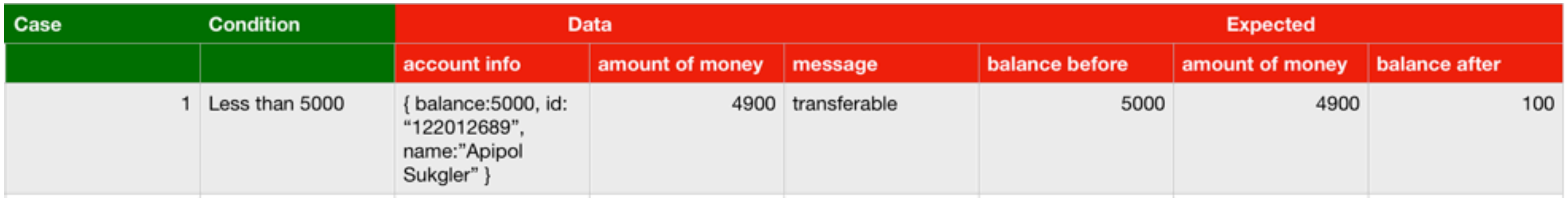
REQ 1

Case	Condition	Data			Expected		
		account info	amount of money	message	balance before	amount of money	balance after
1	Less than 5000	{ balance:5000, id: "122012689", name:"Apipol Sukgler" }	4900	transferable	5000	4900	100
2	equal 5000	{ balance:5000, id: "122014627", name:"Chonnikan Toboonlarng" }	5000	transferable	5000	5000	0
3	more than 5000	{ balance:20000, id: "122014627", name:"Somkiat Puisungnoen" }	5500	can not be transfer			

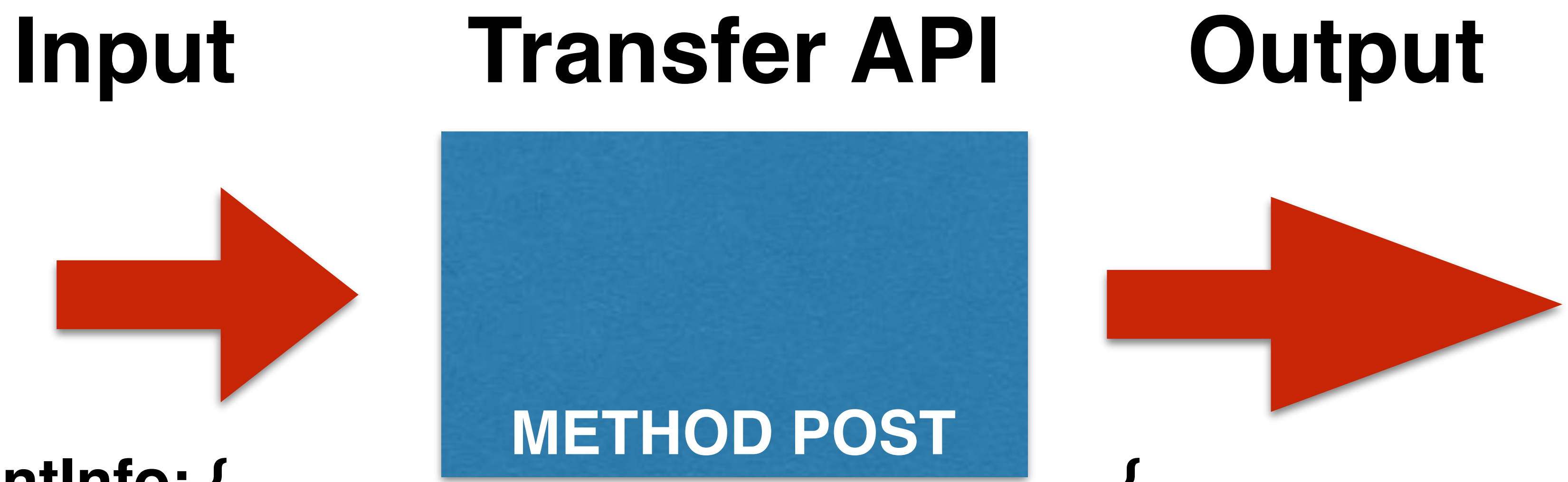
API Design

Case	Condition	Data			Expected		
		account info	amount of money	message	balance before	amount of money	balance after
1	Less than 5000	{ balance:5000, id: “122012689”, name:”Apipol Sukgler” }	4900	transferable	5000	4900	100

