go get better

framework training

business value through education

3 Day Go Training Course

framework training

business value through education

Dom Davis

@idomdavis


Hello
my name is

Deliberate Practice

framework training

House Rules

There is no such thing as a stupid question

We go at your pace

This is your course

This is not school

Have fun

Go

framework training

business value through education

Simple, orthogonal features, that must carry their weight
Prefer maintainability over expressiveness
Typing is not so difficult

Gophers

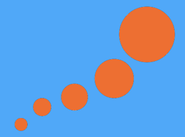
framework training

business value through education

The Plan

Introduction, Setup & Go Katas
Go Idiosyncrasies
Types, Interfaces, Polymorphism & Generics
Concurrency and Parallelism
Networking
Packaging and Distribution
Go Nuts

The Basics



framework training

```
package main

func main() {
    var s string = "Hello"
    println(s)
}
```

```go
package main

func main() {
    var s1 string = "Hello"
    var s2 = "World"

    println(s1, s2)
}
```

```go
package main

func main() {
    s1, s2 := "Hello", "World"
    println(s1, s2)
}
```

```go
package main

var (
    s1 = "Hello"
    s2 = "World"
)

func main() {
    println(s1, s2)
}
```

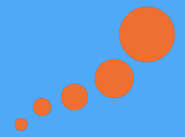
bool
byte
uintptr
int, int8, int16, int32, int64
uint, uint8, uint16, uint32, uint64
float32, float64
complex64, complex128
rune, string

```
func branching(a string) {
    if a == "A" {
        println("Good")
    } else if a == "B" {
        println("Close enough")
    } else {
        println("Nope!")
    }
}
```

```
func branching(a string) {
    if a, b := 1, 2; a == b {
        println(a, "equal to", b)
    } else {
        println(a, "not equal to", b)
    }

    // a, b are undefined here
}
```

```
func loops() {
    for i := 0; i < 100; i++ {
        println(i)
    }
}
```

FizzBuzz

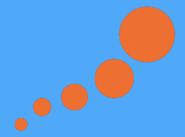


framework training

1, 2, Fizz, 4, Buzz, Fizz, 7, 8, 9, Buzz, 11, Fizz, 13, 14, FizzBuzz

If a number is divisible by 3 output Fizz
If a number is divisible by 5 output Buzz
If a number is divisible by 3 and 5 output FizzBuzz
Otherwise output the number

go get setup



framework training

1: FizzBuzz

For the numbers 1 to 100
If a number is divisible by 3 output Fizz
If a number is divisible by 5 output Buzz
If a number is divisible by 3 and 5 output FizzBuzz
Otherwise output the number

https://github.com/domdavis/go-get-better/tree/exercise-1

```
func main() {
    for i := 1; i <= 100; i++ {
        if i%15 == 1 {
            rand.Seed(int64(176064004))
        }
        fmt.Print([]string{fmt.Sprintf("%d\n", i),
            "fizz", "buzz", "fizzbuzz"}[rand.Int63()%4])
    }
}
```

```go
package main

import "fmt"

func main() {
    for i := 1; i <= 100; i++ {
        if i%3 == 0 && i%5 == 0 {
            fmt.Printf("%-5d - %s\n", i, "FizzBuzz")
        } else if i%3 == 0 {
            fmt.Printf("%-5d - %s\n", i, "Fizz")
        } else if i%5 == 0 {
            fmt.Printf("%-5d - %s\n", i, "Buzz")
        } else {
            fmt.Println(i)
        }
    }
}
```

EnterpriseQualityCoding / FizzBuzzEnterpriseEdition

Watch 166 | Star 8,086 | Fork 424

Code | Issues 184 | Pull requests 22 | Projects 0 | Wiki | Insights

FizzBuzz Enterprise Edition is a no-nonsense implementation of FizzBuzz made by serious businessmen for serious business purposes. http://www.fizzbuzz.enterprises

161 commits | **1** branch | **0** releases | **30** contributors

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

emiln committed on **GitHub** Merge pull request #281 from tkellogg/master ... Latest commit `cdfac75` on 4 Feb

gradle/wrapper	Include Gradle config closes #230	2 years ago
resources/assets/configuration/sp...	Installed spring and restructured for dependency injection.	3 years ago
src	FizzStringReturner may not have copied all characters	8 months ago
.gitattributes	Started the project. Time to learn!	5 years ago
.gitignore	Add unsupported platform files to the .gitignore	2 years ago
.travis.yml	Reverting the revert commit, since it clearly did not revert as inten...	2 years ago
CONTRIBUTING.md	Reverting the revert commit, since it clearly did not revert as inten...	2 years ago
README.md	Reverting the revert commit, since it clearly did not revert as inten...	2 years ago
build.gradle	Include Gradle config closes #230	2 years ago
gradlew	Include Gradle config closes #230	2 years ago
gradlew.bat	Include Gradle config closes #230	2 years ago
pom.xml	Merge pull request #232 from kristianperkins/patch-1	a year ago
settings.gradle	Include Gradle config closes #230	2 years ago

```
func main() {
    pattern :=
        []int{0, 0, 1, 0, 2, 1, 0, 0, 1, 2, 0, 1, 0, 0, 3}

    for i := 1; i <= 100; i++ {
        fmt.Print([]string{fmt.Sprintf("%d\n", i),
            "fizz\n", "buzz\n", "fizzbuzz\n"}[pattern[(i-1)%15]])
    }
}
```

```go
package main

import "fmt"

func main() {
    for i := 1; i <= 100; i++ {
        if i%3 == 0 && i%5 == 0 {
            fmt.Printf("%-5d - %s\n", i, "FizzBuzz")
        } else if i%3 == 0 {
            fmt.Printf("%-5d - %s\n", i, "Fizz")
        } else if i%5 == 0 {
            fmt.Printf("%-5d - %s\n", i, "Buzz")
        } else {
            fmt.Println(i)
        }
    }
}
```

```go
package main

import "fmt"

func main() {
    for i := 1; i <= 100; i++ {
        switch {
        case i%3 == 0 && i%5 == 0:
            fmt.Printf("%-5d - %s\n", i, "FizzBuzz")
        case i%3 == 0:
            fmt.Printf("%-5d - %s\n", i, "Fizz")
        case i%5 == 0:
            fmt.Printf("%-5d - %s\n", i, "Buzz")
        default:
            fmt.Println(i)
        }
    }
}
```

switch

framework training

```
func branching(i int) {
    switch i {
    case 1:
        println("One")
    case 2:
        println("A couple")
    default:
        println("Many")
    }
}
```

```
func branching(i int) {
    switch i {
    case 1:
        println("One")
    case 2, 3, 4, 5:
        println("Some")
    default:
        println("Many")
    }
}
```

```
func branching(i int) {
    switch {
    case i == 1:
        println("One")
    case i <= 5:
        println("Some")
    default:
        println("Many")
    }
}
```

```
func branching(i int) {
    switch i {
    case 1:
        println("One")
        fallthrough
    case 2:
        println("Two")
    case 3:
        println("Three")
    default:
        println("Many")
    }
}
```

Go Is Opinionated

Go is not a language to express your inner artist
There is [almost] always one idiomatic way to do something
There is only one way to format your code: `go fmt`

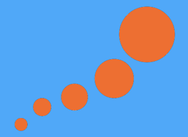
```go
package main

import "fmt"

func main() {
    for i := 1; i <= 100; i++ {
        switch {
        case i%3 == 0 && i%5 == 0:
            fmt.Printf("%-5d - %s\n", i, "FizzBuzz")
        case i%3 == 0:
            fmt.Printf("%-5d - %s\n", i, "Fizz")
        case i%5 == 0:
            fmt.Printf("%-5d - %s\n", i, "Buzz")
        default:
            fmt.Println(i)
        }
    }
}
```

go vet
goimports
golint

```
[go-get-better (exercise-1)]$ go vet
# github.com/domdavis/go-get-better
./alternatives_test.go:12: ExampleIdiomaticSolution refers to unknown identifier: IdiomaticSolution
./alternatives_test.go:34: ExampleBrittleSolution refers to unknown identifier: BrittleSolution
./alternatives_test.go:53: ExampleCompactSolution refers to unknown identifier: CompactSolution

[go-get-better (exercise-1)]$ golint
alternatives_test.go:68:9: if block ends with a return statement, so drop this else and outdent
its block (move short variable declaration to its own line if necessary)

[go-get-better (exercise-1)]$
```

golangci-lint

func



framework training

```
package main

func main() {
    println("Hello, World!")
}
```

```go
package main

func main() {
    hello()
}

func hello() {
    println("Hello, World!")
}
```

```go
package main

import "fmt"

func main() {
    hello("World!")
}

func hello(n string) {
    fmt.Printf("Hello, %s", n)
}
```

```
package main

func main() {
    println(square(3))
}

func square(i int) int {
    return i * i
}
```

```go
package main

func main() {
    i, r := divide(5, 2)
    println(i, r)
}

func divide(x, y int) (int, int) {
    i := x / y
    r := x % y

    return i, r
}
```

```go
package main

func main() {
    i, r := divide(5, 2)
    println(i, r)
}

func divide(x, y int) (i int, r int) {
    i = x / y
    r = x % y

    return
}
```

```go
package main

func main() {
    i, r := divide(5, 2)
    println(i, r)
}

func divide(x, y int) (i int, r int) {
    i = x / y
    r = x % y

    return i, r
}
```

```go
package main

func main() {
    i, r := divide(5, 2)
    println(i, r)
}

func divide(x, y int) (i, r int) {
    i = x / y
    r = x % y

    return i, r
}
```

```go
package main

func main() {
    i, r := divide(5, 2)
    println(i, r)
}

func divide(x, y int) (i, r int) {
    i, r = x/y, x%y
    return i, r
}
```

```go
package main

func main() {
    i, r := divide(5, 2)
    println(i, r)
}

// Divide x by y, returning the integer part, i,
// and the remainder, r.
func divide(x, y int) (i, r int) {
    return x/y, x%y
}
```

```go
package main

func main() {
    i, r := divide(5, 0)
    println(i, r)
}

// Divide x by y, returning the integer part, i,
// and the remainder, r.
func divide(x, y int) (i, r int) {
    return x/y, x%y
}
```

2: FizzBuzz(int)

Convert your solution from Exercise 1 into a function with the signature:
func FizzBuzz(int n)

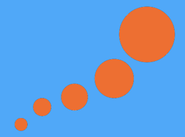
The function should perform the following actions for the numbers 1 to n:
* If a number is divisible by 3 output Fizz
* If a number is divisible by 5 output Buzz
* If a number is divisible by 3 and 5 output FizzBuzz
* Otherwise, output the number

Write an Example style test to check the code.

https://github.com/domdavis/go-get-better/tree/exercise-2

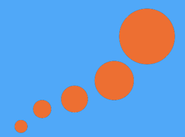
```
func main() {
	for i := 1; i <= 19; i++ {
		var out string
		if i%3 == 0 {
			out = "Fizz"
		}

		if i%5 == 0 {
			out += "Buzz"
		}

		if i%3 != 0 && i%5 != 0 {
			out = strconv.Itoa(i)
		}
		fmt.Println(out)
	}
}
```

Packages and Modules



framework training

Modules

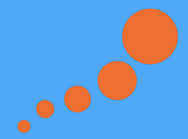


framework training

```
$ mkdir training
$ cd training
$ go mod init training
```

```
$ mkdir training
$ cd training
$ go mod init github.com/domdavis/training
```

Packages



framework training

```go
package training

import ...

func FizzBuzz(n int) string {
    var b strings.Builder
    for i := 1; i <= n; i++ {
        if i%3 == 0 {
            b.WriteString("Fizz")
        }

        if i%5 == 0 {
            b.WriteString("Buzz")
        }

        if i%3 != 0 && i%5 != 0 {
            b.WriteString(strconv.Itoa(i))
        }

        b.WriteString(" ")
    }

    return strings.TrimSpace(b.String())
}
```

```
[exercise3]$ touch fizzbuzz_test.go
```

```go
package training_test

import (
    "fmt"
    "training"
)

func ExampleFizzBuzz() {
    fmt.Println(training.FizzBuzz(15))

    // Output:
    // 1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz
}
```

```
[exercise3] $  go test
PASS
ok      training     0.004s
```

```go
package training_test

import (
    "testing"
    "training"
)

func TestFizzBuzz(t *testing.T) {
    r := training.FizzBuzz(-1)
    if r != "" {
        t.Errorf("unexpected FizzBuzz sequence: %s", r)
    }
}
```

```
$ go test -v --covermode=count
=== RUN   TestFizzBuzz
--- PASS: TestFizzBuzz (0.00s)
=== RUN   ExampleFizzBuzz
--- PASS: ExampleFizzBuzz (0.00s)
PASS
coverage: 100.0% of statements
ok      training     0.004s
```

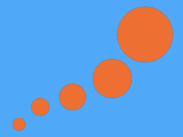
3: FizzBuzz()

