image

Go 101

Pallat Anchaleechamaikorn

> Technical Coach
> Infinitas by KrungThai
> Arise by Infinitas

yod.pallat@gmail.com
https://github.com/pallat
https://dev.to/pallat

https://go.dev/tour (Thai)
https://github.com/uber-go/guide (Thai)

image

Vision

Gopher army

image

Getting Started

https://go.dev/

image

Go vs Golang

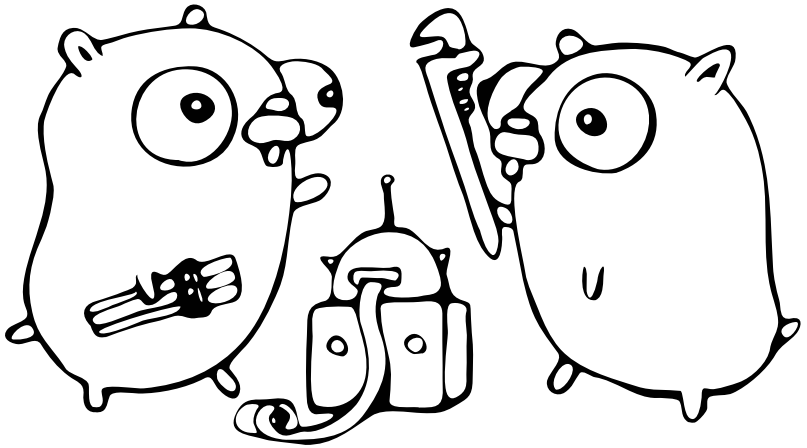

image created by *Pallat Anchaleechamaikorn* **© 2023**

Wiki

https://en.wikipedia.org/wiki/Go_(programming_language)

About Go

image

Go Users

https://github.com/golang/go/wiki/GoUsers

image

Installation

https://dev.to/pallat/install-go-4a1a

image

Download

https://go.dev/

OS Environment

.profile .zshrc

```
GOROOT=$HOME/{go package}
GOPATH=$HOME/go
GOBIN=$GOPATH/bin
PATH=$GOROOT/bin:$GOBIN:$PATH
```

```
GOROOT=/Users/pallat/sdk/go
GOPATH=/Users/pallat/go
GOBIN=$GOPATH/bin
PATH=$GOROOT/bin:$GOBIN:$PATH
```

Go Toolchain: version

```
go version
```

print Go version

image

Go Toolchain: env

```
go env
```

print Go environment information

Visual Studio Code

The VS Code Go Extension

Initial a project

linux/Macbook

```
mkdir hello && cd hello
```

windows

```
md hello
cd hello
```

Open VS Code

```
code .
```

Initial go module

```
go mod init hello
```

or

```
go mod init github.com/pallat/hello
```

go.mod

```
module hello

go 1.20
```

Hello World

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, สวัสดี")
}
```

```
go run main.go
```

image

package main

The package “*main*” tells the **Go** compiler that the *package* should compile as an executable program instead of a shared library

image

func main()

The main function in the package “**main**” will be the entry point of our executable program

image

go run main.go

compile and run Go program

image

Keywords: 3/25

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

Variable declaration: explicit type

```
var s string     // s = ""
var i int        // i = 0
var ok bool      // ok = false
var f float64    // f = 0.0
```

image

Zero Value

Declaration with initial value

```
var s string = "Hello World"
var i int = 9
var ok bool = true
var f float64 = 1
```

Type inference

```
var s = "Hello World"
var i = 9
var ok = true
var f = 1.0
```

image

Type inference without var keyword

```
s := "Hello World"
i := 9
ok := true
f := 1.0
```

Only in Functions

`_` (underscore) blank identifier

to define and use the unused variable

```
var n int 9
_ = n
```

Pointer

Pointer is

Go has pointers. A pointer holds the memory address of a value.

```go
var p *int
i := 42
p = &i

fmt.Println(*p, i)