API Design



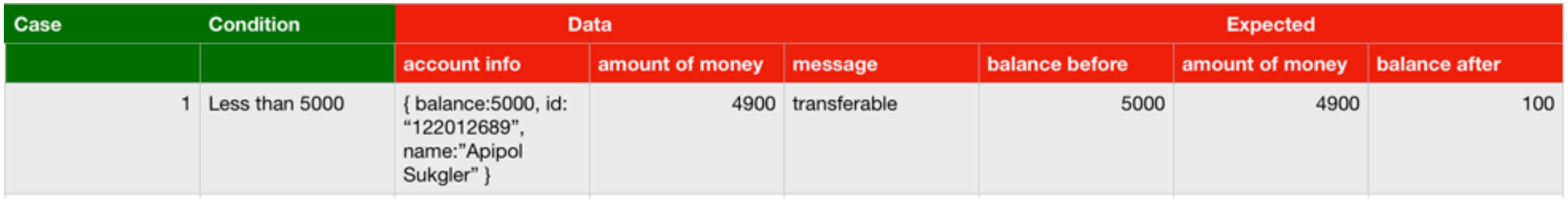
Case	Condition	Data			Expected		
		account info	amount of money	message	balance before	amount of money	balance after
1	Less than 5000	{ balance:5000, id: “122012689”, name:”Apipol Sukgler” }	4900	transferable	5000	4900	100



```
{
    accountInfo: {
        balance: 5000,
        id: “122012689”,
        name: “Apipol Sukgler”
    },
    amountOfMoney: 4900
}
```

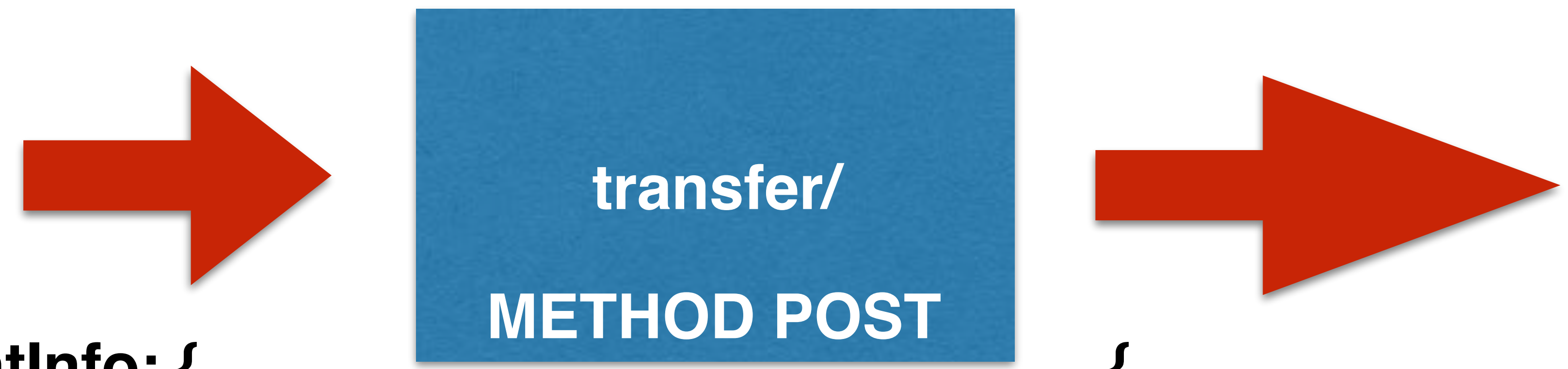
```
{
    status: “transferable”,
    balanceBefore: 5000,
    amountOfMoney: 4900,
    balanceAfter: 100
}
```

API Design



Case	Condition	Data			Expected		
		account info	amount of money	message	balance before	amount of money	balance after
1	Less than 5000	{ balance:5000, id: “122012689”, name:”Apipol Sukgler” }	4900	transferable	5000	4900	100

