

git & golang book



Read golang book and work with git



golang book

AN INTRODUCTION TO
PROGRAMMING
IN **GO**



CALEB DOXSEY



git

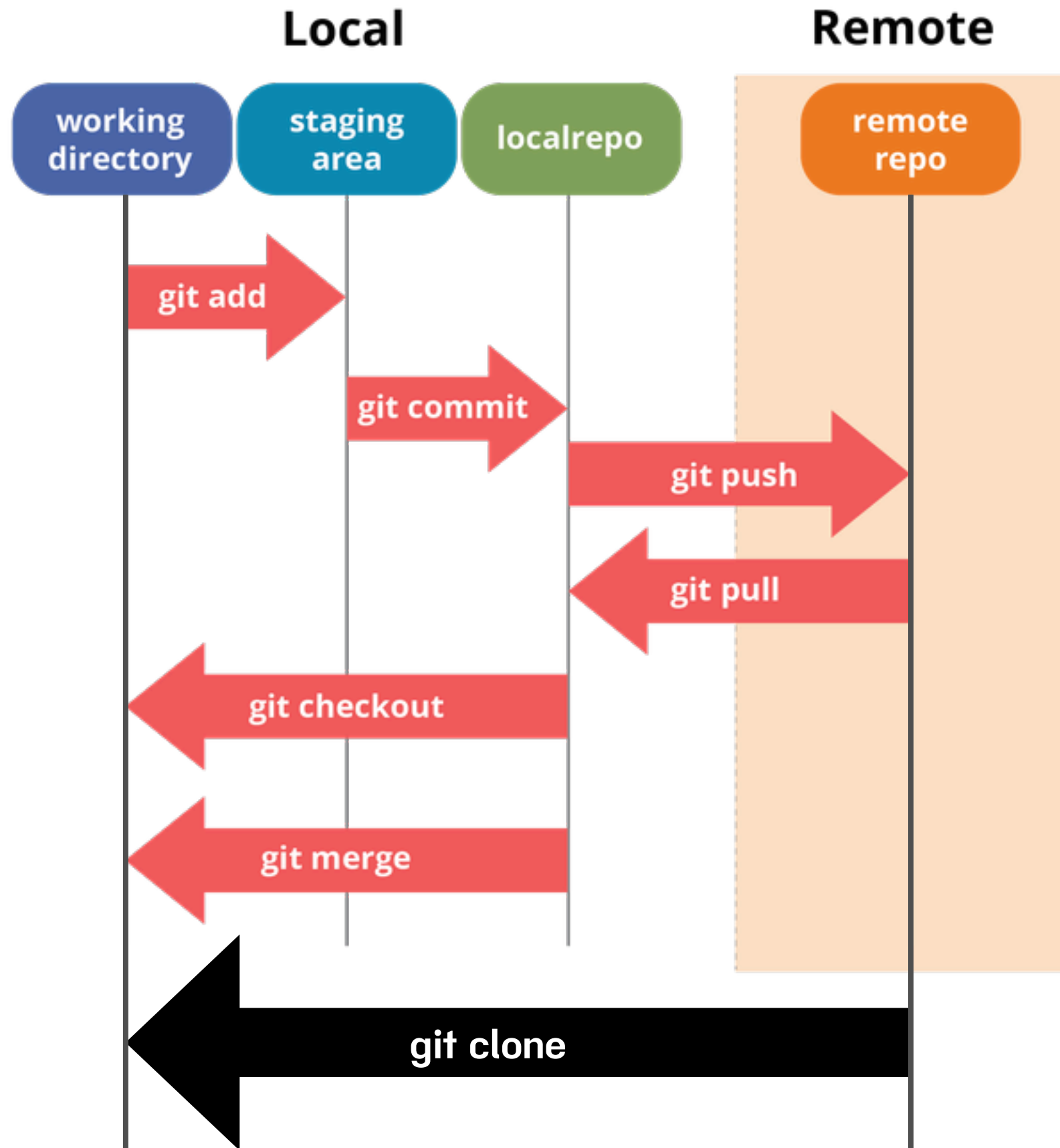
github account:

`https://github.com/[yourname]`

example:

`https://github.com/boyone`





Clone go-101

clone go-101 to your workspace:

```
>git clone https://github.com/boyone/go-101.git
```



golang book [1]

make working directory: [windows]

```
>md src\dojo\golang-book
```

make working directory: [linux/Mac]

```
>mkdir -p src/dojo/golang-book
```



golang book [2]

go to golang-book directory:

```
>cd src\dojo\golang-book
```

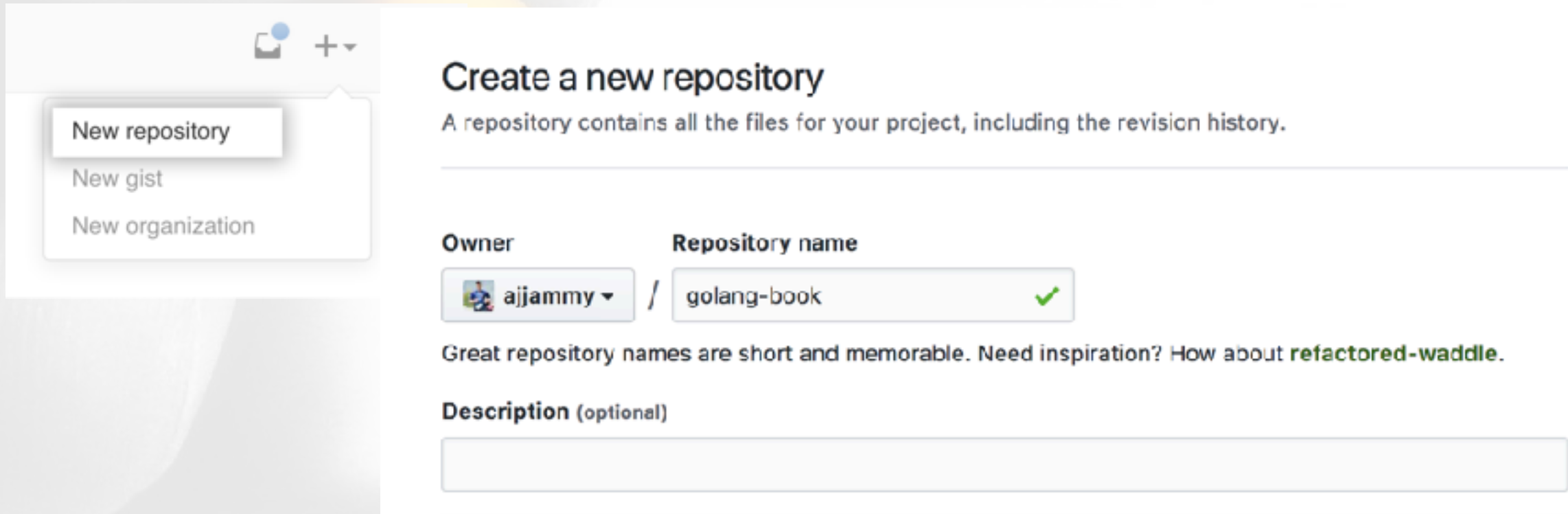
git init for golang-book directory:

```
>git init
```



golang book [3]

create github repository:



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: ajjammy / Repository name: golang-book ✓

Great repository names are short and memorable. Need inspiration? How about **refactored-waddle**.