Implement `FizzBuzz(n int) string`
Write one or more tests to exercise the function

Run your tests using `go test`
https://github.com/domdavis/go-get-better/tree/exercise-3

```go
package calculator

/// Divide x by y, returning the integer part, i,
// and the remainder, r.
func Divide(x, y int) (i, r int) {
    return x/y, x%y
}
```

Error Handling



framework training

```go
package calculator

import (
    "errors"
    "fmt"
)

func Divide(x, y int) (int, int, error) {
    if y == 0 {
        return 0, 0, errors.New("divide by zero error")
    }

    return x/y, x%y, nil
}
```

```go
func ExampleDivide() {
    i, r, err := calculator.Divide(5, 2)

    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println(i, r)
    }

    // Output:
    // 2 1
}
```

```go
func ExampleDivide() {
    if i, r, err := calculator.Divide(5, 2); err != nil {
        fmt.Println(err)
    } else {
        fmt.Println(i, r)
    }

    // Output:
    // 2 1
}
```

"Values can be programmed, and since errors are values, errors can be programmed."

https://blog.golang.org/errors-are-values

```go
func main() {
    var b strings.Builder
    scanner := bufio.NewScanner(os.Stdin)
    for i := 0; i <= 4; i++ {
        scanner.Scan()
        b.WriteString(scanner.Text())
    }

    if err := scanner.Err(); err != nil {
        fmt.Println(err)
    }

    fmt.Println(b.String())
}
```

```go
func main() {
    var b strings.Builder
    scanner := bufio.NewScanner(os.Stdin)
    for i := 0; i <= 4; i++ {
        if err := scanner.Scan(); err != nil {
            fmt.Println(err)
        } else {
            b.WriteString(scanner.Text())
        }
    }

    fmt.Println(b.String())
}
```

```
var (
    ErrCustomError = errors.New("some custom error")
    ErrDivideByZero = errors.New("divide by zero")
)
```

```go
package calculator

import (
    "errors"
    "fmt"
)

var ErrDivideByZero = errors.New("divide by zero")

func Divide(x, y int) (int, int, error) {
    if y == 0 {
        return 0, 0, ErrDivideByZero
    }

    return x/y, x%y, nil
}
```

```go
func ExampleDivide() {
    i, r, err := calculator.Divide(5, 2)

    if errors.Is(err, calculator.ErrDivideByZero) {
        fmt.Println(err)
    } else {
        fmt.Println(i, r)
    }

    // Output:
    // 2 1
}
```

```go
func ExampleDivide() {
    i, r, err := calculator.Divide(5, 2)

    if errors.Is(err, calculator.ErrDivideByZero) {
        fmt.Println(err)
    } else if err != nil {
        fmt.Println("Something went wrong!")
    } else {
        fmt.Println(i, r)
    }

    // Output:
    // 2 1
}
```

```go
func ExampleDivide() {
    i, r, err := calculator.Divide(5, 2)

    switch {
    case errors.Is(err, calculator.ErrDivideByZero):
        fmt.Println(err)
    case err != nil:
        fmt.Println("Something went wrong!")
    default:
        fmt.Println(i, r)
    }

    // Output:
    // 2 1
}
```

```go
func Action() error {
    i, r, err := calculator.Divide(5, 2)

    switch err {
    case nil:
    case errors.Is(err, calculator.ErrDivideByZero):
        return err
    default:
        log.Fatal(err)
    }

    fmt.Println(i, r)
    return nil
}
```

4: `FizzBuzz(int) (string, error)`

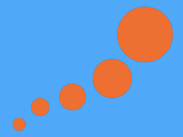
Implement `FizzBuzz(n int) (string, error)`
Bonus: Write one or more tests to exercise the function

https://github.com/domdavis/go-get-better/tree/exercise-4

```go
var ErrNegativeRange = errors.New("cannot produce negative amounts of FizzBuzz")

func FizzBuzz(n int) (string, error) {
    var b strings.Builder

    if n < 0 {
        return "", ErrNegativeRange
    }

    for i := 1; i <= n; i++ {
        if i%3 == 0 {
            b.WriteString("Fizz")
        }

        if i%5 == 0 {
            b.WriteString("Buzz")
        }

        if i%3 != 0 && i%5 != 0 {
            b.WriteString(strconv.Itoa(i))
        }

        b.WriteString(" ")
    }

    return strings.TrimSpace(b.String()), nil
}
```

```go
package exercise4_test
import (
	"fmt"
	"testing"
	"training/exercise4"
)

func ExampleFizzBuzz() {
	if r, err := exercise4.FizzBuzz(15); err != nil {
		fmt.Println(err)
	} else {
		fmt.Println(r)
	}

	// Output:
	// 1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz
}

func TestFizzBuzz(t *testing.T) {
	_, err := exercise4.FizzBuzz(-1)
	if err != exercise4.ErrNegativeRange {
		t.Errorf("unexpected error: %s", err)
	}
}
```

loops



framework training

```go
package main

import "fmt"

func main() {
    for i := 10; i <= 100; i = i + 10 {
        fmt.Println(i)
    }
}
```

```go
package main

import "fmt"

func main() {
    var i int

    for i < 10 {
        i++
        fmt.Println(i)
    }
}
```

```go
package main

import "fmt"

func main() {
    for true {
        fmt.Println("Spam")
    }
}
```

```go
package main

import "fmt"

func main() {
    for {
        fmt.Println("Spam")
    }
}
```

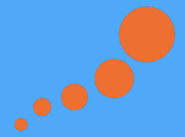
```go
package main

import "fmt"

func main() {
    for i := range []int{1, 1, 2, 3, 5, 8} {
        fmt.Println(i)
    }
}
```

```go
package main

import "fmt"

func main() {
    for _, v := range []int{1, 1, 2, 3, 5, 8} {
        fmt.Println(v)
    }
}
```

```go
package main

import "fmt"

func main() {
    m := map[string]int{
        "one": 1,
        "two": 2,
    }

    for k, v := range m {
        fmt.Println(k, v)
    }
}
```

```go
func TestFizzBuzz(t *testing.T) {
    for i, expected := range map[int]string{
        1: "1",
        2: "1 2",
        3: "1 2 Fizz",
    } {
        if r, err := exercise4.FizzBuzz(i); err != nil {
            t.Errorf("unexpected error FizzBuzzing: %s", err)
        } else if r != expected {
            t.Errorf("expected '%s', got '%s'", expected, r)
        }
    }
}
```

Arrays and Slices



framework training

```go
package main

import "fmt"

func main() {
    var a [4]int
    fmt.Println(a)
}
```

```go
package main

import "fmt"

func main() {
    var a [4]int
    fmt.Println(a)
}

// [0 0 0 0]
```

```go
package main

import "fmt"

func main() {
    a := [2]int{1, 2}
    b := [4]int{1, 2}
    c := [...]int{1, 2}
    fmt.Println(a, b, c)
}

// [1 2] [1 2 0 0] [1 2]
```

```go
package main

import "fmt"

func main() {
    var a []int // read "slice of int"
    a = append(a, 123)
    a = append(a, 456)
    fmt.Println(a)
}

// [123 456]
```

```go
package main

import "fmt"

func main() {
    a := []int{123, 456, 789}
    fmt.Printf("%d\n", a[0])

    a[1] = 555
    fmt.Printf("%v %v %v\n", a[1:], a[:1], a[1:2])
}

// 123
// [555 789] [123] [555]
```

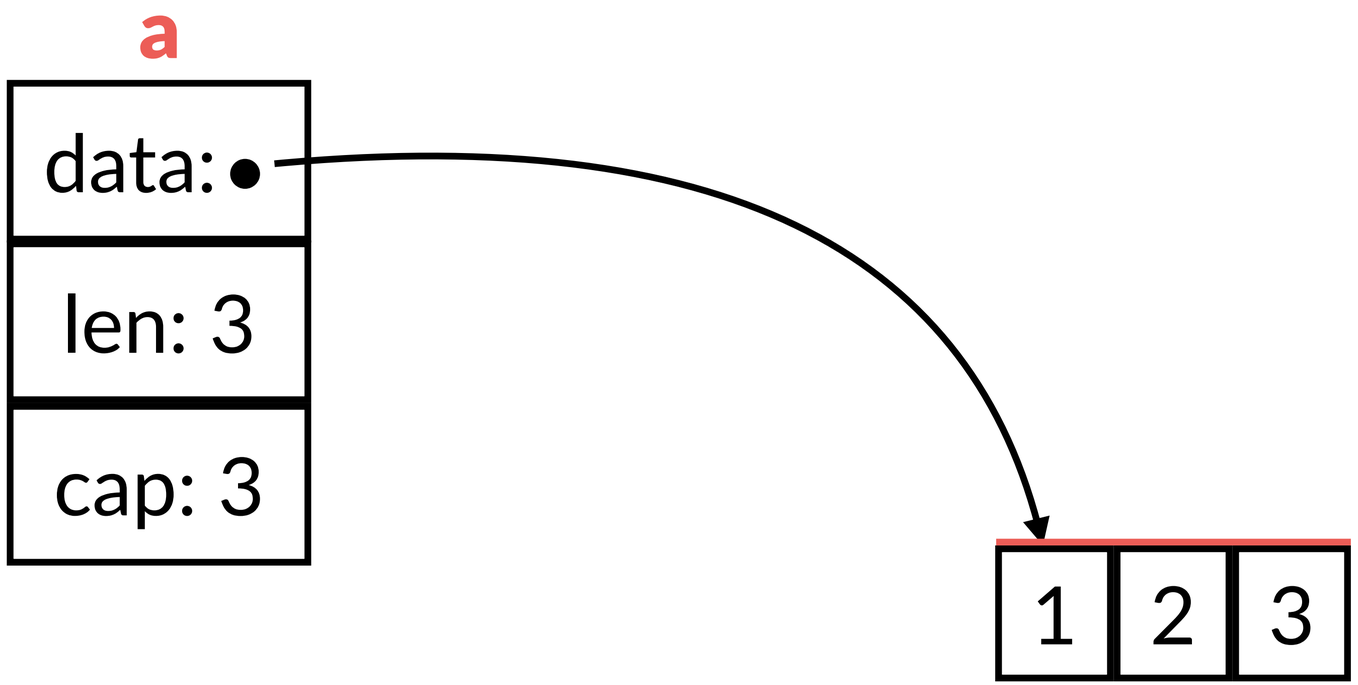
```go
package main

import "fmt"

func main() {
    a := [][]int{
        []int{1, 2, 3},
        []int{4, 5},
        []int{6, 7, 8, 9},
    }

    a = append(a, []int{10, 11, 12})
    fmt.Printf("%v\n", a)
}

// [[1 2 3] [4 5] [6 7 8 9] [10 11 12]]
```

```go
package main

import "fmt"

func main() {
    a := [][]int{
        {1, 2, 3},
        {4, 5},
        {6, 7, 8, 9},
    }

    a = append(a, []int{10, 11, 12})
    fmt.Printf("%v\n", a)
}

// [[1 2 3] [4 5] [6 7 8 9] [10 11 12]]
```

```go
package main

func main() {
    a := make([]int, 5)
    a[3] = 123 // OK
    a = append(a, 456)

    b := make([]int, 0, 5)
    b[3] = 123 // panic!
    b = append(b, 789)
}
```