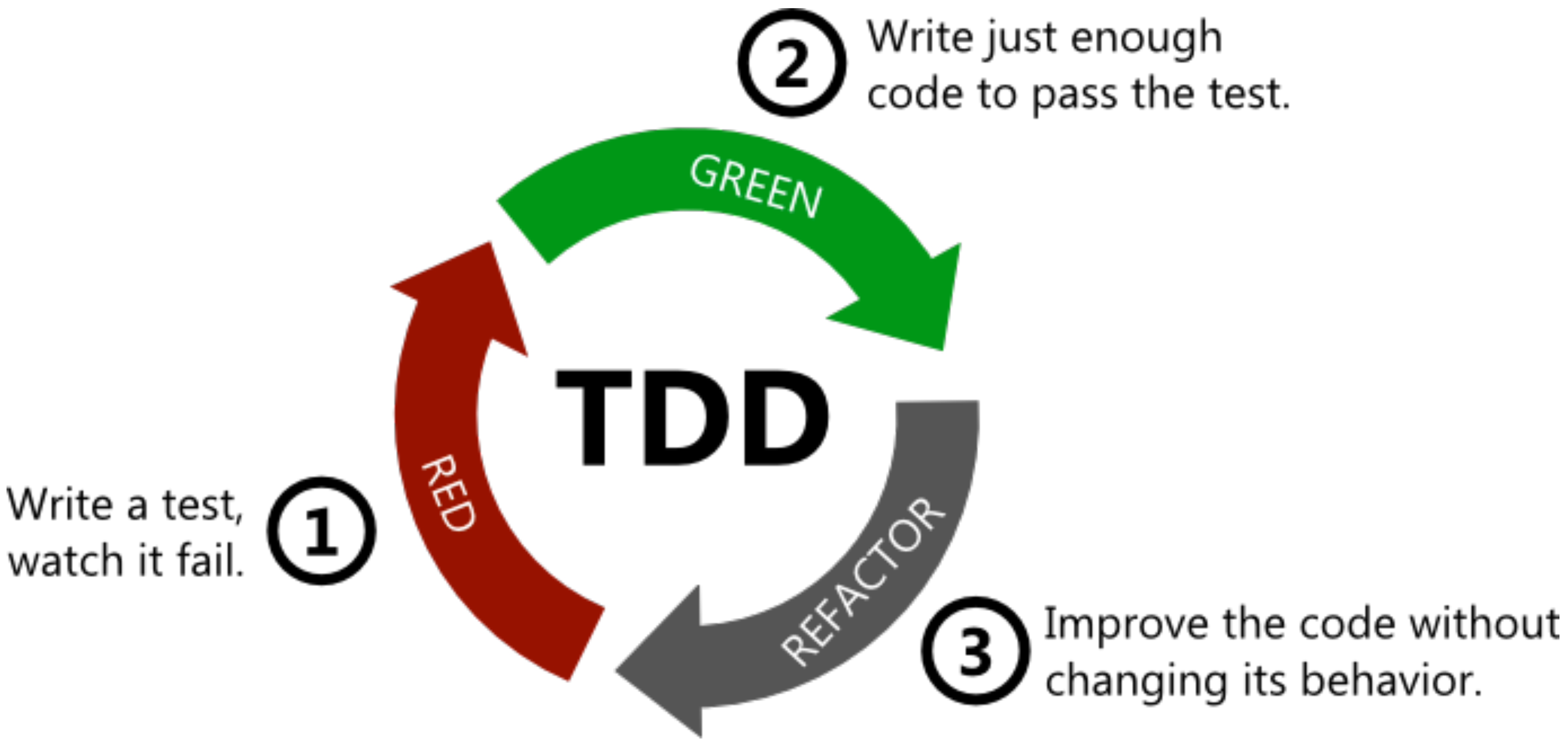
Input **Transfer API** **Output**



```
{
    accountInfo: {
        balance: 5000,
        id: “122012689”,
        name: “Apipol Sukgler”
    },
    amountOfMoney: 4900
}
```