*p = 43
fmt.Println(*p, i)
```

image

Pointer variable

```go
var p *int  // p = nil
```

image

pointer zero value

```
var p *int
```

image

new

```
var p = new(int)
```

image

Question: pointer

```
b := 10
a := &b

*a = 20

fmt.Println(*a, b)
```

Correct

```
b := 10
a := &b

*a = 20

fmt.Println(*a, b)
```

```
20 20
```

Question: pointer

```go
var a = new(int)

b := *a

*a = 10

fmt.Println(*a, b)
```

Correct: pointer

```go
var a = new(int)

b := *a

*a = 10

fmt.Println(*a, b)
```

```
10 0
```

Question: pointer

```go
var a **int
var b *int
var c int

c = 10
b = &c
a = &b

c = **a + *b + c
fmt.Println(c)
```

Correct: pointer

```go
var a **int
var b *int
var c int

c = 10
b = &c
a = &b

c = **a + *b + c
fmt.Println(c)
```

```
30
```

Question: pointer

```go
var a **int
var b *int
var c int

c = 10
b = &c
a = &b

c, d, e := **a + *b, *b + c, c + 10
fmt.Println(c, d, e)
```

Correct: pointer

```go
var a **int
var b *int
var c int

c = 10
b = &c
a = &b

c, d, e := **a + *b, *b + c, c + 10
fmt.Println(c, d, e)
```

```
20 20 20
```

Zero Value with pointer

image

Play with variables

Follow this example

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello.")
    fmt.Print("What is your name?: ")

    var name string
    fmt.Scanln(&name)

    fmt.Printf("Hi %s.\n", name)
}
```

Exercise: variable

```
Hello.
What is your first name?: [first name]
What is your last name?: [last name]
---
Hello [first name] [last name]. Nice to meet you.
```

image

func

```go
func add(a int, b int) int {
    return a + b
}

func add(a, b int) int {
    return a + b
}
```

image

Exercise - Area of a Square Function

```
func squareArea(a float64) float64 {

}
```

> example: squareArea(4) = 16

Exercise - Greeting

```go
func greet(name string) string {

}
```

> example: greet("Gopher") = "Hello, Gopher"

image

Functions with no return

```
func printAdded(a, b int) {
    fmt.Println(a + b)
}
```

Functions multiple return values

```go
func div(a, b int) (int,bool) {
    c := a/b // / is div
    d := a%b // % is mod
    return c, d == 0
}
```

image

Named return values

```go
func add(a, b int) (result int) {
    result = a + b
    return
}
```

Play with func (multiple return values)

```
func divmod(n1, n2 int) (quotient, remainder int) {

}
```

function with pointer parameters

```go
func main() {
    s := "Hi "
    appendString(&s, "Arise")

    fmt.Println(s)
}

func appendString(p *string,s string) {
    *p += s
}
```

copy values into arguments

```go
func main() {
    var x int
    fmt.Printf("%p %v\n", &x, x)
    function(x)
}

func function(x int) {
    fmt.Printf("%p %v\n", &x, x)
}
```

copy values into arguments

```go
func main() {
    var x int
    fmt.Printf("%p %v\n", &x, x)
    function(x)
}

func function(x int) {
    fmt.Printf("%p %v\n", &x, x)
}
```

```
0xc0000b2000 0
0xc0000b2008 0
```

copy values into arguments: pointer

```go
func main() {
    var x int
    var p *int
    p = &x
    fmt.Printf("%p %p %v\n", &p, p, *p)
    function(&x)
}

func function(p *int) {
    fmt.Printf("%p %p %v\n", &p, p, *p)
}
```

copy values into arguments: pointer

```go
func main() {
    var x int
    var p *int
    p = &x
    fmt.Printf("%p %p %v\n", &p, p, *p)
    function(&x)
}

func function(p *int) {
    fmt.Printf("%p %p %v\n", &p, p, *p)
}
```

```
0xc000012028 0xc00001c030 0
0xc000012038 0xc00001c030 0
```

image

exercise: function and pointer

```go
func main() {
    i := 10
    add(&i, i)
    fmt.Println(i)
    // answer = 20
}
```

exercise: function and pointer(2)

```
func main() {
    s := "Arise"
    format("Hi, ",&s,". How are you?")
    fmt.Println(s)
    // answer = ""Hi, Arise. How are you?"
}
```

`{` unexpected newline

for example

```
func add(a, b int)
{

}
```

```
func add(a, b int) {

}
```

image

most semicolons are optional and can be omitted

```
package main;

import "fmt";

func main() {
    var (
        i   int;
        sum int;
    );
    for i < 6 {
        sum += i;
        i++;
    };
    fmt.Println(sum);
};
```

https://go.dev/ref/spec#Semicolons

Control Flow if/else

```
if a != b {
    println("a not equal to b")
} else if a < b {
    println("a less than b")
} else {
    println("ok")
}
```

Play with if/else: tell me my generation

```
func generation(age uint) string
```

Generaion	Range
Z	1997 - 2012
Millennials	1981 – 1996
X	1965 – 1980
Boomers II (a/k/a Generation Jones)	1955 – 1964

> generation(44) == "X"

