Go 101

Pallat Anchaleechamaikorn

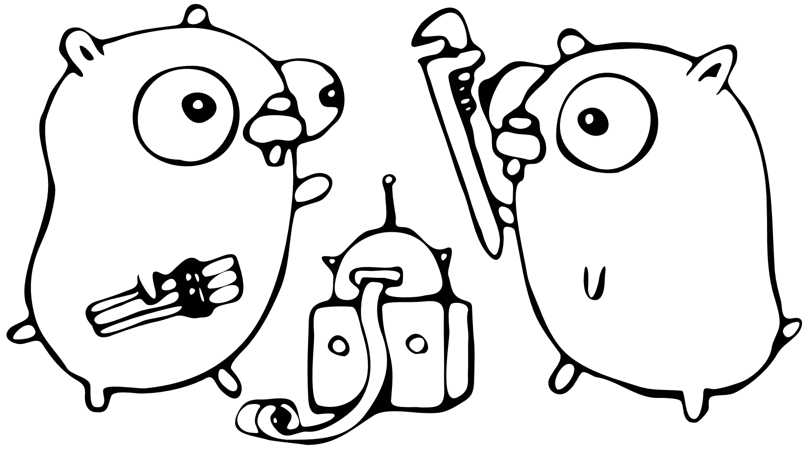
yod.pallat@gmail.com
https://github.com/pallat
https://dev.to/pallat

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

Getting Started

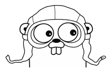
https://go.dev/

Go vs Golang



Wiki

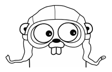
https://en.wikipedia.org/wiki/Go_(programming_language)



About Go

Go Users

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



Installation

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

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

```
go version
```

print Go version

Go Toolchain

```
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
```

package main

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

func main()

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

go run main.go

compile and run Go program

Keywords: 3/10

```
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

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

Zero Value

Declaration with initial value

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

Type inference

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

Type inference without var keyword

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

Only in Functions

_ underscore identifier

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

Pointer variable

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

Pointer

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

The type *T is a pointer to a T value. Its zero value is nil.

```
var p *int
```

The & operator generates a pointer to its operand.

```
i := 42
p = &i
```

The # operator denotes the pointer's underlying value.

```
fmt.Println(*p) // read i through the pointer p
*p = 21         // set i through the pointer p
```

> This is known as "dereferencing" or "indirecting".
> Unlike C, Go has no pointer arithmetic.

pointer zero value

```go
var p *int
```

new

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

Example Pointer

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

fmt.Println(*p, i)

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

Question: pointer

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

*a = 10

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

Question: pointer

```go
var a = new(int)

b := *a

*a = 10

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

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)
```

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)
```

Zero Value with pointer

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.
```

func

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

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

Exercise - Area of a Square Function

```go
func squareArea(a float64) float64 {

}
```

example: squareArea(4) = 16

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

}
```

example: hi("Gopher") = "Hi, Gopher"

Functions with no return

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

Functions multiple return values

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

Named return values

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

Play with func (multiple return values)

```go
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
}
```

exercise: function and pointer

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

exercise: function and pointer(2)

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

{ unexpected newline

for example

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

}
```

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

}
```

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

```go
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

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

Variable Scoping and Variable Shadowing

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

Variable Shadowing (2)

```go
package main

import "fmt"

var i int

func main() {
    // fmt.Printf("this i address is %p and the value is %v\n",&i,i)
    i := 10
    fmt.Println("which i is here:", i)
    // fmt.Printf("this i address is %p and the value is %v\n",&i,i)
}
```

Test

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

> atLeastTen(2) == ?

Test

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

a = ?

b = ?

```go
func sum(s string) (count int,err error) {
    var operand1, operand2 int
    operands := strings.Split(s, "+")
    if operand1, err := strconv.Atoi(operands[0]); err != nil {
        return operand1, err
    }
    operand2, err = strconv.Atoi(operands[0])

    return operand1 + operand2, err
}
```

sum("1+2") = ?

sum("a+2") = ?

sum("1+b") = ?

What's wrong with it?

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

```go
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("-")
}
```

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)

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

```go
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
```

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.

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**

```go
import "testing"

func TestACase(t *testing.T) {

}

func Test_a_case(t *testing.T) {

}
```

Unit testing

AAA

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

// Act
get := foobar(given)

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

FooBar

```
given 1 want "1"
given 2 want "2"
```

FooBar

```
given 1 want "1"
given 2 want "2"
given 3 want "Foo"
```

FooBar

```
given 1 want "1"
given 2 want "2"
given 3 want "Foo"
given 4 want "4"
given 5 want "Bar"
```

FooBar

```
given 1 want "1"
given 2 want "2"
given 3 want "Foo"
given 4 want "4"
given 5 want "Bar"
given 6 want "Foo"
given 7 want "7"
given 8 want "9"
```

FooBar

```
given 1 want "1"
given 2 want "2"
given 3 want "Foo"
given 4 want "4"
given 5 want "Bar"
given 6 want "Foo"
given 7 want "7"
given 8 want "9"
given 9 want "Foo"
```

FooBar

```
given 1 want "1"
given 2 want "2"
given 3 want "Foo"
given 4 want "4"
given 5 want "Bar"
given 6 want "Foo"
given 7 want "7"
given 8 want "9"
given 9 want "Foo"
given 10 want "Bar"
```

FooBar

```
given 1 want "1"
given 2 want "2"
given 3 want "Foo"
given 4 want "4"
given 5 want "Bar"
given 6 want "Foo"
given 7 want "7"
given 8 want "9"
given 9 want "Foo"
given 10 want "Bar"
given 20 want "Bar"
```

FooBar

```
given 1 want "1"
given 2 want "2"
given 3 want "Foo"
given 4 want "4"
given 5 want "Bar"
given 6 want "Foo"
given 7 want "7"
given 8 want "9"
given 9 want "Foo"
given 10 want "Bar"
given 15 want "FooBar"
given 20 want "Bar"
```

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

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

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

new type

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

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

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

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

iota (ɪː /aɪˈoʊtə/) shift

```
type ByteSize float64

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

play with const & iota

```go
type weekday int
```

sunday = 1

monday = 2

.

.

.

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**

Next > Go 102