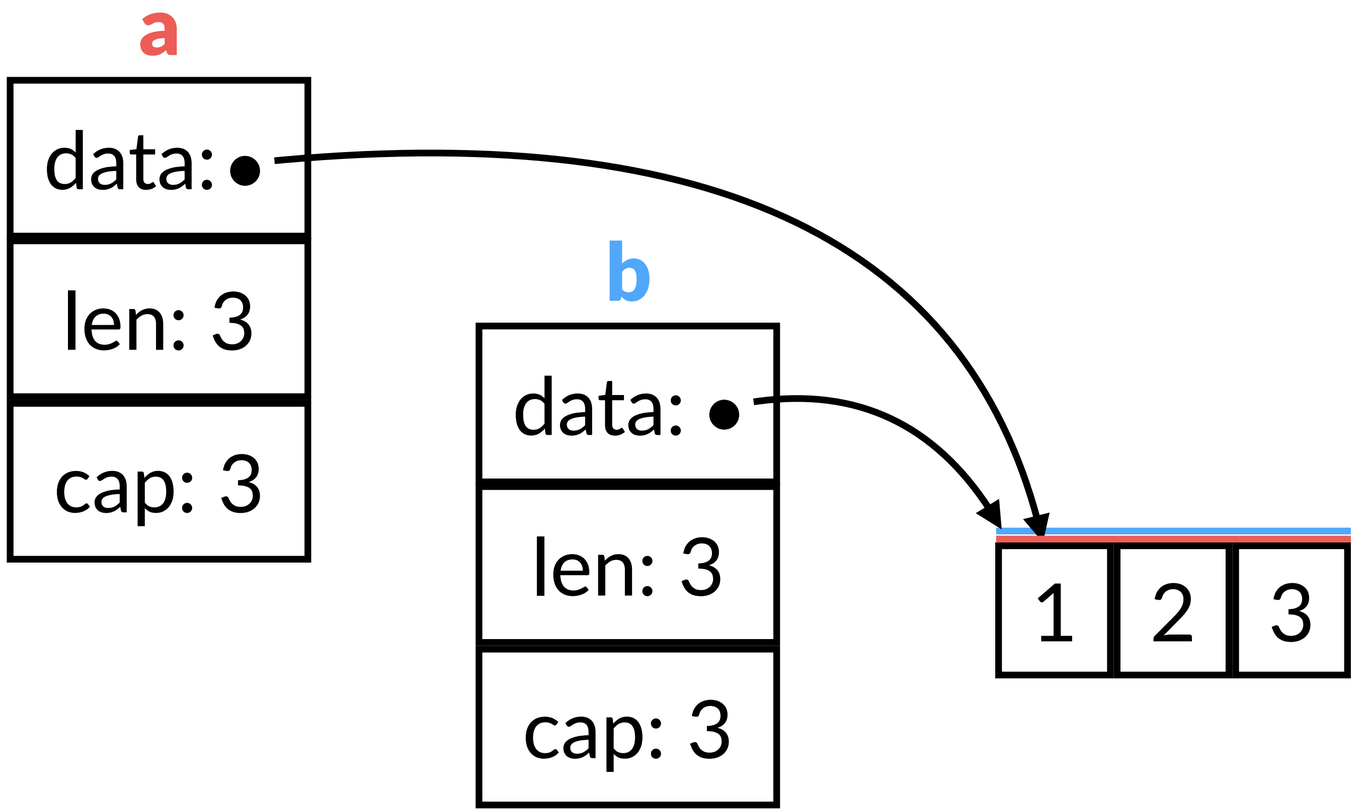
Slice Implementation

```
a := []int{1, 2, 3}
```



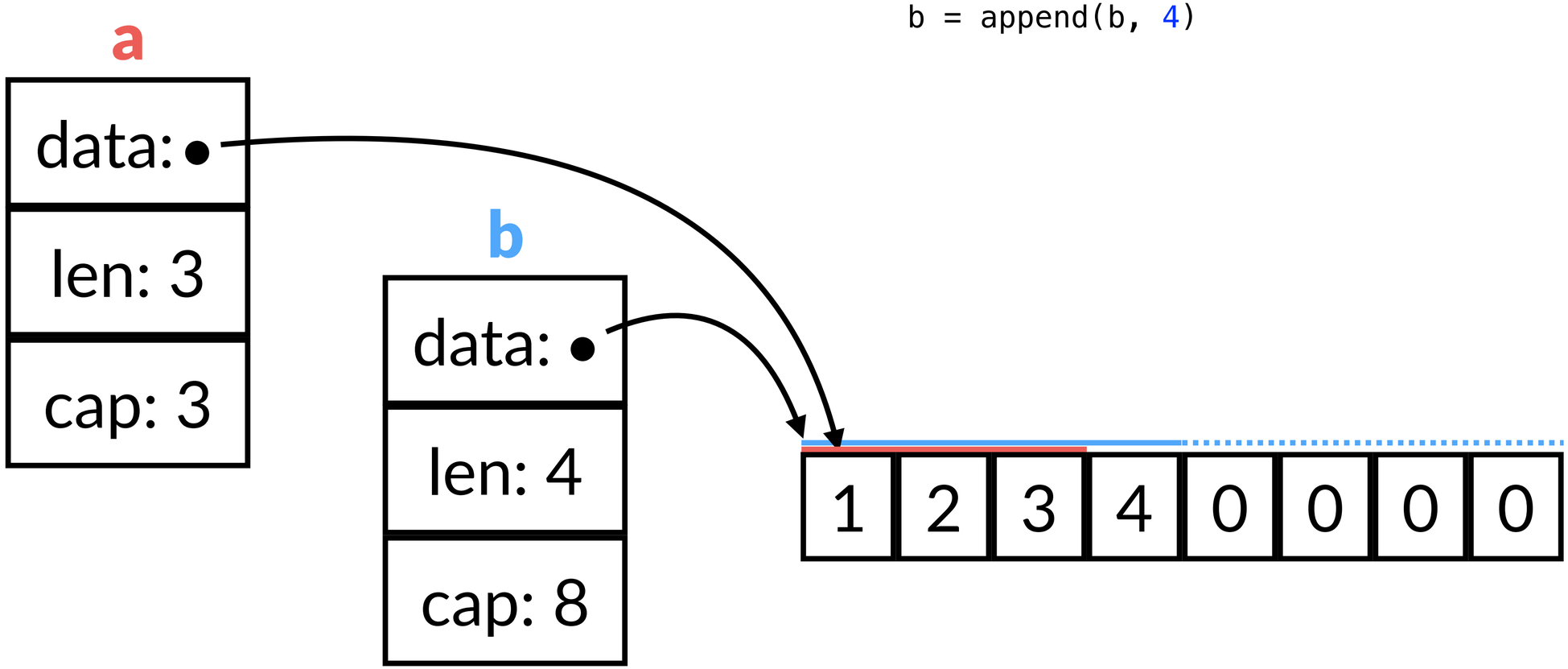
Slice Implementation

```
a := []int{1, 2, 3}
b := a
```



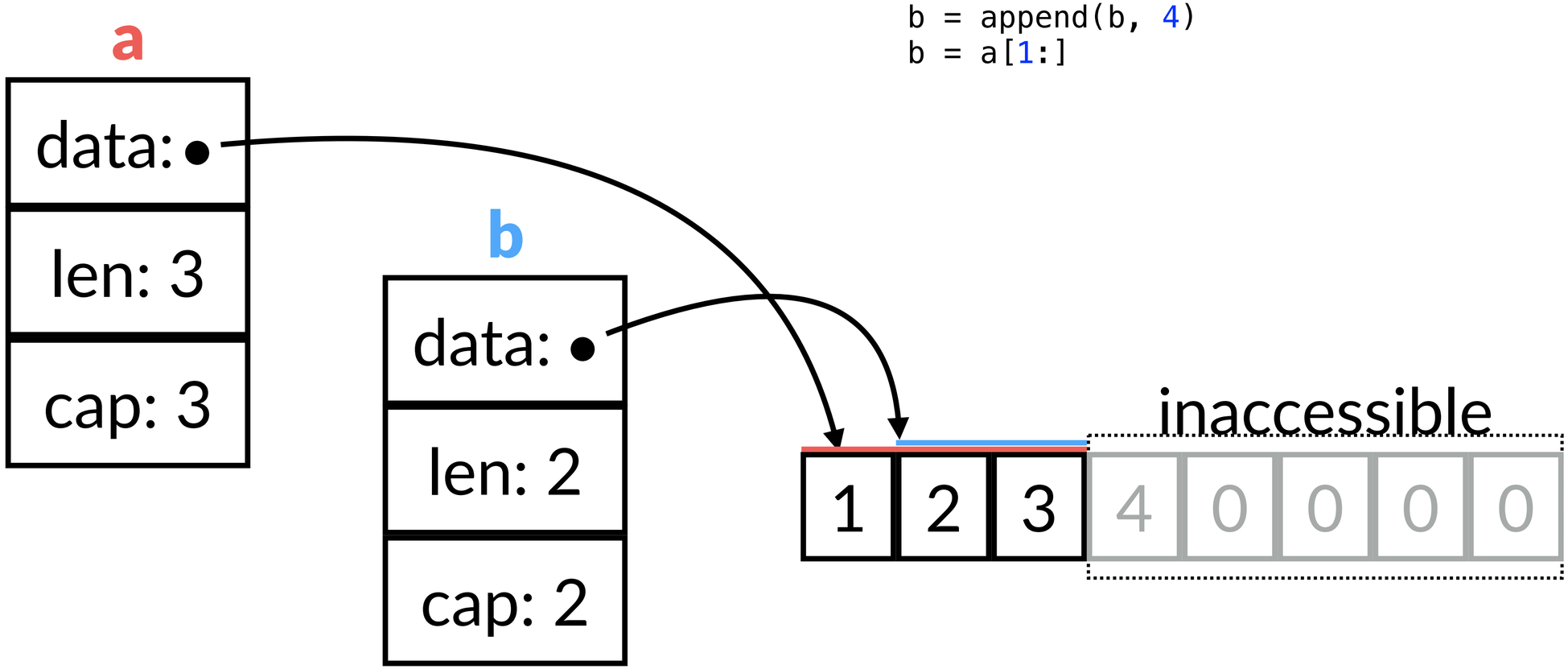
Slice Implementation

```
a := []int{1, 2, 3}
b := a
b = append(b, 4)
```



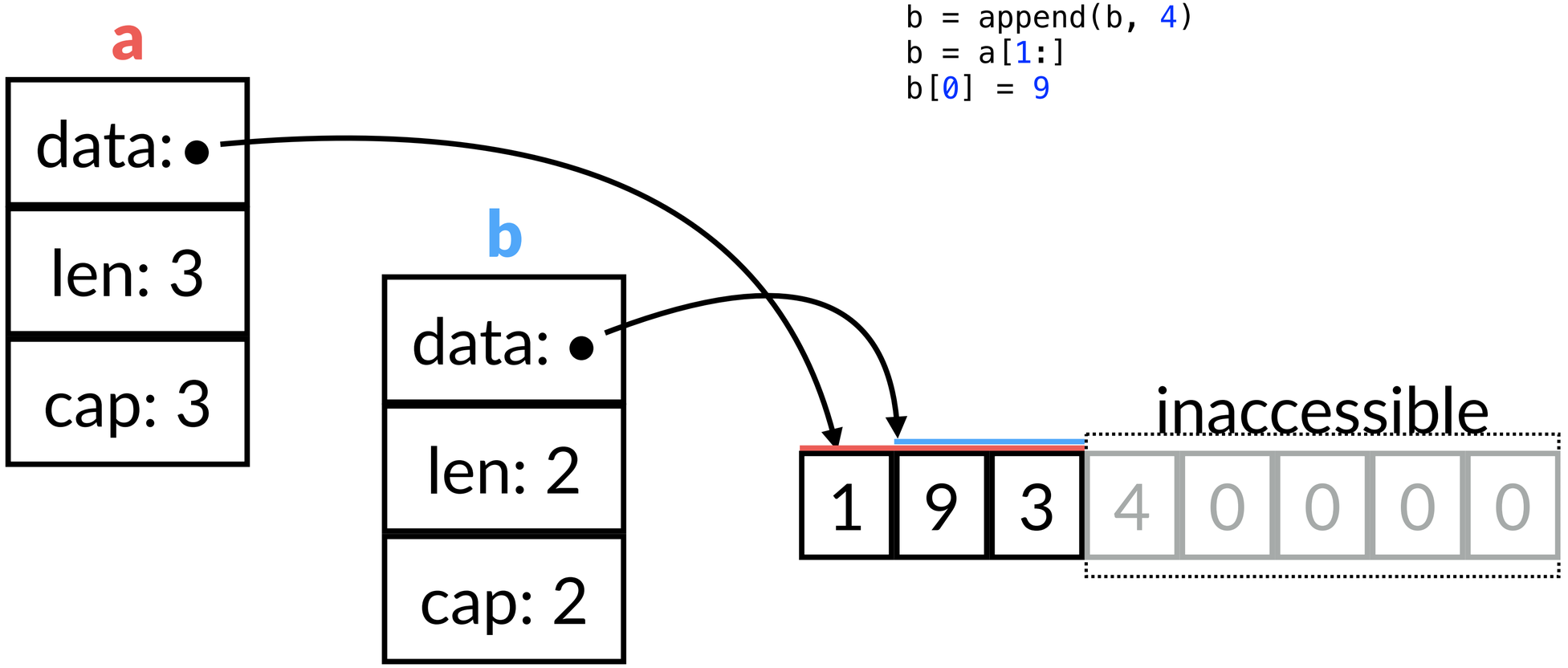
Slice Implementation

```
a := []int{1, 2, 3}
b := a
b = append(b, 4)
b = a[1:]
```