Control Flow if/else with statement

```
ok := IsOK()
if ok {
    println("It's correct")
}

if ok := IsOK(); ok {
    println("It's correct")
}

if ok := IsOK()
ok {
    println("It's correct")
}
```

Control Flow if/else with statement: example

```
if n, err := strconv.Atoi("5"); err != nil {
    log.Println(err)
} else {
    log.Printf("the number is %d\n", n)
}
```

image

image

Variable Scoping and Variable Shadowing

```go
var a int

func scope() {
    fmt.Println(a)
    a := 1
    fmt.Println(a)
    {
        a := 2
        {
            fmt.Println(a)
            a := 3
            fmt.Println(a)
        }
        fmt.Println(a)
    }
    fmt.Println(a)
}
```

image

Variable Scoping and Variable Shadowing

```go
var a int

func scope() {
    fmt.Println(a)              // 0
    a := 1
    fmt.Println(a)              // 1
    {
        a := 2
        {
            fmt.Println(a)      // 2
            a := 3
            fmt.Println(a)      // 3
        }
        fmt.Println(a)          // 2
    }
    fmt.Println(a)              // 1
}
```

Test

```go
func atLeastTen(n int) int {
    if n := n; n < 10 {
        n += 10
    }
    return n
}
```

> atLeastTen(2) == ?

Test

```
a, b := 1, 1
a, b = b, a + b
```

a = ?

b = ?

image

```go
func sum(leftOperand, rightOperand string) int {
 var operand1, operand2 int = 0, 0

 if operand1, err := strconv.Atoi(leftOperand); err != nil {
    operand1 = 0
 } else {
    operand1 = operand1
    if operand2, err := strconv.Atoi(rightOperand); err != nil {
        operand2 = 0
    } else {
        operand2 = operand2
    }
 }

    return operand1 + operand2
}
```

```
    sum("1", "2") = 3
    sum("a", "2") = 2
    sum("1", "b") = 1
```

switch statement

```go
switch os := runtime.GOOS; os {
    case "darwin":
        fmt.Println("This is a Macbook")
    case "linux":
        fmt.Println("GNU?")
    case "windows":
        fmt.Println("What???")
    default:
        fmt.Printf("%s\n", os)
}
```

switch with no condition

```
t := time.Now()
switch {
    case t.Hour() < 12:
        fmt.Println("Good morning!")
    case t.Hour() < 17:
        fmt.Println("Good afternoon.")
    default:
        fmt.Println("Good evening")
}
```

switch with fallthrough

```
num := 3
switch {
    case num > 3:
        fmt.Print("3")
        fallthrough
    case num > 2:
        fmt.Print("2")
    case num > 1:
        fmt.Print("1")
    default:
        fmt.Println("-")
}
```

image

Basic syntax - loop

```
for i := 0; i < 10; i++ {

}

for i <= 10 {

}

for {

}
```

Test

```go
func count(i int) int {
    n := 0
    for i := 0; i < i; i++ {
        n += i
    }
    return n
}
```

> count(5) == ?

Demo - Prime factor

print prime number in 1..100

Excercise - Exponentiation (Power)

$$b^x = \underbrace{b \times \cdots \times b}_{x \text{ times}}.$$

```
func power(b, x int) int
```

Packages

Keyword: package

rules

```
only one package in any directory except testing file can plus suffix _test in there

exposed name begins with capital charactor
```

Good Package naming

```
the package name should be good: short, concise, evocative.

By convention, packages are given lower case, single-word names;

there should be no need for underscores or mixedCaps.
```

> https://go.dev/doc/effective_go#package-names

Example package naming

https://pkg.go.dev/std

```
hash/maphash
index/suffixarray
mime/quotedprintable
```

How about your name package?

```
membership
trader
patient
```

or

```
service
model
repository
core
```

image

Exposed name

In ***Go***, a name is exported if it begins with a capital letter. For example, **Pizza** is an exported name, as is **Pi**, which is exported from the math package.

image

import

> import keyword imports the specified package from the directory of module

package directory

```
go.mod                          // module github.com/gophernment/financial
main.go
    |_📂customer
            |_customer.go   // package customer
            |_address.go
```

```go
package main

import "github.com/gophernment/financial/customer"
```

import usage

The importer of a package will use the name to refer to its contents, so exported names in the package can use that fact to avoid repetition. For instance, the buffered reader type in the bufio package is called **Reader**, not **BufReader**, because users see it as **bufio.Reader,** which is a clear, concise name. Moreover, because imported entities are always addressed with their package name, bufio.Reader does not conflict with io.Reader. Similarly, the function to make new instances of ring.Ring—which is the definition of a constructor in Go—would normally be called NewRing, but since Ring is the only type exported by the package, and since the package is called ring, it's called just New, which clients of the package see as ring.New. Use the package structure to help you choose good names.