Description (optional)

add remote:

```
>git remote add origin https://github.com/<user>/golang-book.git  
>git remote -v
```



golang book [4]

create README.md file:

```
① README.md x
1  # Go Book
2
3  **Name:** *Chamnan Inta*
4
5  **Nickname:** *Jammy*
6
7  **Job Title:** *Programmer*
8
9  ## Chapter 2
10
11 ## Chapter 3
12
13 ## Chapter 4
```



golang book [5]

git add / git commit :

```
>git add README.md  
>git commit -m "Add README.md file"
```

git push

```
>git push -u origin master
```



golang book [6]

create .gitignore file:

```
.gitignore x
1 *.exe
2 *.DS_Store
```



golang book [7]

git add / git commit :

```
>git add .gitignore  
>git commit -m "Add .gitignore file"
```

git push

```
>git push
```



golang book [8]

1. Read book chapter 2
2. Update README.md
3. Create main.go file at directory golang-book/chapter2-1

```
1  # Go Book
2
3  **Name:** *Chamnan Inta*
4
5  **Nickname:** *Jammy*
6
7  **Job Title:** *Programmer*
8
9  ## Chapter 2
10
11  * chapter2-1 : My First Program
12
13  ## Chapter 3
```

```
1  package main
2
3  import "fmt"
4
5  // this is a comment
6  func main() {
7      fmt.Println("Hello World")
8  }
9
```



golang book [9]

git add / git commit :

```
>git add .  
>git commit -m "Add chapter2-1 My first program"
```

git push

```
>git push
```



add collaborators

ajjammy / golangbook



Unwatch 3 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Options

- Collaborators**
- Branches
- Webhooks
- Integrations & services
- Deploy keys

Collaborators Push access to the repository

	Thawatchai Jongsuwanpaisan boyone	×
	ployploy	×

Search by username, full name or email address

You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

ajjammy

ajjammy Jammy

Add collaborator



Exercise



Read book
Chapter 1 to 4



Create folder chapter<...>-...
Create file main.go
Update README.md file

```
git add  
git commit  
git push
```



Type



GO



golang : type Zero Value

```
package main

import "fmt"

func main() {
    fmt.Println("====Zero Value====")
    var number int
    var str string
    var boolean bool
    fmt.Printf("number: %v\n", number)
    fmt.Printf("str: '%v'\n", str)
    fmt.Printf("boolean: %v\n", boolean)
}
```



golang : type Strings

```
package main

import "fmt"

func main() {
    fmt.Println("====String====")
    backticks := `hello world!,
today's good day.`
    fmt.Println(backticks)

    doubleQuotes := "hello world!,\ntoday's good day."
    fmt.Println(doubleQuotes)
}
```



golang : type Floating point [1]

```
package main

import "fmt"

func main() {
    fmt.Println("====Floating point====")
    third := 1.0 / 3.0
    fmt.Printf("third = %v\n", third)
    fmt.Printf("third + third + third = %v\n", third+third+third)
}
```



golang : type Floating point [2]

```
package main

import "fmt"

func main() {
    fmt.Println("====Comparing floating point====")
    fmt.Println("0.1 + 0.2 == 0.3 is", 0.1+0.2 == 0.3)
    num := 0.1
    num += 0.2
    fmt.Println("num == 0.3 is", num == 0.3)
    fmt.Println("num is", num)
}
```



Variables



GO



golang : Variables [1]

create main.go in folder chapter4-1 :

```
package main

import "fmt"

func main() {

}
```

run -> no error -> push to your git repository



golang : Variables [2]

create main.go in folder chapter4-2 :

```
package main

import "fmt"

func main() {
    fmt.Print("Enter a number: ")
    var input float64
    fmt.Scanf("%f", &input)
    output := input * 2
    fmt.Println(output)
}
```

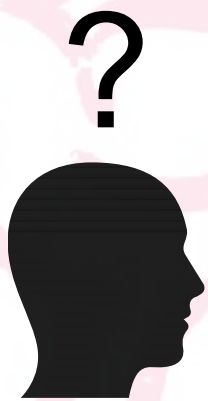
run -> no error -> push to your git repository



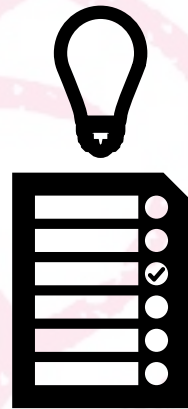
Exercise

Modify main.go in folder chapter4-2 for solve

Problem No.5 of Chapter 4 :