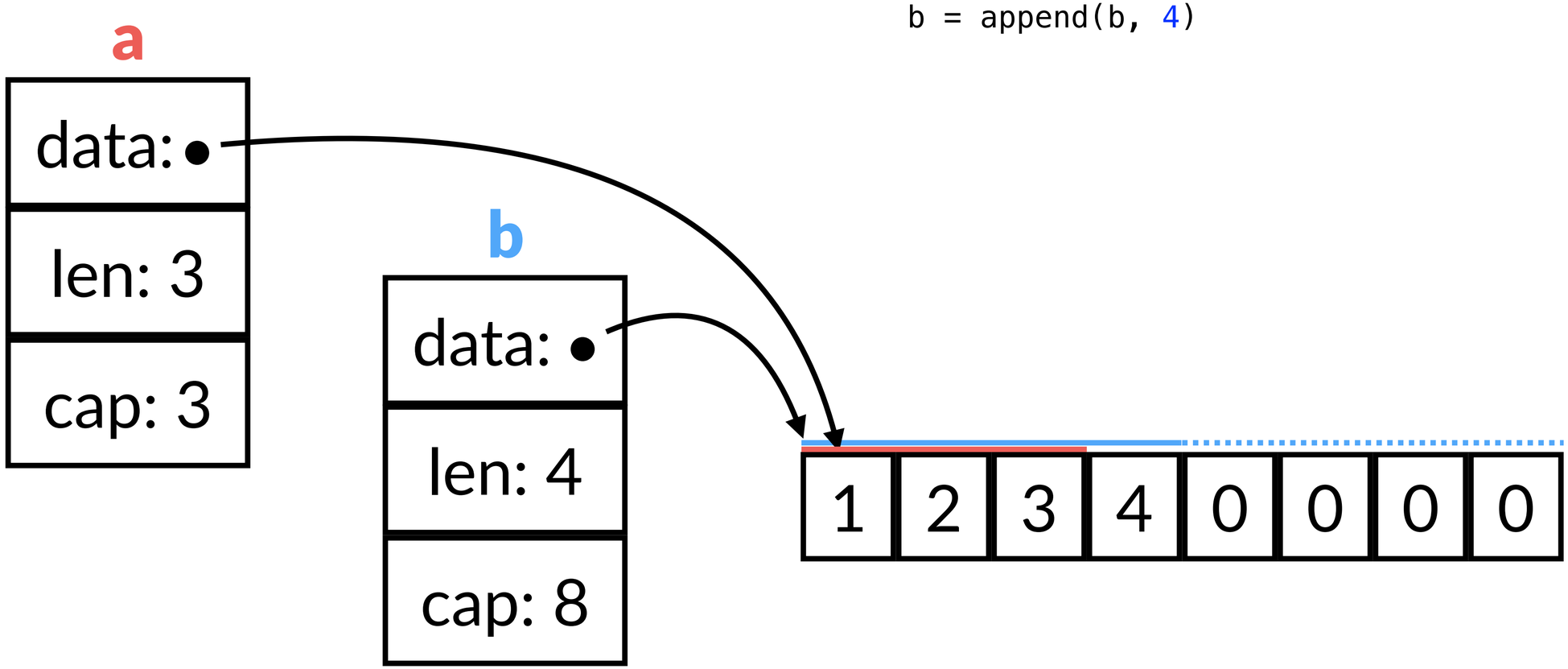
Slice Implementation

```
a := []int{1, 2, 3}
b := a
b = append(b, 4)
b = a[1:]
b[0] = 9
```



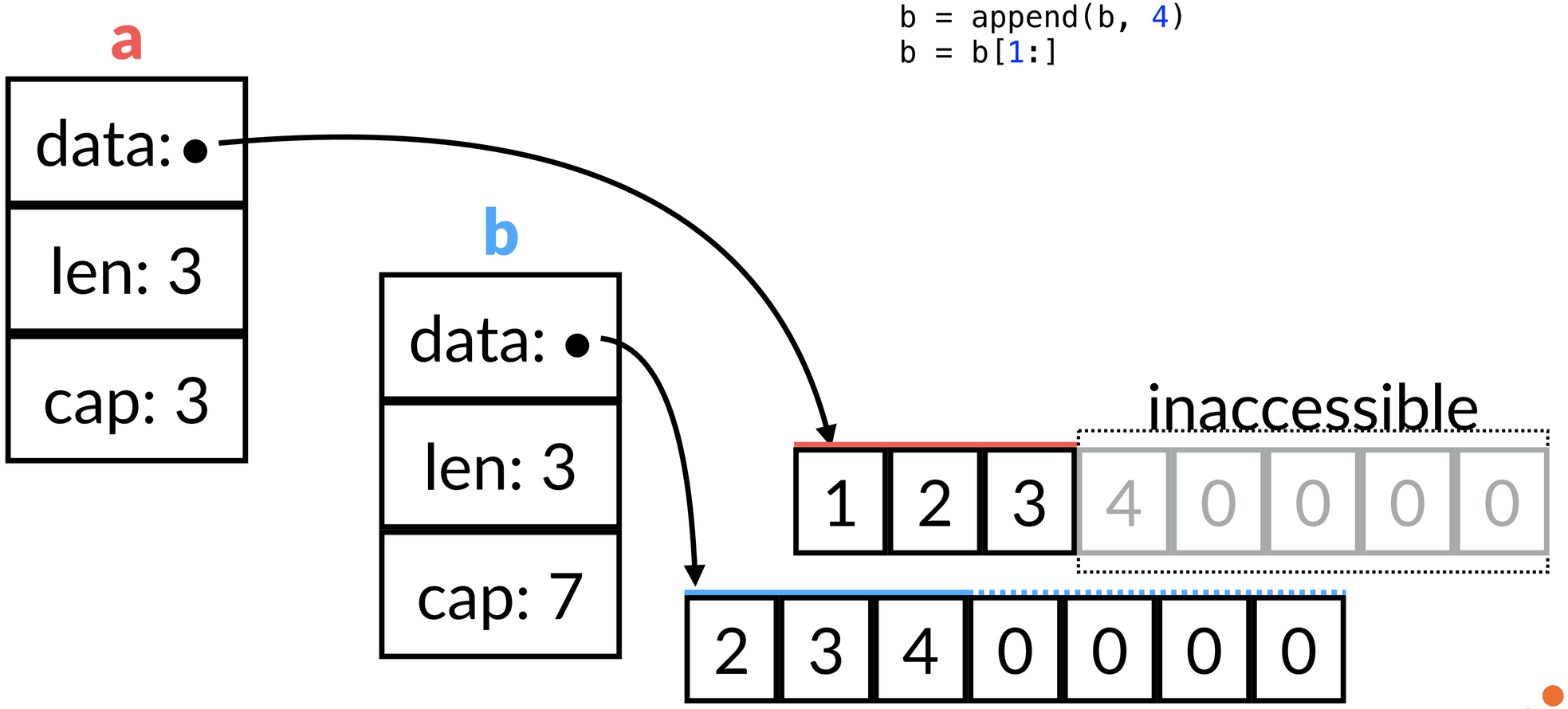
Slice Implementation

```
a := []int{1, 2, 3}
b := a
b = append(b, 4)
```



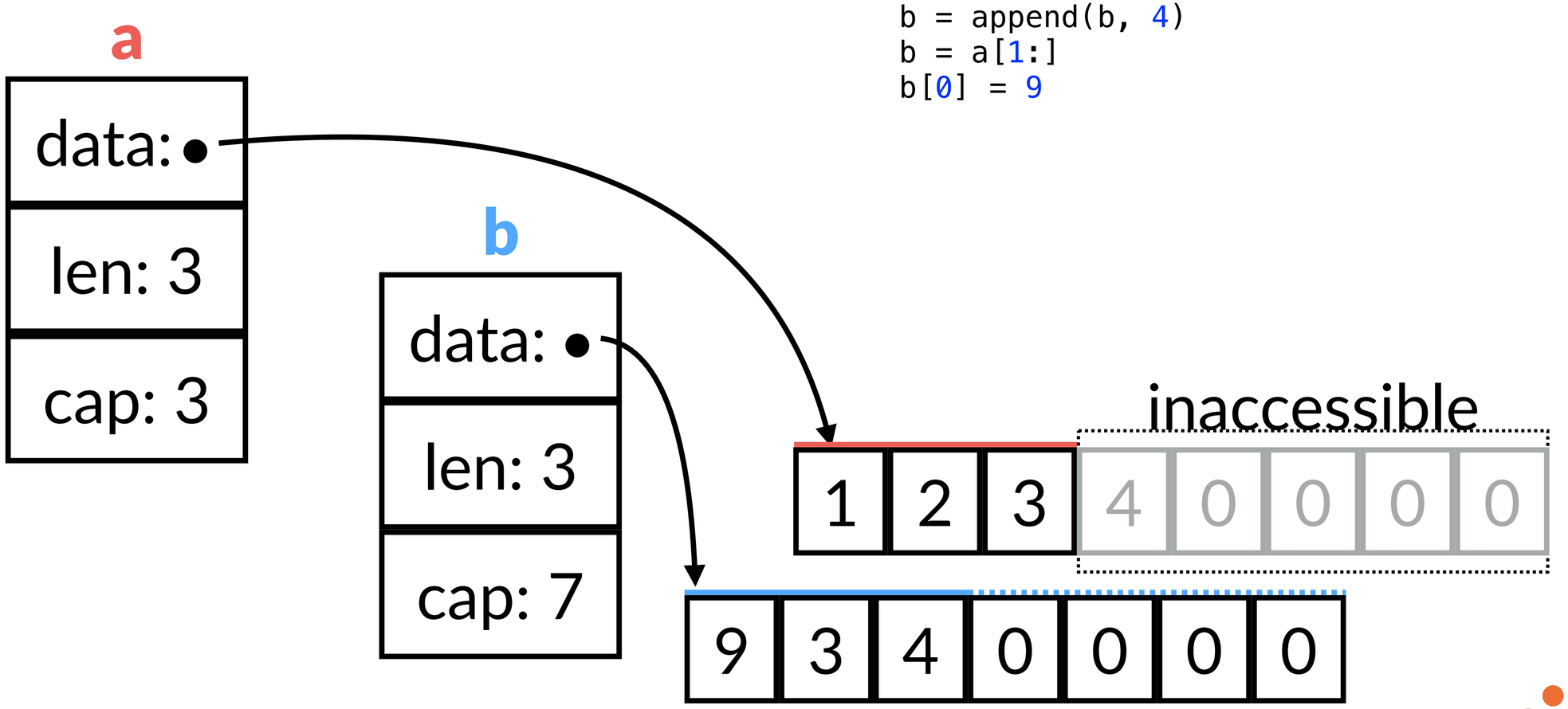
Slice Implementation

```
a := []int{1, 2, 3}
b := a
b = append(b, 4)
b = b[1:]
```



Slice Implementation

```
a := []int{1, 2, 3}
b := a
b = append(b, 4)
b = a[1:]
b[0] = 9
```



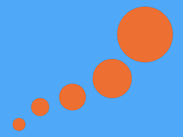
5: `FizzBuzz(int) ([]string, error)`

Implement `FizzBuzz(n int) ([]string, error)`
Bonus: Write one or more tests to exercise the function

https://github.com/domdavis/go-get-better/tree/exercise-5

```
func FizzBuzz(n int) ([]string, error) {
    if n < 0 { return []string{}, ErrNegativeRange }
    s := make([]string, n)
    for i := 1; i <= n; i++ {
        switch {
        case i%3 == 0 && i%5 == 0:
            s[i-1] = "FizzBuzz"
        case i%3 == 0:
            s[i-1] = "Fizz"
        case i%5 == 0:
            s[i-1] = "Buzz"
        default:
            s[i-1] = strconv.Itoa(i)
        }
    }
    return s, nil
}
```

Zero Values



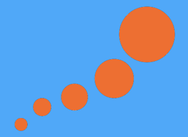
framework training

```go
package main

import "fmt"

func main() {
    var b bool
    var i int
    var f float64
    var s string
    fmt.Printf("%t %d %f %q", b, i, f, s)
}

// false 0 0.000000 ""
```

nil

```go
package main

import "fmt"

func main() {
    fmt.Println([]string{})
}

// []
```

maps



framework training

```go
package main

func main() {
    var m map[string]int
    m["A"] = 1
}
```

```go
package main

func main() {
    var m map[string]int = make(map[string]int)
    m["A"] = 1
}
```

```go
package main

func main() {
    var m = make(map[string]int)
    m["A"] = 1
}
```

```go
package main

func main() {
    m := map[string]int{}
    m["A"] = 1
}
```

```go
package main

func main() {
    var m = make(map[string]int, 100)
    m["A"] = 1
}
```

```go
package main

import "fmt"

func main() {
    m := map[string]int{"A": 1}
    fmt.Println(m)
}

// map[A:1]
```

MapType = "map" "[" **KeyType** "]" ElementType

"The comparison operators == and != must be fully defined for operands of the key type; thus the key type must not be a function, map, or slice."

https://golang.org/doc/ref#KeyType

```go
package main

func main() {
    m := map[string]int{"A": 1}
    println(m["A"])
}

// 1
```

```go
package main

func main() {
    m := map[string]int{"A": 1}
    println(m["B"])
}

// 0
```

```go
package main

import "fmt"

func main() {
    m := map[string]int{"A": 1}
    v, ok := m["B"]
    fmt.Println(v, ok)
}

// 0, false
```

```go
package main

import "fmt"

func main() {
    m := map[string]int{"A": 0}
    v, ok := m["A"]
    fmt.Println(v, ok)
}
```

```go
package main

import "fmt"

func main() {
    m := map[string]int{"A": 1}
    delete(m, "A")
    v, ok := m["A"]
    fmt.Println(v, ok)
}
```

```go
func main() {
    m := map[string]int{"A": 1, "B": 2, "C": 3}

    for k := range m {
        fmt.Print(k, " ")
    }

    for k, v := range m {
        fmt.Print(k, " ", v, " ")
    }

    for _, v := range m {
        fmt.Print(v, " ")
    }
}
```

```go
package main

func main() {
    m := map[int]string{
        1: "A", 2: "B", 3: "C"}

    for k := range m {
        if k%2 == 0 {
            delete(m, k)
        }
    }
}
```

The iteration order over maps is not specified and is not guaranteed to be the same from one iteration to the next. If map entries that have not yet been reached are **removed during iteration**, the corresponding iteration values will not be produced. If map entries are **created during iteration**, that entry may be produced during the iteration or may be skipped. The choice may vary for each entry created and from one iteration to the next. If the map is nil, the number of iterations is 0.