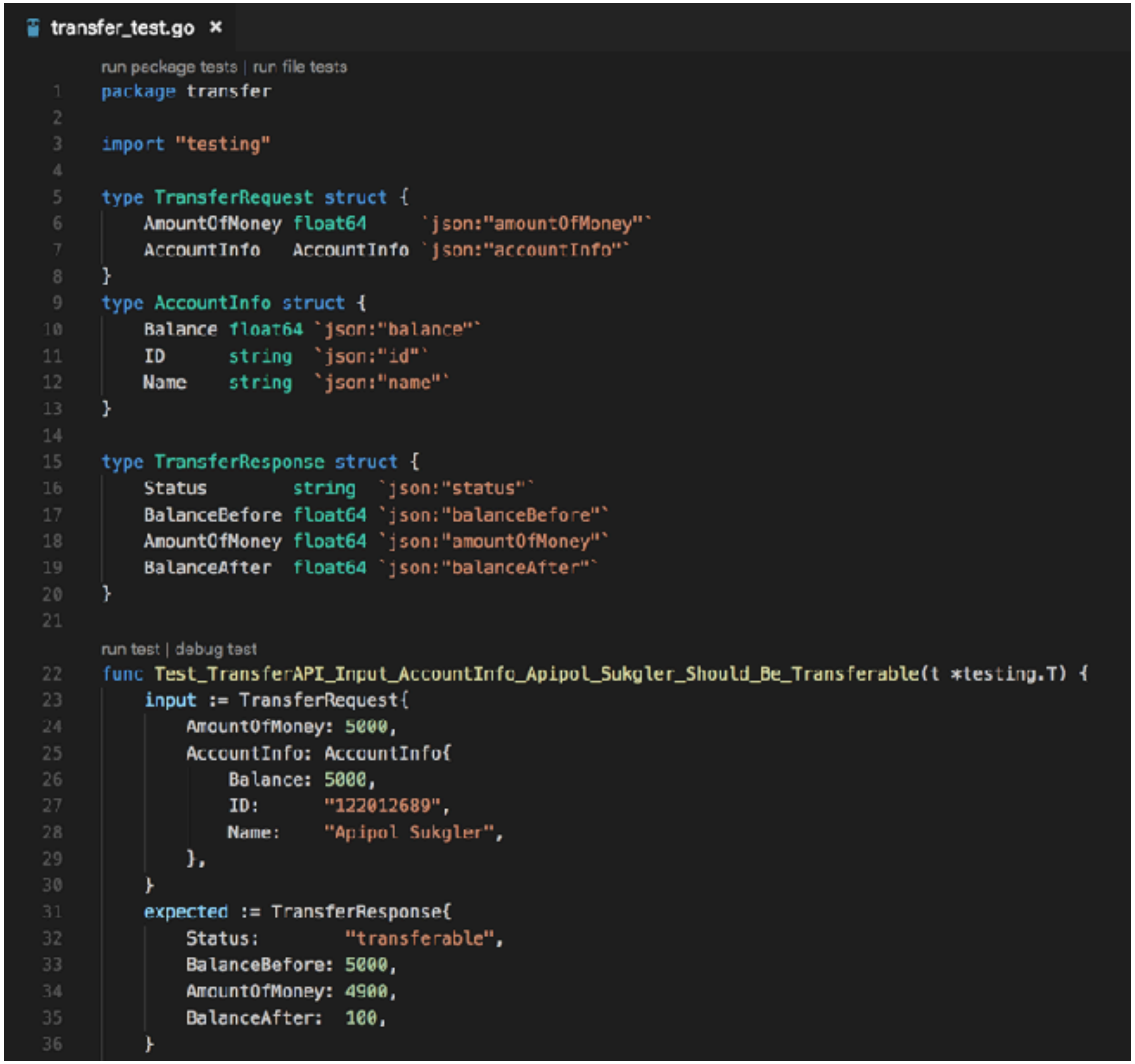
```
{
    status: “transferable”,
    balanceBefore: 5000,
    amountOfMoney: 4900,
    balanceAfter: 100
}
```

Test-Driven Development



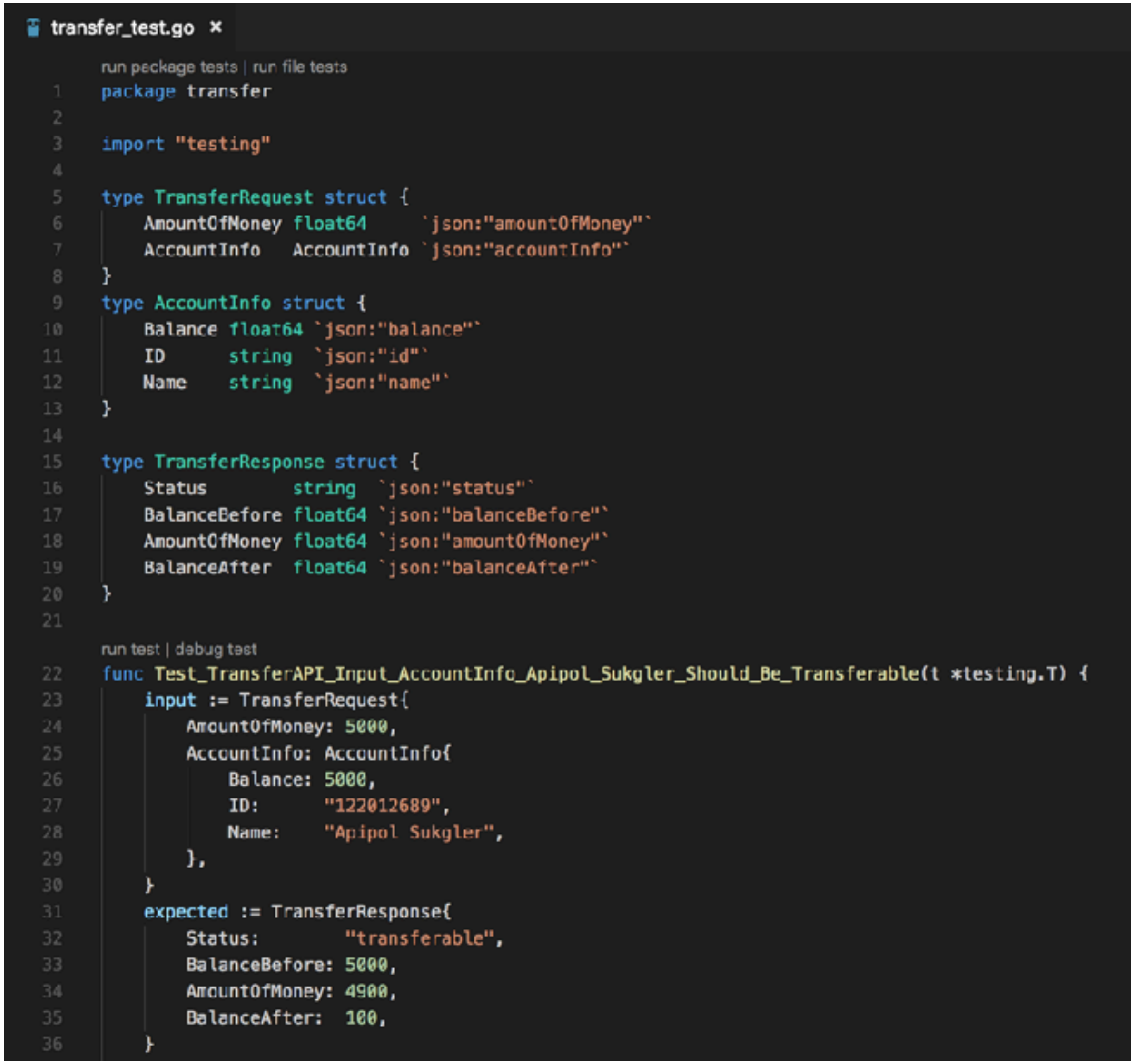
2 Write just enough code to pass the test.
GREEN
TDD
RED
REFACTOR
Write a test, watch it fail. 1
3 Improve the code without changing its behavior.