1



2



3



4

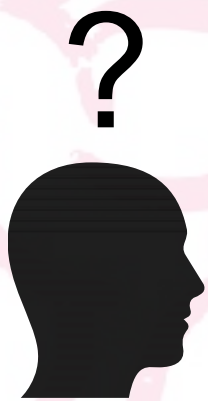
run -> no error -> push to your git repository



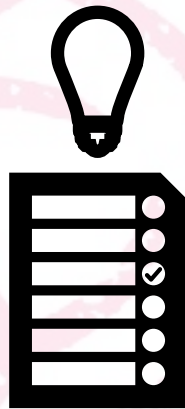
Exercise

Create main.go in folder chapter4-3 for solve

Problem No.6 of Chapter 4 :



1



2



3



4

run -> no error -> push to your git repository



Conditions



GO



golang : Conditions

create main.go in folder chapter5-1 :

```
package main

import "fmt"

func main() {
    fmt.Println("1")
    fmt.Println("2")
    fmt.Println("3")
    fmt.Println("4")
    fmt.Println("5")
    fmt.Println("6")
    fmt.Println("7")
    fmt.Println("8")
    fmt.Println("9")
    fmt.Println("10")
}
```



golang : Conditions [for]

```
package main

import "fmt"

func main() {
    number := 1
    for number <= 10 {
        fmt.Println(number)
        number = number + 1
    }
}
```



golang : Conditions [if]

create main.go in folder chapter5-2 :

```
package main

import "fmt"

func main() {
    for number := 1; number <= 100; number++ {
        if number%15 == 0 {
            fmt.Println(number, "FizzBuzz")
        } else if number%3 == 0 {
            fmt.Println(number, "Fizz")
        } else if number%5 == 0 {
            fmt.Println(number, "Buzz")
        } else {
            fmt.Println(number)
        }
    }
}
```



golang : Conditions [switch case]

create main.go in folder chapter5-3 :

```
package main

import "fmt"

func main() {
    switch i := 5 {
        case 0:
            fmt.Println("Zero")
        case 1:
            fmt.Println("One")
        case 2:
            fmt.Println("Two")
        case 3:
            fmt.Println("Three")
        case 4:
            fmt.Println("Four")
        case 5:
            fmt.Println("Five")
        default:
            fmt.Println("Unknown Number")
    }
}
```



Exercise

create exercise.go in folder chapter5-4 :

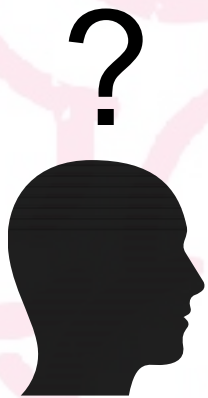
โปรแกรมจะให้ใส่ตัวเลขได้ไม่เกิน 5 ครั้ง

ถ้าเจอตัวเลขที่สุ่มมาจะแสดงคำว่า เจอแล้ว และจบการทำงาน

ถ้าเลขที่ใส่มากกว่าจะแสดงคำว่า มากไป

ถ้าเลขที่ใส่น้อยกว่าจะแสดงคำว่า น้อยไป

ถ้าใส่เกิน 5 ครั้งจะแสดงคำว่า เกินพอ และจบการทำงาน



1



2



3



4

run -> no error -> push to your git repository



function



GO



golang : function

```
package main

func main() {
}

func f() {
}

func fWithReturn(i int) int {
    return i
}

func fWithMultipleReturn(i int, s string) (int, error) {
    return i, nil
}
```



golang : function

create main.go in folder chapter6-1 :