6: `FizzBuzz(int) (map[int]string, error)`

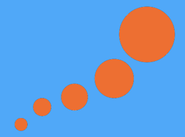
Implement `FizzBuzz(n int) (map[int]string, error)`
The map should contain Fizz, Buzz, FizzBuzz or the number as the value
Bonus: Write one or more tests to exercise the function

https://github.com/domdavis/go-get-better/tree/exercise-6

```
func FizzBuzz(n int) (map[int]string, error) {
    if n < 0 { return nil, ErrNegativeRange }
    m := make(map[int]string, n)
    for i := 1; i <= n; i++ {
        switch {
        case i%3 == 0 && i%5 == 0:
            m[i] = "FizzBuzz"
        case i%3 == 0:
            m[i] = "Fizz"
        case i%5 == 0:
            m[i] = "Buzz"
        default:
            m[i] = strconv.Itoa(i)
        }
    }
    return m, nil
}
```

```go
func ExampleFizzBuzz() {
    n := 15

    r, err := exercise6.FizzBuzz(n)

    if err != nil {
        fmt.Println(err)
    }

    for i := 1; i <= n; i++ {
        fmt.Print(r[i], " ")
    }

    // Output:
    // 1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz
}
```

closures



framework training

```go
package main

import "fmt"

func main() {
    a := func() int { return 123 }
    b := a()
    fmt.Println(b)
}
```

```go
package main

func apply(f func(int) int, i int) int {
    return f(i)
}

func main() {
    double := func(x int) int { return x+x }
    square := func(x int) int { return x*x }
    i := apply(double, apply(square, 3))
    println(i)
}
```

```go
package main

func apply(f func(int) int, i int) int {
    return f(i)
}

func square(n int) int {
    return n * n
}

func main() {
    f := square
    i := apply(f, 3)
    println(i)
}
```

```go
package main

func apply(f func(int) int, i int) int {
 return f(i)
}

func main() {
 i := apply(func(x int) int { return x+x }, 3)
 println(i)
}
```

```
package main

func apply(f func(int) int, i int) int {
    return f(i)
}

func main() {
    n := 123
    f := func(x int) int { return n }
    i := apply(f, 3)
    println(i)
}
```

7: `Run()`

Implement

```
Run(n int, func (int) ([]string, error)) ([]string, error)
```

Pass your FizzBuzz function from exercise 5 into sequence in the test code.

https://github.com/domdavis/go-get-better/tree/exercise-7

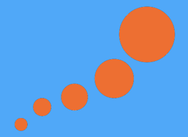
```
func FizzBuzz(n int) ([]string, error) {
    if n < 0 { return []string{}, ErrNegativeRange }
    s := make([]string, n)
    for i := 1; i <= n; i++ {
        switch {
        case i%3 == 0 && i%5 == 0:
            s[i-1] = "FizzBuzz"
        case i%3 == 0:
            s[i-1] = "Fizz"
        case i%5 == 0:
            s[i-1] = "Buzz"
        default:
            s[i-1] = strconv.Itoa(i)
        }
    }
    return s, nil
}
```

```go
func Run(n int, f func(int) ([]string, error)) ([]string, error) {
    return f(n)
}
```

```go
func ExampleRun() {
    r, err := exercise7.Run(15, exercise7.FizzBuzz)

    if err != nil {
        fmt.Println(err)
    }

    for _, v := range r {
        fmt.Print(v, " ")
    }

    // Output:
    // 1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz
}
```

```go
func Run(n int, f func(int) ([]string, error)) ([]string, error) {
    return f(n)
}
```

types



framework training

```go
package types

func Handle(code int) {
    switch code {
    case 200:
        // OK
    case 404:
        // Not found
    case 500:
        // Server error
    }
}
```

bool
byte
uintptr
int, int8, int16, int32, int64
uint, uint8, uint16, uint32, uint64
float32, float64
complex64, complex128
rune, string
slice, map
interface, struct

```go
package main

import "fmt"

type ResponseCode int

func main() {
    Handle(200)
}

func Handle(code ResponseCode) {
    switch code {
    case 200:
        fmt.Println("OK")
    case 404:
        fmt.Println("Not Found")
    case 500:
        fmt.Println("Server Error")
    default:
        fmt.Printf("Unknown code %d\n", code)
    }
}
```

```go
package main

import "fmt"

type ResponseCode int

func main() {
    a := ResponseCode(200)
    b := 200

    if a == b {
        fmt.Println("Equal")
    }
}
```

```go
package main

import "fmt"

type ResponseCode int

func main() {
    Handle(200)
}

func Handle(code ResponseCode) {
    switch code {
    case 200:
        fmt.Println("OK")
    case 404:
        fmt.Println("Not Found")
    case 500:
        fmt.Println("Server Error")
    default:
        fmt.Printf("Unknown code %d\n", code)
    }
}
```

```go
package main

import "fmt"

type ResponseCode int

func main() {
    a := ResponseCode(200)
    b := 200

    if a == b {
        fmt.Println("Equal")
    }
}
```

```go
package main

import "fmt"

type ResponseCode int

func main() {
    a := ResponseCode(200)
    b := 200

    if int(a) == b {
        fmt.Println("Equal")
    }
}
```

```go
package main

import "fmt"

type ResponseCode int

func main() {
    a := ResponseCode(200)
    b := 200

    if a == ResponseCode(b) {
        fmt.Println("Equal")
    }
}
```

```go
func Run(n int, f func(int) ([]string, error)) ([]string, error) {
    return f(n)
}
```

```
type Sequence []string
type Generator func(int) (Sequence, error)

func Run(g Generator, n int) (Sequence, error) {
    return g(n)
}
```

```go
type Sequence []string
type Generator func(int) (Sequence, error)

var ErrNegativeRange = errors.New("cannot produce negative sequence")

func Run(g Generator, n int) (Sequence, error) {
    return g(n)
}

func Simple(n int) (Sequence, error) {
    if n < 0 {
        return Sequence{}, ErrNegativeRange
    }
    r := make(Sequence, n)
    for i := 1; i <= n; i++ { r[i-1] = strconv.Itoa(i) }
    return r, nil
}

func main() {
    s, _ := Run(Simple, 5)
    fmt.Println(s)
}

// [1 2 3 4 5]
```

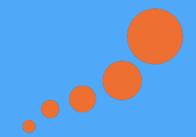
8: Putting it all together

Using the template branch, make the tests work using your FizzBuzz code

https://github.com/domdavis/go-get-better/tree/template
https://github.com/domdavis/go-get-better/tree/exercise-8

```go
func FizzBuzz(n int) (Sequence, error) {
    if n < 0 {
        return []string{}, ErrNegativeRange
    }

    s := make([]string, n+1)
    s[0] = "FizzBuzz"

    for i := 1; i <= n; i++ {
        switch {
        case i%3 == 0 && i%5 == 0:
            s[i] = "FizzBuzz"
        case i%3 == 0:
            s[i] = "Fizz"
        case i%5 == 0:
            s[i] = "Buzz"
        default:
            s[i] = strconv.Itoa(i)
        }
    }
    return s, nil
}
```

structs



framework training

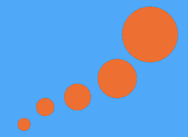
```
type Car struct {
    Make string
    Model string
    Doors int
}
```

```
type Car struct {
    Make string
    Model string
    Doors int
}

var Corolla = Car{"Toyota", "Corolla", 4}
```

```
type Car struct {
    Make string
    Model string
    Doors int
}

var Corolla = Car{Make: "Toyota", Model: "Corolla"}
```

```go
package main

import "fmt"

type Car struct {
    Make string
    Model string
    Doors int
}

var Corolla = Car{Make: "Toyota", Model: "Corolla"}

func main() {
    Corolla.Doors = 5
    fmt.Println(Corolla)
}

// {Toyota Corolla 5}
```

defer



framework training

```go
package main

func main() {
    defer println("goodbye")
    println("hello")
}
```

```go
package main

import (
    "log"
    "os"
)

func main() {
    f, err := os.Open("/etc/passwd")
    if err != nil {
        log.Fatal(err)
    }
    defer f.Close()
    // use f
}
```

```go
package main

import (
    "log"
    "os"
)

func main() {
    f, err := os.Open("/etc/passwd")
    if err != nil {
        log.Fatal(err)
    }
    defer func() { _ = f.Close() }()
    // use f
}
```

```go
package main

func main() {
    defer println("one")
    println("two")
    defer println("three")
}
```

```go
package main

func main() {
    i := 1
    defer println(i)
    i++
    defer println(i)
    println(i)
}
```

```go
package main

func main() {
    for i := 0; i < 5; i++ {
        println("open file", i)
        defer println("close file", i)
        println("use file", i)
    }
}
```

```go
func DeferredReverse(n int) (Sequence, error) {
    if n < 0 {
        return Sequence{}, ErrNegativeRange
    }

    s := []string{"DeferredReverse"}

    for i := 1; i <= n; i++ {
        defer func() { s = append(s, strconv.Itoa(i)) }()
    }

    return s, nil
}
```

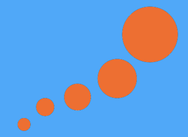
```go
func DeferredReverse(n int) (Sequence, error) {
    if n < 0 {
        return Sequence{}, ErrNegativeRange
    }

    s := []string{"DeferredReverse"}

    for i := 1; i <= n; i++ {
        defer func() {
            s = append(s, strconv.Itoa(i))
            fmt.Println(s)
        }()
    }

    return s, nil
}
```

```
[DeferredReverse 6]
[DeferredReverse 6 6]
[DeferredReverse 6 6 6]
[DeferredReverse 6 6 6 6]
[DeferredReverse 6 6 6 6 6]
```

```go
func DeferredReverse(n int) (Sequence, error) {
    if n < 0 {
        return Sequence{}, ErrNegativeRange
    }

    s := []string{"DeferredReverse"}

    for i := 1; i <= n; i++ {
        defer func() {
            s = append(s, strconv.Itoa(i))
            fmt.Println(s)
        }()
    }

    return s, nil
}
```

```
func DeferredReverse(n int) (Sequence, error) {
    if n < 0 {
        return Sequence{}, ErrNegativeRange
    }

    s := []string{"DeferredReverse"}

    for i := 1; i <= n; i++ {
        defer func(i int) {
            s = append(s, strconv.Itoa(i))
            fmt.Println(s)
        }(i)
    }

    return s, nil
}
```

Pointers



framework training

```go
package main

import "fmt"

func main() {
    a := "Thing" // I am a thing
    b := &a      // I point to the thing
    c := *b      // I am the thing that was pointed to

    a = "New Thing"

    fmt.Println(a, b, c)
}
```

```go
package main

import "fmt"

func main() {
    a := "Thing" // I am a thing
    b := &a      // I point to the thing
    c := *b      // I am the thing that was pointed to

    a = "New Thing"

    fmt.Println(a, b, c)
}

// New Thing 0xc0000601c0 Thing
```

```go
package main

import "fmt"

func main() {
    a := "Thing" // I am a thing
    b := &a      // I point to the thing
    c := *b      // I am the thing that was pointed to

    a = "New Thing"

    fmt.Println(a, *b, c)
}

// New Thing New Thing Thing
```

```
func DeferredReverse(n int) (Sequence, error) {
    if n < 0 {
        return Sequence{}, ErrNegativeRange
    }

    s := Sequence{"DeferredReverse"}
    p := &s

    for i := 1; i <= n; i++ {
        defer func(i int) { *p = append(*p, strconv.Itoa(i)) }(i)
    }

    return s, nil
}
```

```
type Generator func(int) (*Sequence, error)
```

```go
type Generator func(int) (*Sequence, error)

func DeferredReverse(n int) (*Sequence, error) {
    if n < 0 {
        return &Sequence{}, ErrNegativeRange
    }

    s := &Sequence{"DeferredReverse"}

    for i := 1; i <= n; i++ {
        defer func(i int) { *s = append(*s, strconv.Itoa(i)) }(i)
    }

    return s, nil
}
```