case 1 Less than 5000



```go
package transfer

import "testing"

type TransferRequest struct {
	AmountOfMoney float64     `json:"amountOfMoney"`
	AccountInfo   AccountInfo `json:"accountInfo"`
}
type AccountInfo struct {
	Balance float64 `json:"balance"`
	ID      string  `json:"id"`
	Name    string  `json:"name"`
}

type TransferResponse struct {
	Status        string  `json:"status"`
	BalanceBefore float64 `json:"balanceBefore"`
	AmountOfMoney float64 `json:"amountOfMoney"`
	BalanceAfter  float64 `json:"balanceAfter"`
}

func Test_TransferAPI_Input_AccountInfo_Apipol_Sukgler_Should_Be_Transferable(t *testing.T) {
	input := TransferRequest{
		AmountOfMoney: 5000,
		AccountInfo: AccountInfo{
			Balance: 5000,
			ID:      "122012689",
			Name:    "Apipol Sukgler",
		},
	}
	expected := TransferResponse{
		Status:        "transferable",
		BalanceBefore: 5000,
		AmountOfMoney: 4900,
		BalanceAfter:  100,
	}
```

case 1 Less than 5000



transfer_test.go ×

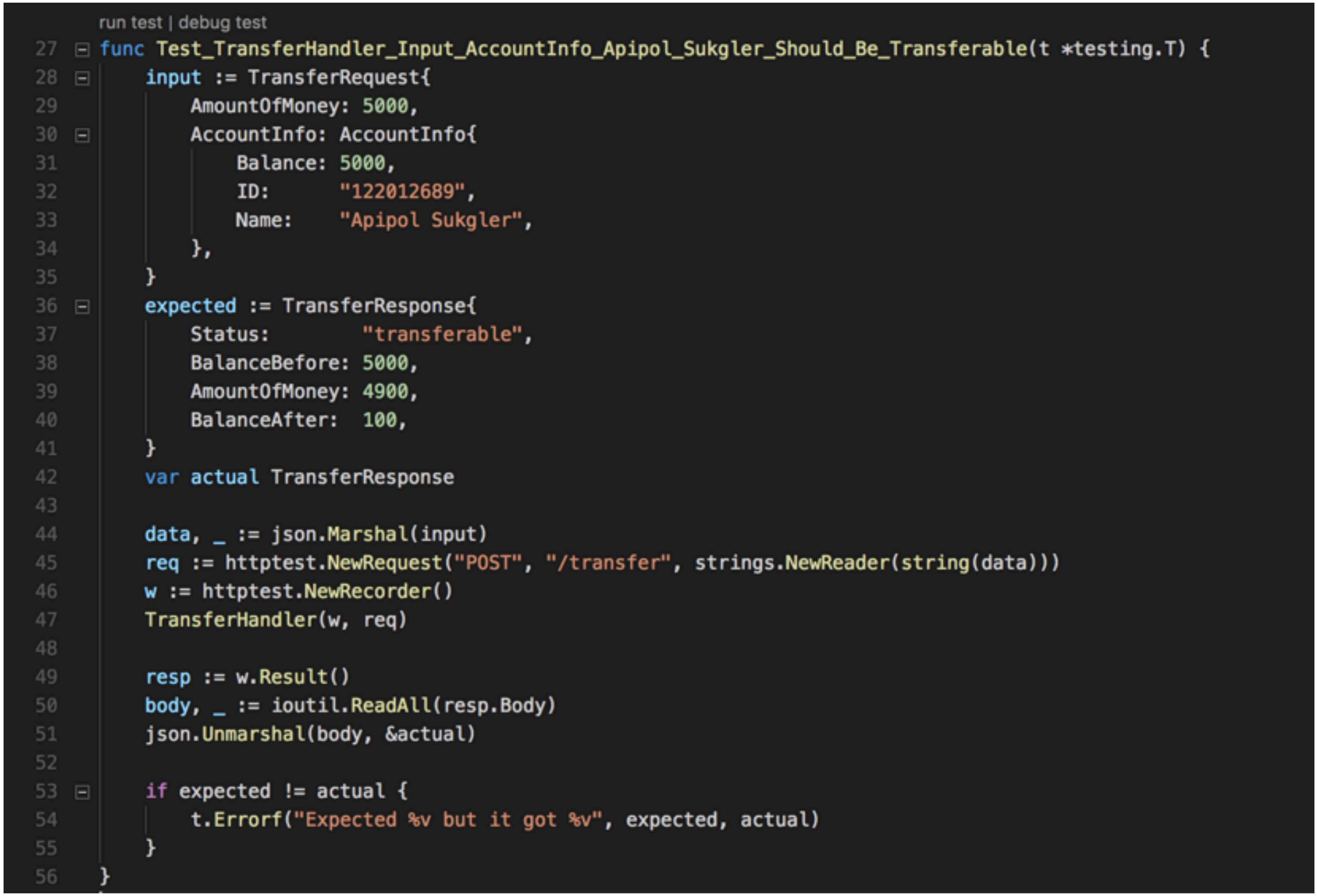
run package tests | run file tests

```go
package transfer

import "testing"

type TransferRequest struct {
	AmountOfMoney float64     `json:"amountOfMoney"`
	AccountInfo   AccountInfo `json:"accountInfo"`
}
type AccountInfo struct {
	Balance float64 `json:"balance"`
	ID      string  `json:"id"`
	Name    string  `json:"name"`
}

type TransferResponse struct {
	Status        string  `json:"status"`