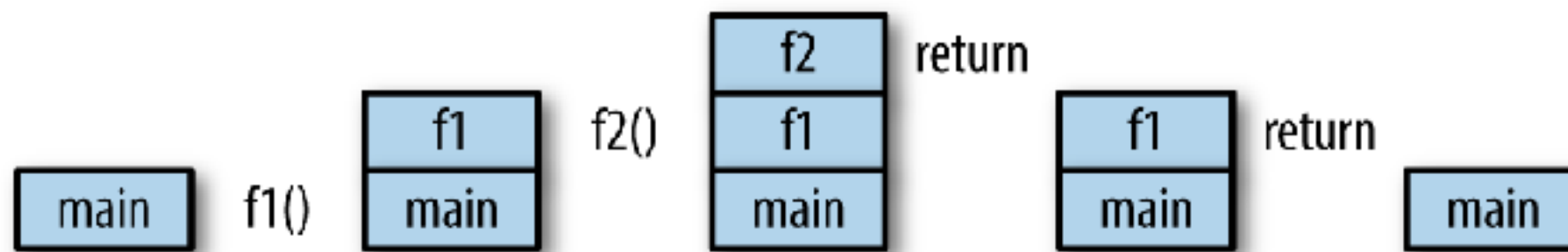
```
package main

import "fmt"

func main() {
    fmt.Println(f1())
}

func f1() int {
    return f2()
}

func f2() int {
    return 1
}
```



golang : function

create main.go in folder chapter6-2 :

```
package main

import "fmt"

func main() {
    fmt.Println(f2())
}

func f2() (r int) {
    r = 1
    return
}
```

Return types can have names



golang : function

create main.go in folder chapter6-3 :

```
package main

import "fmt"

func main() {
    x, y := f()
    fmt.Println(x, y)
}

func f() int, int {
    return 5, 7
}
```

Multiple values can be returned



golang : function

create main.go in folder chapter6-4 :

```
package main

import "fmt"

func main() {
    fmt.Println(add(1,2,3))

    xs := []int{1,2,3}
    fmt.Println(add(xs...))
}

func add(args ...int) int {
    total := 0
    for _, v := range args {
        total += v
    }
    return total
}
```

Variadic Function



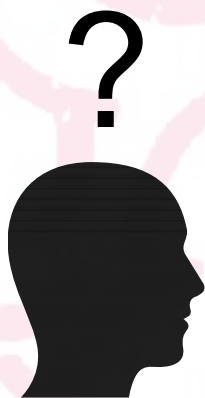
Exercise

create exercise.go in folder chapter6-5 :