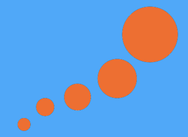
```
func Run(g Generator, n int) (Sequence, error) {
    p, err := g(n)
    return *p, err
}
```

9: Pointers

Implement deferred reverse into our training package

https://github.com/domdavis/go-get-better/tree/exercise-9

types revisited



framework training

```go
package main

import "fmt"

var counter int

func main() {
    counter++
    fmt.Println(counter)
}
```

```go
package main

import "fmt"

type counter int

func main() {
    var c counter
    c++
    fmt.Println(c)
}
```

```go
package main

import "fmt"

type counter int

func (c counter) String() string {
    return fmt.Sprintf("Counter: %d\n", c)
}

func main() {
    var c counter
    c++
    fmt.Println(c)
}
```

```go
package main

import "fmt"

type counter int

func (c counter) Increment() {
    c++
}

func main() {
    var c counter
    c.Increment()
    fmt.Println(c)
}
```

```go
package main

import "fmt"

type counter int

func (c *counter) Increment() {
    *c = *c + 1
}

func main() {
    var c *counter
    c.Increment()
    fmt.Println(c)
}
```

```go
type counter struct {
    value int
}

func (c *counter) Increment() {
    c.value++
}

func main() {
    var c counter
    c.Increment()
    fmt.Println(c)
}
```

```go
type counter struct {
    value int
}

func (c *counter) Increment() {
    c.value++
}

func (c counter) String() string {
    return strconv.Itoa(c.value)
}

func main() {
    var c counter
    c.Increment()
    fmt.Println(c)
}
```

```go
type counter struct {
    value int
}

func (c *counter) Increment() {
    c.value++
}

func (c *counter) String() string {
    return strconv.Itoa(c.value)
}

func main() {
    var c counter
    c.Increment()
    fmt.Println(c)
}
```

```go
type counter struct {
    value int
}

func (c *counter) Increment() {
    c.value++
}

func (c *counter) String() string {
    return strconv.Itoa(c.value)
}

func main() {
    c := &counter{}
    c.Increment()
    fmt.Println(c)
}
```

Making New Types

```
// Value types
var a thing
var b thing = make(thing)
var c thing = thing{}
var d = make(thing)
var e = thing{}
f := make(thing)
g := thing{}
```

```
// Pointer types
var a *thing
var b *thing = new(thing)
var c *thing = &thing{}
var d = new(thing)
var e = &thing{}
f := new(thing)
g := &thing{}
```

Making New Types

```
// Value types
var a thing
var b thing = make(thing)
var c thing = thing{}
var d = make(thing)
var e = thing{}
f := make(thing)
g := thing{}
```

```
// Pointer types
var a *thing
var b *thing = new(thing)
var c *thing = &thing{}
var d = new(thing)
var e = &thing{}
f := new(thing)
g := &thing{}
```

Making New Types

```
// Value types
var a thing
var b thing = make(thing)
var c thing = thing{}
var d = make(thing)
var e = thing{}
f := make(thing) // slice + map
g := thing{}
```

```
// Pointer types
var a *thing
var b *thing = new(thing)
var c *thing = &thing{}
var d = new(thing)
var e = &thing{}
f := new(thing)
g := &thing{}
```

10: `Generator.Run()`

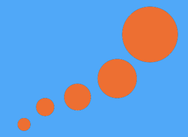
Change the `Run` function so that it's a type function for `Generator`.
You'll need to update the example and the tests to call the function correctly

https://github.com/domdavis/go-get-better/tree/exercise-10

```
func (g Generator) Run(n int) (Sequence, error) {
    p, err := g(n)
    return *p, err
}
```

```go
func ExampleSequence() {
    simple := exercise10.Generator(exercise10.Simple)
    if r, err := simple.Run(5); err != nil {
        fmt.Println(err)
    } else {
        for _, v := range r {
            fmt.Print(v, " ")
        }
    }

    // Output:
    // Simple 1 2 3 4 5
}
```

panic and recover



framework training

```go
package main

func die() {
    panic("process died")
}

func main() {
    die()
    println("hello")
}
```

```
panic: process died

goroutine 1 [running]:
main.die(...)
    /Users/davisd/Library/Preferences/IntelliJIdea2018.3/
scratches/scratch.go:4
main.main()
    /Users/davisd/Library/Preferences/IntelliJIdea2018.3/
scratches/scratch.go:8 +0x39

Process finished with exit code 2
```

```go
package main

import "fmt"

func die() {
    defer func() {
        if x := recover(); x != nil {
            fmt.Printf("recovered: %v\n", x)
        }
    }()
    panic("process died")
}

func main() {
    die()
    println("hello")
}
```

Don't Panic!

Interfaces

```
type duck interface {
    walk()
    quack()
}
```

```
type duck interface {
    walk()
    quack()
}

type goose struct{}

func (g goose) walk()  { println("waddle, waddle") }
func (g goose) quack() { println("HONK") }
func (g goose) eat()   { println("om nom nom") }
```

```
type duck interface {
    walk()
    quack()
}

type goose struct{}

func (g goose) walk()  { println("waddle, waddle") }
func (g goose) quack() { println("HONK") }
func (g goose) eat()   { println("om nom nom") }

func main() {
    var g duck
    g = goose{}
    g.walk()
}
```

```
func annoy(d duck) {
    d.walk()
    d.quack()
    //d.eat() // doesn't work
}

func main() {
    g := goose{}
    annoy(g)
}
```

```
func main() {
    var walker interface{
        walk()
    }

    walker = goose{}
    walker.walk()
}
```

```go
type Reader interface {
    Read(p []byte) (n int, err error)
}

type Writer interface {
    Write(p []byte) (n int, err error)
}


type ReadWriter interface {
    Reader
    Writer
}
```

Interfaces Should Be Small

```
type Store interface {
    Put(key string, value []byte, options *WriteOptions) error
    Get(key string) (*KVPair, error)
    Delete(key string) error
    Exists(key string) (bool, error)
    Watch(key string, stopCh <-chan struct{}) (<-chan *KVPair, error)
    WatchTree(directory string, stopCh <-chan struct{}) (<-chan []*KVPair, error)
    NewLock(key string, options *LockOptions) (Locker, error)
    List(directory string) ([]*KVPair, error)
    DeleteTree(directory string) error
    AtomicPut(key string, value []byte, previous *KVPair, options *WriteOptions) (bool, *KVPair, error)
    AtomicDelete(key string, previous *KVPair) (bool, error)
    Close()
}
```

(ʘ_ʘ)

```
type NotVeryUseful interface{}
```

```go
func main() {
    var i interface{}
    i = 123      // valid
    i = "hello" // valid
    i = true     // valid
    fmt.Println(i)
}
```

```
func foo(v interface{}) {
    // ???
}
```

```
func foo(v interface{}) {
    i := v.(int)
    println("int", i)
}
```

```
func foo(v interface{}) {
    if i, ok := v.(int); ok {
        println("int", i)
    } else {
        println("not an int")
    }
}
```

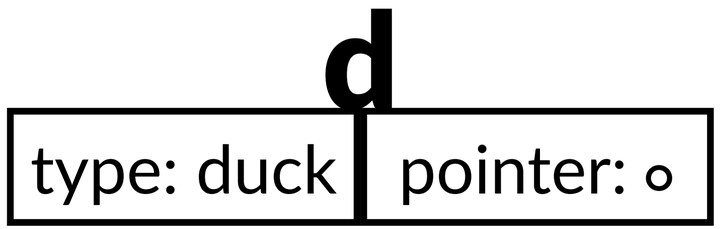
```go
func foo(v interface{}) {
    switch v := v.(type) {
    case int:
        println("int", v)
    case string:
        println("string", v)
     case io.Reader:
        println("io.Reader")
    default:
        println("unknown")
    }
}
```

```
func foo(v interface{}) {
    switch v := v.(type) {
    case int, float32:
        println("number", v)
    default:
        println("unknown")
    }
}
```

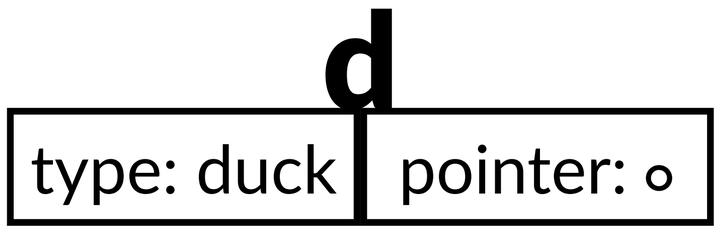
```
func foo(v interface{}) {
    switch v := v.(type) {
    case int, float32:
        println("number", v + v)
    default:
        println("unknown")
    }
}
```

This is not generics

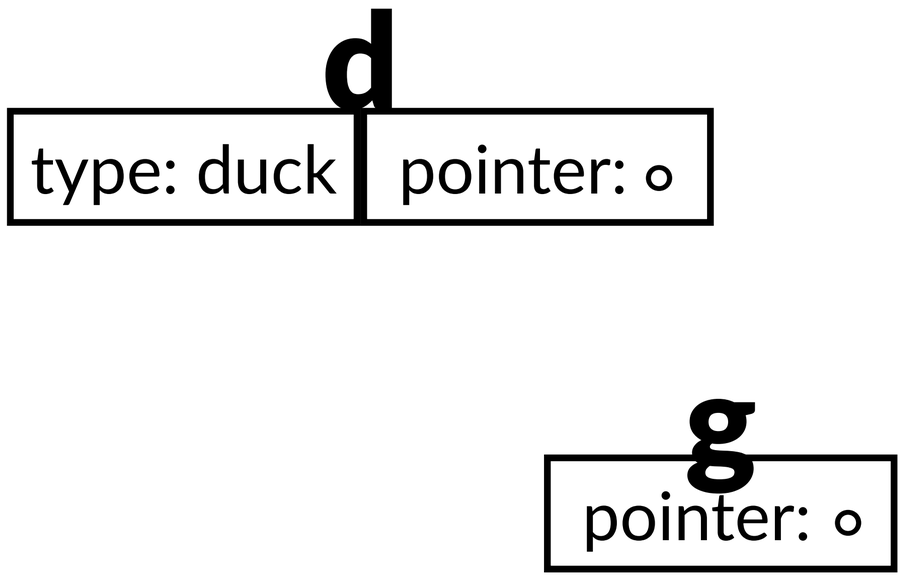
```
var d duck
```



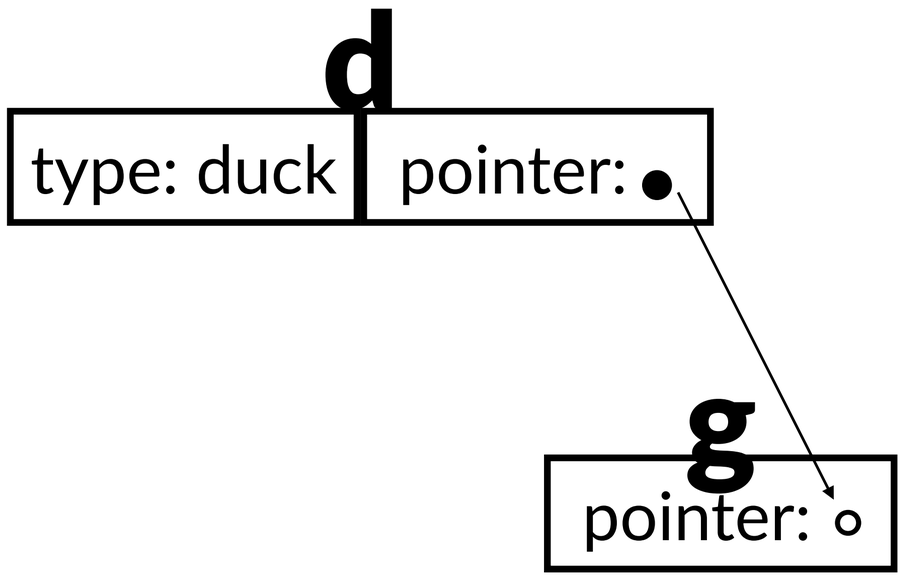
```
var d duck
//d.quack()
```



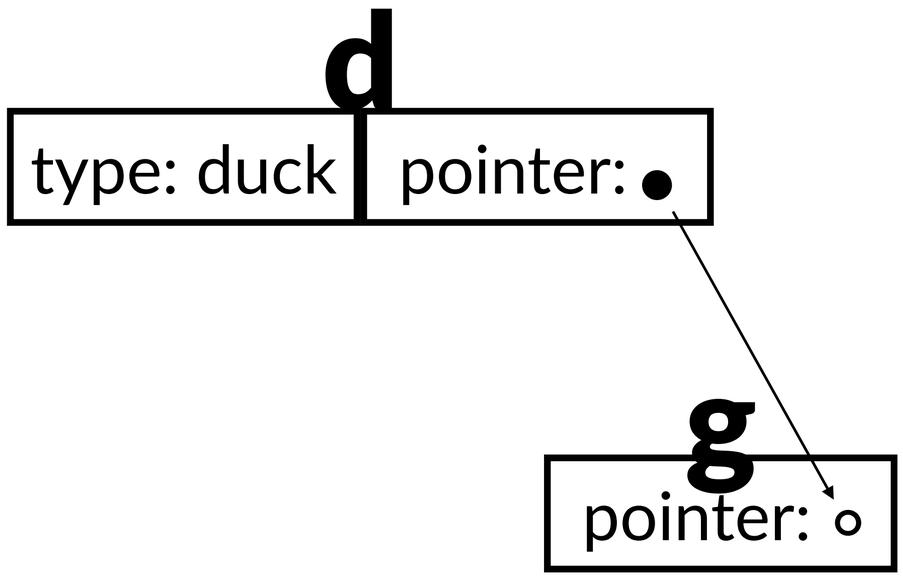
```
var d duck
//d.quack()
var g *goose
```



```
var d duck
//d.quack()
var g *goose
d = g
```



```
var d duck
//d.quack()
var g *goose
d = g
d.quack() // ?
```