	BalanceBefore float64 `json:"balanceBefore"`
	AmountOfMoney float64 `json:"amountOfMoney"`
	BalanceAfter  float64 `json:"balanceAfter"`
}

```

run test | debug test

```go
func Test_TransferAPI_Input_AccountInfo_Apipol_Sukgler_Should_Be_Transferable(t *testing.T) {
	input := TransferRequest{
		AmountOfMoney: 5000,
		AccountInfo: AccountInfo{
			Balance: 5000,
			ID:      "122012689",
			Name:    "Apipol Sukgler",
		},
	}
	expected := TransferResponse{
		Status:        "transferable",
		BalanceBefore: 5000,
		AmountOfMoney: 4900,
		BalanceAfter:  100,
	}
```

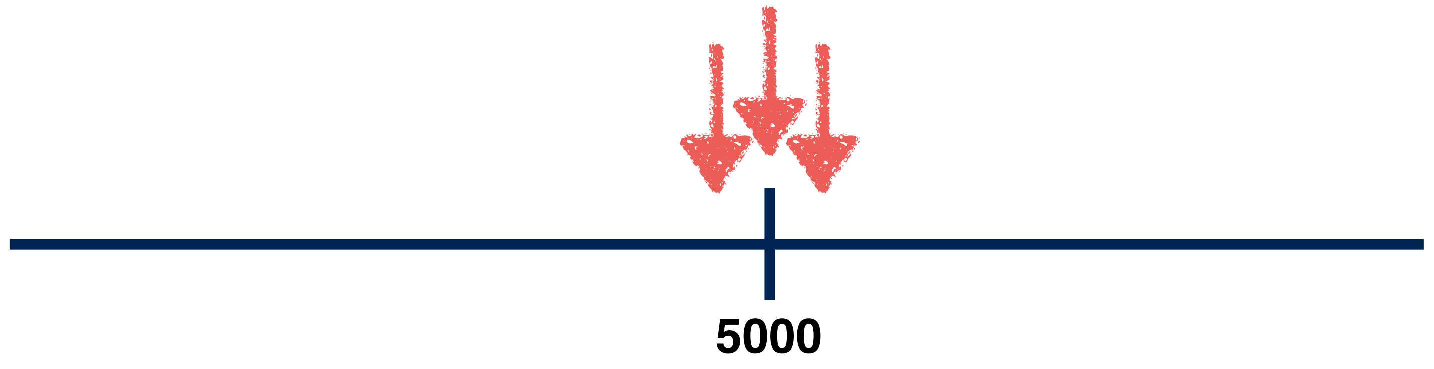
case 1 Less than 5000



```
run test | debug test
func Test_TransferHandler_Input_AccountInfo_Apipol_Sukgler_Should_Be_Transferable(t *testing.T) {
	input := TransferRequest{
		AmountOfMoney: 5000,
		AccountInfo: AccountInfo{
			Balance: 5000,
			ID:      "122012689",
			Name:    "Apipol Sukgler",
		},
	}
	expected := TransferResponse{
		Status:        "transferable",
		BalanceBefore: 5000,
		AmountOfMoney: 4900,
		BalanceAfter:  100,
	}
	var actual TransferResponse

	data, _ := json.Marshal(input)
	req := httptest.NewRequest("POST", "/transfer", strings.NewReader(string(data)))
	w := httptest.NewRecorder()
	TransferHandler(w, req)

	resp := w.Result()
	body, _ := ioutil.ReadAll(resp.Body)
	json.Unmarshal(body, &actual)

	if expected != actual {
		t.Errorf("Expected %v but it got %v", expected, actual)
	}
}
```