REFACTOR FIZZBUZZ ใน CHAPTER5-2

ให้เรียกใช้งาน FUNCTION

แทนที่จะทำงานทุกอย่างใน FUNC MAIN



1



2



3



4

run -> no error -> push to your git repository

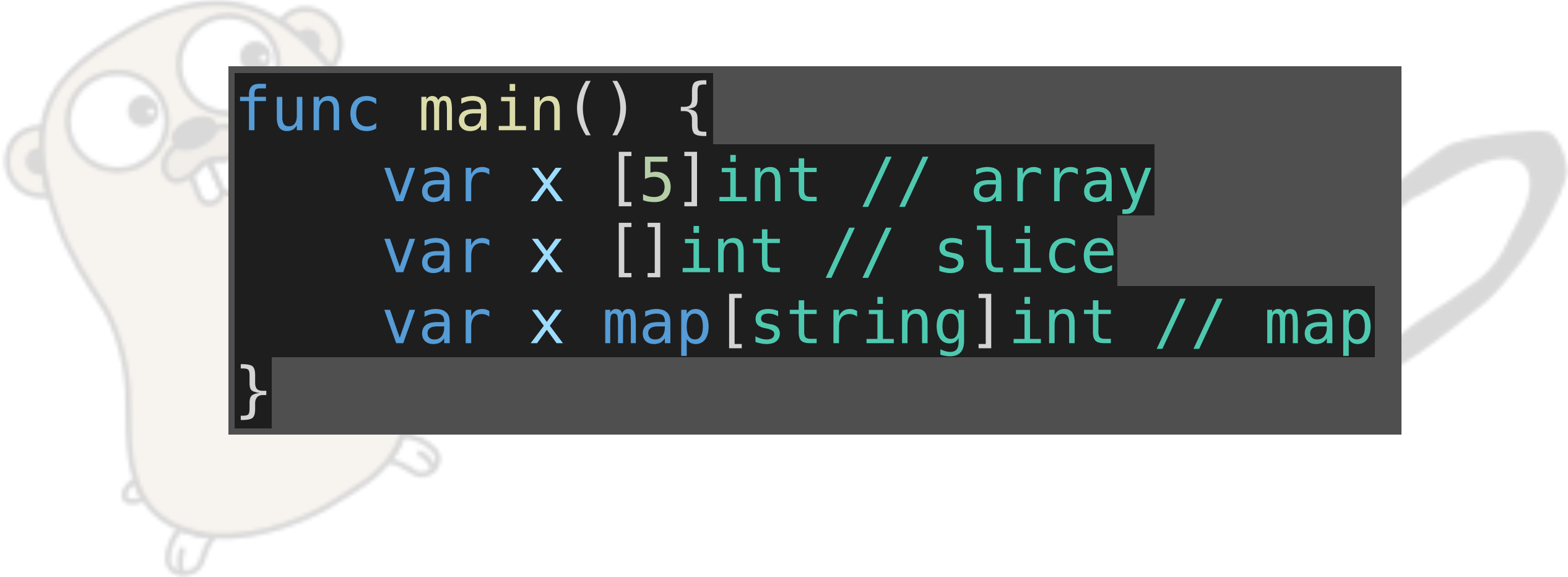


Collections:

Arrays, Slices, Map



golang : collections



```
func main() {  
    var x [5]int // array  
    var x []int // slice  
    var x map[string]int // map  
}
```



golang : arrays

create main.go in folder chapter7-1 :

```
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)
}
```



golang : slice

create main.go in folder chapter7-2 :

```
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)
}
```



golang : slice

create main.go in folder chapter7-3 :

```
package main

import "fmt"

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

    slide := arr[0:3]
    fmt.Println(slide)
}
```

Create slice from array



golang : slice

create main.go in folder chapter7-4 :

```
package main

import "fmt"

func main() {
    slice := []int{1, 2, 3}
    fmt.Println(slice)
    newSlice := make([]int, 2)
    fmt.Println(newSlice)
    copy(slice, newSlice)
    fmt.Printf("slice: %v\n", slice)
    fmt.Printf("slice: %v\n", newSlice)
}
```

Copy slices



golang : map

create main.go in folder chapter7-5 :

```
package main

import "fmt"

func main() {
    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)
}
```



golang : map

create main.go in folder chapter7-6 :

```
package main

import "fmt"

func main() {
    x := map[string]int{
        "one": 1,
        "two": 2,
        "three": 3,
    }
    fmt.Println(x)

    delete(x, "two")
    fmt.Printf("After delete: %v\n", x)
}
```

Delete map



golang : map

create main.go in folder chapter7-7 :

```
package main

import "fmt"

func main() {
    mymap := make(map[int]int)
    mymap[1] = 1
    mymap[2] = 2

    fmt.Println(mymap[3])
    if mymap[3] != 0 {
        fmt.Println(mymap[3])
    }

    // ok?
    if value, ok := mymap[3]; ok {
        fmt.Println(value)
    }
}
```

Avoid to check zero value



golang : range and collections

create main.go in folder chapter7-8 :

```
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)
    }
}
```

Range: Array



golang : range and collections

create main.go in folder chapter7-9 :

```
package main

import "fmt"

func main() {
    slice := []int{1, 2, 3, 4, 5}
    for i, number := range slice {
        fmt.Println(i, number)
    }
}
```

Range: Slice



golang : range and collections

create main.go in folder chapter7-10 :

```
package main

import "fmt"

func main() {
    maps := map[string]int{
        "one": 1,
        "two": 2,
        "three": 3,
    }

    for key, number := range maps {
        fmt.Println(key, number)
    }
}
```

Range: Map



golang : range and collections

create main.go in folder chapter7-11 :

```
package main

import "fmt"

func main() {
    for i, c := range "golang" {
        fmt.Println(i, c)
        fmt.Printf("%v\n", string(c))
    }
}
```

Range: String



Pointers



==GO