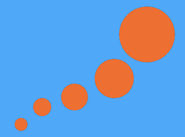
```go
func (g goose) quack() {
    // it's not possible to get here via a nil ptr
    // "value method main.goose.quack called using
    // nil *goose pointer"
}

func (g *goose) quack() {
    // it is possible to get here with a nil ptr!
    // g may be nil
}
```

11: Generator Interface

Using the solution to exercise 9 change Run to take a `Generator` interface

https://github.com/domdavis/go-get-better/tree/exercise-11

Interfaces and Testing



framework training

Whenever possible, write code in a functional style.
Take all dependencies as parameters.
Avoid depending on or (especially) mutating global state.
Make heavy use of interfaces!

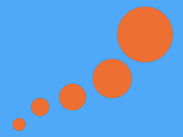
```go
package main

func process(db *database) (result, error) {
    var r result
    rows, err := db.Query("SELECT foo")

    if err != nil {
        return result{}, err
    }

    defer rows.Close()
    if err := rows.Scan(&r); err != nil {
        return result{}, err
    }

    return r, nil
}

func main() {
    db := newDatabase()
    r, err := process(db)
}
```

```go
package main

type queryer interface {
    Query(s string) (rows, error)
}

func process(db queryer) (result, error) {
    var r result
    rows, err := db.Query("SELECT foo")

    if err != nil {
        return result{}, err
    }

    defer rows.Close()
    if err := rows.Scan(&r); err != nil {
        return result{}, err
    }

    return r, nil
}

func main() {
    db := newDatabase()
    r, err := process(db)
}
```

```go
type fakeQueryer struct{}

func (q fakeQueryer) Query(s string) (rows, error) {
    return []row{"fakerow"}, nil
}
```

```go
type fakeQueryer struct{}

func (q fakeQueryer) Query(s string) (rows, error) {
    return []row{"fakerow"}, nil
}

func TestProcess(t *testing.T) {
    q := fakeQueryer{}
    have, err := process(q)

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

    want := result{"fakedata"} // or whatever

    if want != have {
        t.Errorf("process: want %v, have %v", want, have)
    }
}
```

Implementing Interfaces



framework training

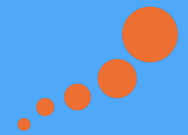
```go
type myType struct{}

func (_ myType) String() string {
    return "This is my type"
}
```

```
f() {
    sleep "$1"
    echo "$1"
}
while [ -n "$1" ]
do
    f "$1" &
    shift
done
wait
```

```go
package main

import (
    "fmt"
    "sort"
)

func main() {
    strings := []string{"c", "a", "b"}
    sort.Strings(strings)
    fmt.Println("Strings:", strings)

    ints := []int{7, 2, 4}
    sort.Ints(ints)
    fmt.Println("Ints:   ", ints)

    s := sort.IntsAreSorted(ints)
    fmt.Println("Sorted: ", s)
}

// Strings: [a b c]
// Ints:    [2 4 7]
// Sorted:  true
```

```go
package main

import (
    "fmt"
    "sort"
)

type ByLength []string

func (s ByLength) Len() int {
    return len(s)
}

func (s ByLength) Swap(i, j int) {
    s[i], s[j] = s[j], s[i]
}

func (s ByLength) Less(i, j int) bool {
    return len(s[i]) < len(s[j])
}

func main() {
    fruits := []string{"peach", "banana", "kiwi"}
    sort.Sort(ByLength(fruits))
    fmt.Println(fruits)
}
```

Generics

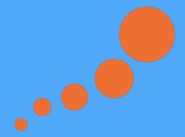


framework training

- Functions can have an additional type parameter list that uses square brackets but otherwise looks like an ordinary parameter list: func F[T any](p T) { ... }.
- These type parameters can be used by the regular parameters and in the function body.
- Types can also have a type parameter list: type M[T any] []T.
- Each type parameter has a type constraint, just as each ordinary parameter has a type: func F[T Constraint](p T) { ... }.
- Type constraints are interface types.
- The new predeclared name any is a type constraint that permits any type.
- Interface types used as type constraints can embed additional elements to restrict the set of type arguments that satisfy the contraint:
- an arbitrary type T restricts to that type
- an approximation element ~T restricts to all types whose underlying type is T
- a union element T1 | T2 | ... restricts to any of the listed elements
- Generic functions may only use operations supported by all the types permitted by the constraint.
- Using a generic function or type requires passing type arguments.
- Type inference permits omitting the type arguments of a function call in common cases.

```go
package main

import (
  "fmt"
  "math"
)

type Number interface {
  int | int8 | int16 | int32 | int64 | uint | uint8 | uint16 | uint32 |
    uint64 | float32 | float64
}

type myType[T Number] []T

func main() {
  i := myType[int]{1, 2, 3, 4}
  f := myType[float64](1.0, math.Pi)

  fmt.Println(i.Add(), f.Add())
}

func (m myType[T]) Add() T {
  var total T

  for _, v := range m {
    total += v
  }

  return total
}
```

JSON Parsing



framework training

```go
package main

import (
    "encoding/json"
    "fmt"
)

type Car struct {
    Make string
    Model string
    doors int
}

func main() {
    c := Car{Make: "Toyota", Model: "Corolla", doors: 5}
    b, _ := json.Marshal(c)
    fmt.Println(string(b))
}
```

```go
package main

import (
    "encoding/json"
    "fmt"
)

type Car struct {
    Make string
    Model string
    doors int
}

func main() {
    c := Car{Make: "Toyota", Model: "Corolla", doors: 5}
    b, _ := json.Marshal(c)
    fmt.Println(string(b))
}

// {"Make":"Toyota","Model":"Corolla"}
```

```go
package main

import (
    "encoding/json"
    "fmt"
)

type Car struct {
    Make string `json:"make"`
    Model string `json:"model"`
    Doors int `json:"doors"`
}

func main() {
    c := Car{Make: "Toyota", Model: "Corolla", Doors: 5}
    b, _ := json.Marshal(c)
    fmt.Println(string(b))

    var car Car
    _ = json.Unmarshal(b, &car)
    fmt.Println(car.Doors)
}

// {"make":"Toyota","model":"Corolla","doors":5}
// 5
```

```go
package main

import (
    "encoding/json"
    "fmt"
)

type Car struct {
    Make string `json:"make"`
    Model string `json:"model"`
    Doors int `json:"doors"`
}

func (c Car) MarshalJSON() ([]byte, error) {
    s := fmt.Sprintf("%d door %s %s", c.Doors, c.Make, c.Model)
    return json.Marshal(s)
}

func main() {
    c := Car{Make: "Toyota", Model: "Corolla", Doors: 5}
    b, _ := json.Marshal(c)
    fmt.Println(string(b))
}

// "5 door Toyota Corolla"
```

12: Sequence JSON

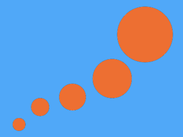
Update the sequence type so when it's rendered as JSON it's output in the format `{"name": ["1", "2",...]}`
An empty sequence should simple be `{}`

https://github.com/domdavis/go-get-better/tree/exercise-12

```go
func (s Sequence) MarshalJSON() ([]byte, error) {
    var sequence []string
    o := map[string][]string{}

    if len(s) > 2 {
        sequence = s[1:]
    }

    if len(s) > 1 {
        o[s[0]] = sequence
    }

    return json.Marshal(o)
}
```

```go
func ExampleSequence_MarshalJSON() {
    r, _ := training.Run(training.Simple{}, 5)

    if b, err := json.Marshal(r); err != nil {
        fmt.Println(err)
    } else {
        fmt.Println(string(b))
    }

    r, _ = training.Run(training.Simple{}, 0)

    if b, err := json.Marshal(r); err != nil {
        fmt.Println(err)
    } else {
        fmt.Println(string(b))
    }

    // Output:
    // {"Simple":["1","2","3","4","5"]}
    // {}
}
```

Concurrency & Parallelism



framework training

Concurrency != Parallelism

Concurrency is about designing your program so that multiple things can execute independently of each other.

Concurrency is about designing your program so that multiple things can execute independently of each other.
Parallelism is executing those things at the same time.

Concurrency is about designing your program so that multiple things can execute independently of each other.
Parallelism is executing those things at the same time.
Go programs should be written for concurrency, but parallelism is a decision for the runner!

```go
package main

func main() {
    foo("a")
    go foo("b")
}

func foo(s string) {
    println(s)
}
```

```go
package main

import "time"

func main() {
    foo("a")
    go foo("b")
    time.Sleep(time.Second)
}

func foo(s string) {
    println(s)
}
```

```go
package main

func main() {
    n := 10
    for i := 0; i < n; i++ {
        go println(i)
    }
    // when is everything done?
}
```

```go
package main

import "sync"

func main() {
    var wg sync.WaitGroup
    for i := 0; i < 10; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            print(i, " ")
        }()
    }

    wg.Wait()
}
```

```go
package main

import "sync"

func main() {
    var wg sync.WaitGroup
    for i := 0; i < 10; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            print(i, " ")
        }()
    }

    wg.Wait()
}
// 8 8 8 8 10 10 10 10 10 10
```

```go
package main

import "sync"

func main() {
    var wg sync.WaitGroup
    for i := 0; i < 10; i++ {
        wg.Add(1)
        go func(i int) {
            defer wg.Done()
            print(i, " ")
        }(i)
    }

    wg.Wait()
}
// 1 0 4 2 3 9 6 5 8 7
```

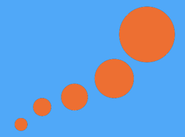
Mutex

```go
package main

type thing struct {
    m map[int]int
}

func newThing() *thing {
    return &thing{m: map[int]int{}}
}

func (t *thing) set(k, v int) {
    t.m[k] = v
}

func (t *thing) get(k int) int {
    return t.m[k]
}
```

```go
package main

import "sync"

type thing struct {
    mtx sync.Mutex
    m   map[int]int
}

func newThing() *thing {
    return &thing{m: map[int]int{}}
}

func (t *thing) set(k, v int) {
    t.mtx.Lock()
    defer t.mtx.Unlock()
    t.m[k] = v
}

func (t *thing) get(k int) int {
    t.mtx.Lock()
    defer t.mtx.Unlock()
    return t.m[k]
}
```

```go
package main

import "sync"

type thing struct {
    mtx sync.RWMutex
    m   map[int]int
}

func newThing() *thing {
    return &thing{m: map[int]int{}}
}

func (t *thing) set(k, v int) {
    t.mtx.Lock()
    defer t.mtx.Unlock()
    t.m[k] = v
}

func (t *thing) get(k int) int {
    t.mtx.RLock()
    defer t.mtx.RUnlock()
    return t.m[k]
}
```

Channels



framework training

Don’t communicate by sharing memory

Don’t communicate by sharing memory
Share memory by communicating

Don’t communicate by sharing memory
Share memory by communicating
A channel is like a UNIX pipe