ACCEPTANCE TEST

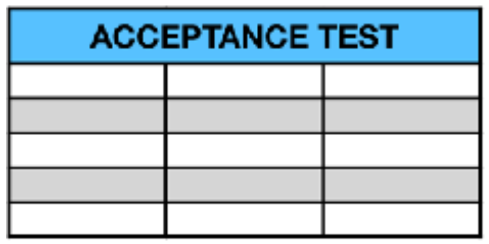
Test Design



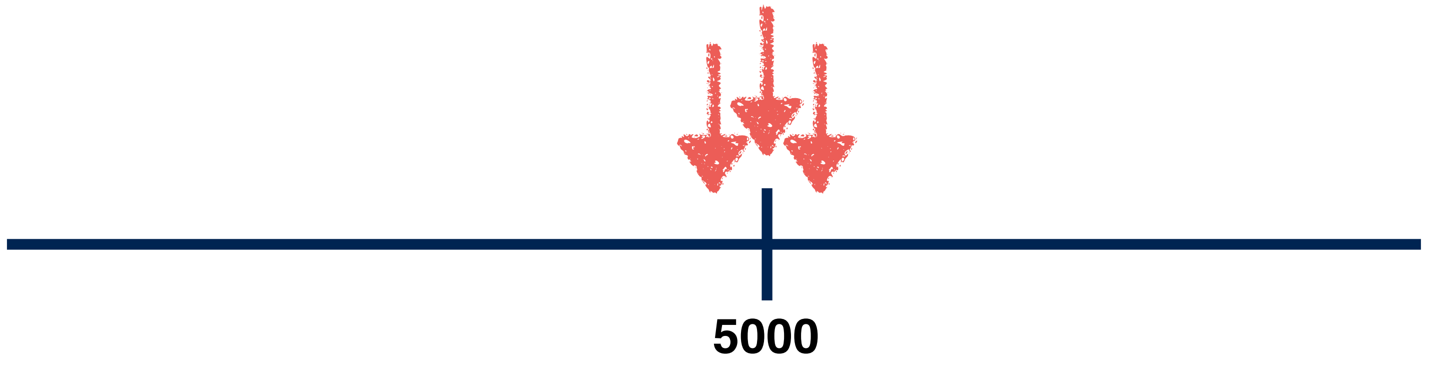
money is over 5000 cannot be transferred

REQ 1

Case	Condition	Data		Expected			
		account info	amount of money	message	balance before	amount of money	balance after
1	Less than 5000	{ balance:5000, id: “122012689”, name:”Apipol Sukgler” }	4900	transferable	5000	4900	100
➡ 2	equal 5000	{ balance:5000, id: “122014627”, name:”Chonnikan Toboonlarng” }	5000	transferable	5000	5000	0
3	more than 5000	{ balance:20000, id: “122014627”, name:”Somkiat Puisungnoen” }	5500	can not be transfer			