image

Package naming exercise

Horoscope API ดูดวงครอบจักรวาล

- Zodiac (ดูดวงตามราศีเกิด): **func([birthYear string]) string**
- Numerology (ดูดวงเลขบัตร): **func([idCardNo string]) string**
- Tarot Card (สุ่มไพ่ทำนาย): **func() string**

สร้าง package พร้อม function แล้วเรียกใช้จาก main

Unit testing in go

3 Conditions

1. filename has suffix **_test.go** such as **foobar_test.go**
2. function name prefix is **Test**
3. the test function only get 1 parametter type ***testing.T**

```
import "testing"

func TestACase(t *testing.T) {

}

func Test_a_case(t *testing.T) {

}
```

Unit testing

image

AAA

```
// Arrange
given := 1
want := "1"

// Act
get := foobar(given)

// Assert
if want != get {
    // error report
}
```

Testing absolute add function

```go
func absoluteAdd(a, b string) float64 {}
```

> strconv.ParseFloat("1", 64)
> math.Abs(float64)

1. add("1", "1"): return 2

Testing absolute add function

```go
func absoluteAdd(a, b string) float64 {}
```

> strconv.ParseFloat("1", 64)
> math.Abs(float64)

1. add("1", "1"): return 2
2. add("", ""): return 0

Testing absolute add function

```go
func absoluteAdd(a, b string) float64 {}
```

> strconv.ParseFloat("1", 64)
> math.Abs(float64)

1. add("1", "1"): return 2
2. add("", ""): return 0
3. add("", "1"): return 1

Testing absolute add function

```go
func absoluteAdd(a, b string) float64 {}
```

> strconv.ParseFloat("1", 64)
> math.Abs(float64)

1. add("1", "1"): return 2
2. add("", ""): return 0
3. add("", "1"): return 1
4. add("a", "1"): return -1

Testing absolute add function

```go
func absoluteAdd(a, b string) float64 {}
```

> strconv.ParseFloat("1", 64)
> math.Abs(float64)

1. add("1", "1"): return 2
2. add("", ""): return 0
3. add("", "1"): return 1
4. add("a", "1"): return -1
5. add("1", "a"): return -1

Testing absolute add function

```go
func absoluteAdd(a, b string) float64 {}
```

> strconv.ParseFloat("1", 64)
> math.Abs(float64)

1. add("1", "1"): return 2
2. add("", ""): return 0
3. add("", "1"): return 1
4. add("a", "1"): return -1
5. add("1", "a"): return -1
6. add("1", "-1"): return 2

image

Basic Type

```
bool

string

int  int8  int16  int32  int64

uint uint8 uint16 uint32 uint64 uintptr

byte  // alias for uint8

rune  // alias for int32
      // represents a Unicode code point

float32 float64

complex64 complex128
```

Type Conversion

```go
var i int = 9
var f float64 = 9

if i == int(f) {
    fmt.Println("same")
}
```

Type Conversion

```go
var char byte = 'A'
var ascii uint8 = 65

if char == ascii {
    fmt.Println("same")
}
```

Type Conversion

```go
var char rune = 'ก'
var unicode int32 = 0xe01

if char == unicode {
    fmt.Println("same")
}
```

alias type

```
type char = byte
var b byte = 'a'
var c char = 'a'

fmt.Println(b == c)
```

image

new type

```go
type char byte
var b byte = 'a'
var c char = 'a'

fmt.Println(b == byte(c))
```

image

constants

```
Constants are declared like variables, but with the const keyword.
Constants can be character, string, boolean, or numeric values.
Constants cannot be declared using the := syntax.
```

const

once the value of constant is defined, it cannot be modified further

```
const (
    zero = 0
    one = 1
    two = 2
)
```

iota (ɪ: /aɪˈoʊtə/) identifier

```
const (
    zero = iota
    one
    two
)
```

iota (ɪ: /aɪˈoʊtə/) shift

```go
type ByteSize float64

const (
    _ = iota
    KB ByteSize = 1 << (10 * iota)
    MB
    GB
    TB
    PB
    EB
    ZB
    YB
)
```

play with const & iota

```
type weekday int
```

sunday = 1

monday = 2

.

.

.

image

Keywords: 14/25

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

image

Next > Go 102