Don’t communicate by sharing memory
Share memory by communicating
A channel is like a UNIX pipe
Typed conduit for information, typically between goroutines

```go
package main

func main() {
    c := make(chan int)
    go compute(c)
    println(<-c)
}

func compute(c chan int) {
    c <- 123
}
```

```go
package main

func main() {
    c := make(chan int, 100)
    go compute(c)
    println(<-c)
}

func compute(c chan int) {
    c <- 123
}
```

```go
func main() {
 c := make(chan int)
 for i := 0; i < 10; i++ {
  go compute(i, c)
 }
 for i := 0; i < 10; i++ {
  fmt.Print(i, <-c, ", ")
 }
}

func compute(id int, c chan int) {
 c <- id
}

// 0 1, 1 0, 2 3, 3 5, 4 4, 5 6, 6 2, 7 8, 8 9, 9 7,
```

```go
package main

func main() {
    c := make(chan int)
    for i := 0; i < 10; i++ {
        go readOne(c)
    }
    c <- 123
    c <- 456
    c <- 789
}

func readOne(c chan int) {
    println(<-c)
}
```

```go
package main

func main() {
    c := make(chan int)
    for i := 0; i < 10; i++ {
        go readOne(c)
    }
    c <- 123
    c <- 456
    c <- 789
    close(c)
}

func readOne(c chan int) {
    println(<-c)
}
```

```go
package main

func main() {
    c := make(chan int)
    for i := 0; i < 10; i++ {
        go readOne(c)
    }
    c <- 123
    c <- 456
    c <- 789
    close(c)
    time.Sleep(time.Second)
}

func readOne(c chan int) {
    println(<-c)
}
```

```
func readOne(c chan int) {
    v, ok := <-c
    if ok {
        println("received value", v)
    } else {
        println("channel was closed")
    }
}
```

```go
func main() {
    c := make(chan int)
    go read(c)
    c <- 123
    c <- 456
    c <- 789
    close(c)
    time.Sleep(time.Second)
}

func read(c chan int) {
    for v := range c {
        println("received value", v)
    }
}
```

Select

framework training

```go
func main() {
    c1, c2 := make(chan int), make(chan int)

    for i := 0; i < 10; i++ {
        go read(c1, c2)
    }

    c1 <- 123
    c2 <- 456
    close(c1)
    c2 <- 789
    close(c2)
    time.Sleep(time.Second)
}

func read(c1, c2 chan int) {
    select {
    case v := <-c1:
        fmt.Printf("Read from channel 1: %d\n", v)
    case v := <-c2:
        fmt.Printf("Read from channel 2: %d\n", v)
    }
}
```

```go
func read(c1, c2 chan int) {
    select {
    case v := <-c1:
        fmt.Printf("Read from channel 1: %d\n", v)
    case v := <-c2:
        fmt.Printf("Read from channel 2: %d\n", v)
    default:
        fmt.Println("No channel ready")
    }
}
```

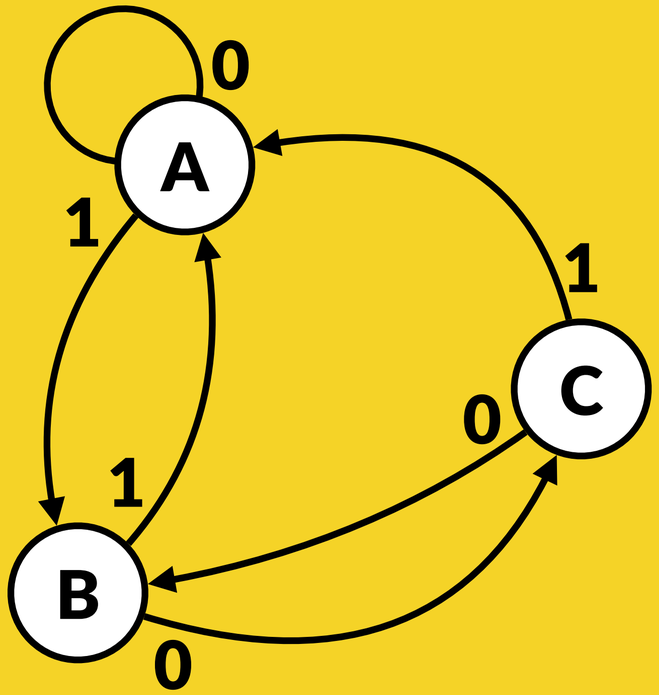
```
func doWork(i chan int, s chan string, t chan thing) {
    for {
        select {
        case v := <-i:
            println("do work with int", i)
        case v := <-s:
            println("do work with string", s)
        case v := <-t:
            println("do work with thing", t)
        }
    }
}
```

```go
func (a *Actor) SendEvent(e Event) {
    a.event <- e
}

func (a *Actor) SendReq(r *Request) {
    a.request <- r
}

func (a *Actor) Stop() {
    close(a.quit)
}

func (a *Actor) loop() {
    for {
        select {
        case e := <-a.event:
            // process event
        case r := <-a.request:
            // process request
        case <-a.quit:
            return
        }
    }
}
```

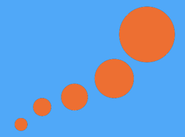
```
type Actor struct {
    event   chan Event
    request chan *Request
    quit    chan struct{}
}
```

13: State Machine

```
package main

// Implement stateMachine.
type stateMachine struct{}

func main() {
    sm := newStateMachine()
    sm.send(1)           // "state A + 1 => state B"
    sm.send(0)           // "state B + 0 => state C"
    println(sm.state()) // "state C"
}
```



Pipelines



framework training

```
$ echo "test" | grep -o . | sort
e
s
t
t
```

```go
package main

import "strings"

func trim(in chan string, out chan string) {
    for s := range in {
        s = strings.TrimSpace(s)
        out <- s
    }
}

func capitalize(in chan string, out chan string) {
    for s := range in {
        s = strings.ToUpper(s)
        out <- s
    }
}
```

```go
package main

import "strings"

func trim(in <-chan string, out chan<- string) {
    for s := range in {
        s = strings.TrimSpace(s)
        out <- s
    }
}

func capitalize(in <-chan string, out chan<- string) {
    for s := range in {
        s = strings.ToUpper(s)
        out <- s
    }
}
```

```go
func main() {
    a := make(chan string)
    b := make(chan string)
    c := make(chan string)

    go trim(a, b);
    go capitalize(b, c)

    a <- " hello world  "
    fmt.Printf("%q", <-c)

    close(a)
    close(b)
    close(c)
}
```

```go
func main() {
    a := make(chan string)
    b := make(chan string)
    c := make(chan string)

    go trim(a, b); go trim(a, b); go trim(a, b); go trim(a, b)
    go capitalize(b, c)

    a <- " hello world  "
    fmt.Printf("%q", <-c)

    close(a)
    close(b)
    close(c)
}
```

14: FizzBuzz Pipelines

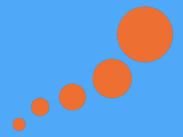
Checkout and finish the code in the branch pipelines

https://github.com/domdavis/go-get-better/tree/pipelines
https://github.com/domdavis/go-get-better/tree/exercise-14

```
func fizz(in <-chan int, out chan<- string) {
    if <-in%3 == 0 {
        out <- "Fizz"
    } else {
        out <- ""
    }
}

func buzz(in <-chan int, out chan<- string) {
    if <-in%5 == 0 {
        out <- "Buzz"
    } else {
        out <- ""
    }
}

func number(in <-chan int, out chan<- string) {
    i := <-in
    if i%3 != 0 && i%5 != 0 {
        out <- strconv.Itoa(i)
    } else {
        out <- ""
    }
}
```

```
var FizzBuzz = Generator{
    Name:  "FizzBuzz",
    Steps: []Step{fizz, buzz, number},
}
```

Parallelism



framework training

goroutines are multiplexed onto OS threads

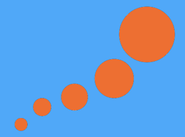
goroutines are multiplexed onto OS threads
GOMAXPROCS controls number of threads available

goroutines are multiplexed onto OS threads
GOMAXPROCS controls number of threads available
By default, GOMAXPROCS = num CPUs

Playtime

https://divan.github.io/posts/go_concurrency_visualize/

Networking



framework training

```go
package main

import (
    "fmt"
    "io/ioutil"
    "net/http"
)

func main() {
    resp, err := http.Get("http://google.com")
    if err != nil {
        panic(err)
    }

    defer func() { _ = resp.Body.Close() }()

    if b, err := ioutil.ReadAll(resp.Body); err != nil {
        panic(err)
    } else {
        fmt.Println(string(b))
    }
}
```

```go
package main

import (
    "io"
    "net/http"
    "os"
)

func main() {
    resp, err := http.Get("http://google.com")
    if err != nil {
        panic(err)
    }

    defer func() { _ = resp.Body.Close() }()

    if _, err := io.Copy(os.Stdout, resp.Body); err != nil {
        panic(err)
    }
}
```

```go
package main

import (
    "io"
    "net/http"
    "os"
)

func main() {
    c := http.Client{} // zero value is usable

    req, err := http.NewRequest("GET", "http://google.com", nil)
    if err != nil {
        panic(err)
    }

    resp, err := c.Do(req)
    if err != nil {
        panic(err)
    }

    defer func() { _ = resp.Body.Close() }()

    if _, err := io.Copy(os.Stdout, resp.Body); err != nil {
        panic(err)
    }
}
```

```go
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", h)
    if err := http.ListenAndServe(":8080", nil); err != nil {
        panic(err)
    }
}

func h(w http.ResponseWriter, r *http.Request) {
    _, _ = fmt.Fprintf(w, "Hello world\n")
}
```

```go
package main

import (
    "fmt"
    "log"
    "net/http"
)

func main() {
    s := &server{msg: "Hello from server"}
    http.Handle("/", s)
    if err := http.ListenAndServe(":8080", nil); err != nil {
        panic(err)
    }
}

type server struct{ msg string }

func (s *server) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    log.Printf("%s %s from %s", r.Method, r.URL, r.RemoteAddr)
    _, _ = fmt.Fprintf(w, s.msg+"\n")
}
```

```go
package main

import (
    "fmt"
    "log"
    "net/http"
)

func main() {
    mux := http.NewServeMux()
    mux.HandleFunc("/foo", handleFoo)
    mux.HandleFunc("/bar", handleBar)
    log.Fatal(http.ListenAndServe(":8080", mux))
}

func handleFoo(w http.ResponseWriter, r *http.Request) {
    log.Printf("%s %s from %s", r.Method, r.URL, r.RemoteAddr)
    _, _ = fmt.Fprintf(w, "foo\n")
}

func handleBar(w http.ResponseWriter, r *http.Request) {
    log.Printf("%s %s from %s", r.Method, r.URL, r.RemoteAddr)
    _, _ = fmt.Fprintf(w, "bar\n")
}
```

Other Routers and Mixers

Usability: github.com/gorilla/mux
Raw speed: github.com/julienschmidt/httprouter

```go
func main() {
    ln, err := net.Listen("tcp4", ":1234")
    if err != nil {
        panic(err)
    }

    defer func() { _ = ln.Close() }()

    for {
        c, err := ln.Accept()
        if err != nil { break }
        go handle(c)
    }
}

func handle(c net.Conn) {
    log.Printf("%s: start conn", c.RemoteAddr())
    defer log.Printf("%s: close conn", c.RemoteAddr())

    s := bufio.NewScanner(c)
    for s.Scan() {
        log.Printf("%s: %s", c.RemoteAddr(), s.Text())
    }
}
```

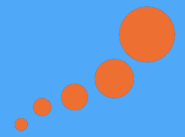
```go
func main() {
    ln, err := net.Listen("tcp4", ":1234")
    if err != nil {
        panic(err)
    }

    defer func() { _ = ln.Close() }()

    for {
        c, err := ln.Accept()
        if err != nil { break }
        go handle(c)
    }
}

func handle(c net.Conn) {
    log.Printf("%s: start conn", c.RemoteAddr())
    defer log.Printf("%s: close conn", c.RemoteAddr())

    s := bufio.NewScanner(c)
    for s.Scan() {
        log.Printf("%s: %s", c.RemoteAddr(), s.Text())
        _, _ = fmt.Fprintf(c, "%s\n", strings.ToUpper(s.Text()))
    }
}
```

15: FizzBuzz Microservice

Write an implementation off FizzBuzz that will respond to a HTTP request to `fizzbuzz/n`

https://github.com/domdavis/go-get-better/tree/exercise-15

Packaging and Distribution



framework training

go mod

```
$ go get github.com/domdavis/go-get-better

# before modules was roughly equivalent to...

$ cd $GOPATH/src/github.com/go-get-better
$ git clone https://github.com/domdavis/go-get-better
$ go install github.com/domdavis/go-get-better

# with modules go get is used to add a package

$ go get -u github.com/domdavis/go-get-better
```


https://golang.org/pkg/bytes/
Hamburger
The Project
Help
Blog
Play
Search
Package bytes
import "bytes"
Overview
Index
Examples
Overview
Package bytes implements functions for the manipulation of byte slices. It is analogous to the facilities of the strings package.
Index
Constants
Variables
func Compare(a, b []byte) int
func Contains(b, subslice []byte) bool
func Count(s, sep []byte) int
func Equal(a, b []byte) bool
func EqualFold(s, t []byte) bool
func Fields(s []byte) [][]byte
func FieldsFunc(s []byte, f func(rune) bool) [][]byte

```
FROM scratch
ADD app /
ENTRYPOINT ["/app"]
CMD ["args"]
```

```
CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .
docker build -t image:latest .
```

Go Nuts

framework training

business value through education

Welcome Back

framework training

business value through education