Test Design



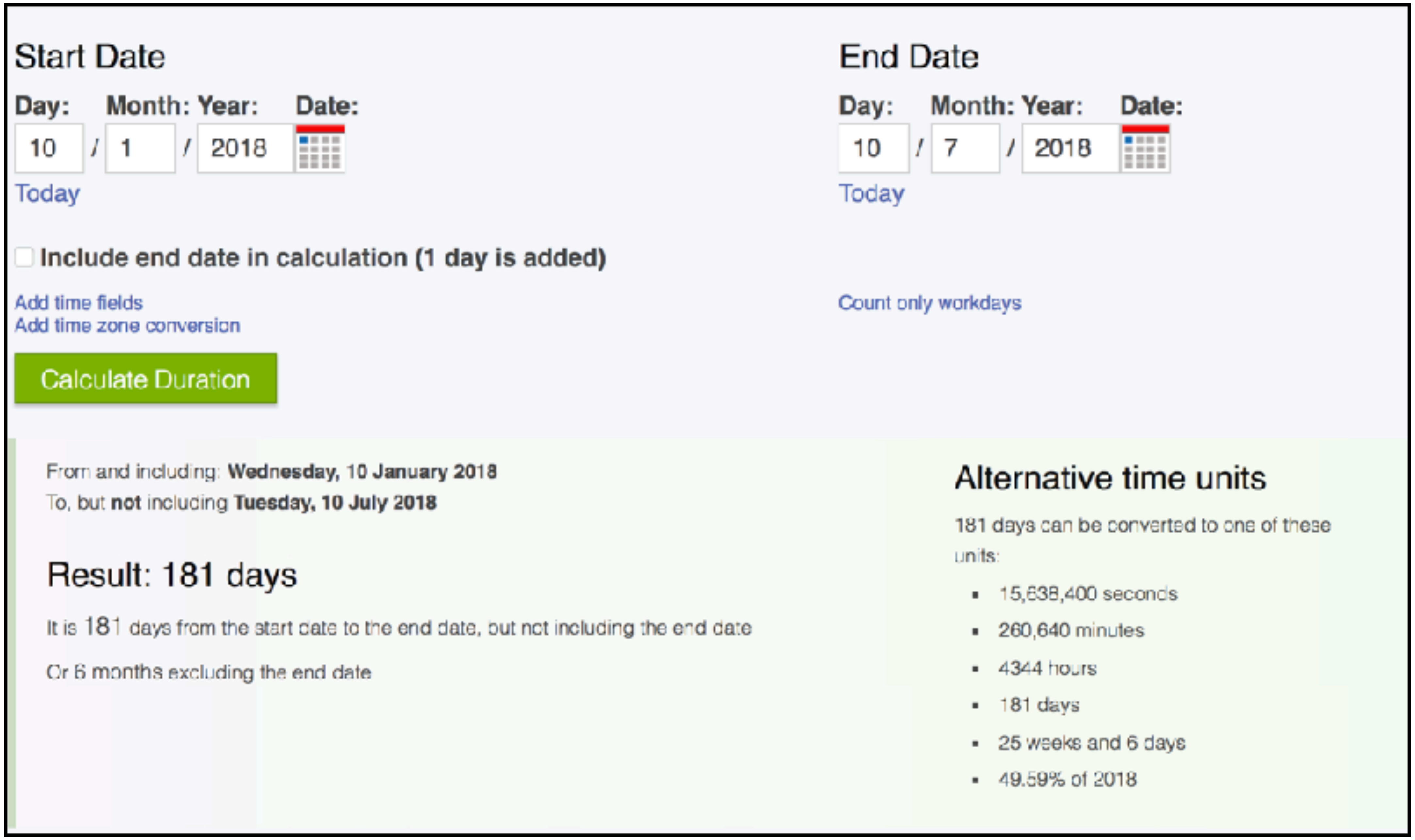
money is over 5000 cannot be transferred

REQ 1

Case	Condition	Data			Expected		
		account info	amount of money	message	balance before	amount of money	balance after
1	Less than 5000	{ balance:5000, id: "122012689", name:"Apipol Sukgler" }	4900	transferable	5000	4900	100
2	equal 5000	{ balance:5000, id: "122014627", name:"Chonnikan Toboonlarng" }	5000	transferable	5000	5000	0
3	more than 5000	{ balance:20000, id: "122014627", name:"Somkiat Puisungnoen" }	5500	can not be transfer			

Q & A

Start with Problem



Start Date

Day:	Month:	Year:	Date:
10	1	2018	

Today

End Date

Day:	Month:	Year:	Date:
10	7	2018	

Today

☐ **Include end date in calculation (1 day is added)**

Add time fields
Add time zone conversion

Count only workdays

From and including: **Wednesday, 10 January 2018**
To, but **not** including **Tuesday, 10 July 2018**

Result: 181 days

It is 181 days from the start date to the end date, but not including the end date

Or 6 months excluding the end date

Alternative time units

181 days can be converted to one of these units:

- 15,638,400 seconds
- 260,640 minutes
- 4344 hours
- 181 days
- 25 weeks and 6 days
- 49.59% of 2018