Go

ODDS | Thaibev

Day 1

Coverage

Go Syntax

Go Types

Go Interface

Go Error

Welcome to the Course

What is Go?

Statically typed & compiled language

Designed by Robert Griesemer, Rob Pike, Ken Thompson at Google

Similar to C, but with memory safety, garbage collection

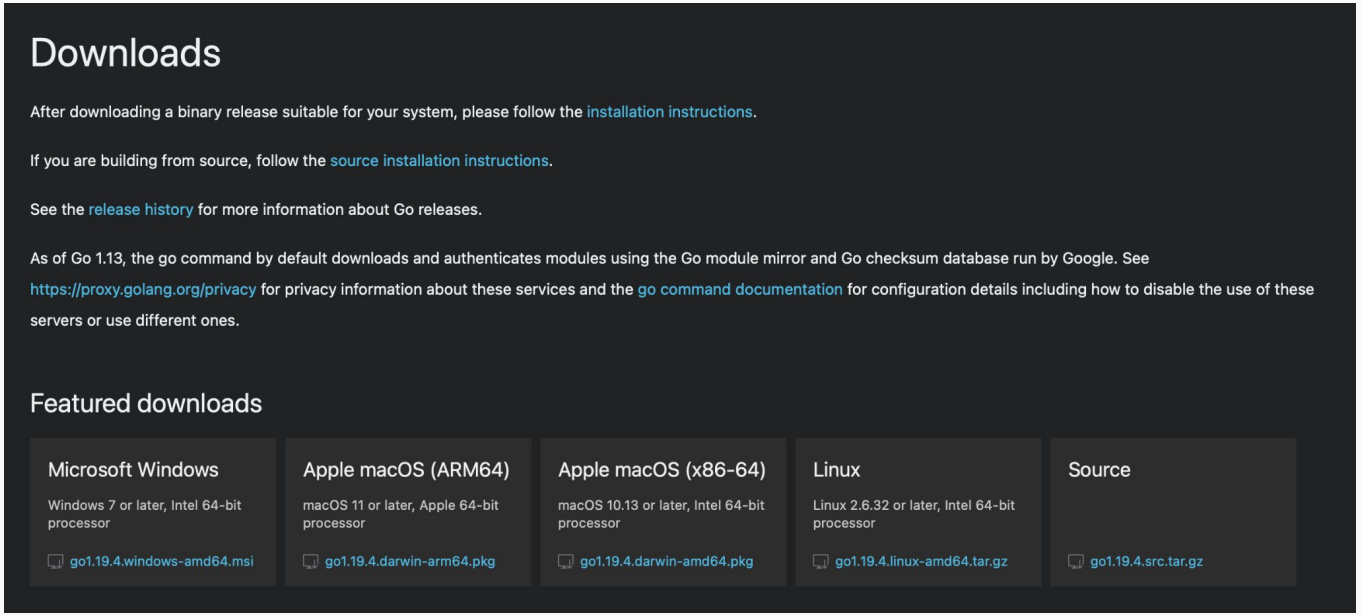
Go compiler is available on Linux, OS X, Window, various BSD & Unix version

Go is open source

Environment Setup

You can download and install Golang based on your distribution here

https://golang.org/dl/



Get Started

Go Hello

```
package main

import "fmt"

func main() {

    fmt.Println("Hello, Thaibev")

}
```

Go Hello

```
package main

import "fmt"

func main() {

    fmt.Println("Hello, Thaibev")

}
```

Package

All Go code lives in packages.

Packages contain type, function, variable, and constant declarations.

Packages can be very small (package errors has just one declaration) or very large (package net/http has >100 declarations).

Case determines visibility: Println is exported, println is not.

More Packages

Some packages are part of the standard library:

- “fmt”: formatting and printing

- “encoding/json”: JSON encoding and decoding

golang.org/pkg for the whole list

Convention: package names match the last element of the import path.

import “fmt” → fmt.Println import “math/rand” → rand.Intn

Variables

```
package main

import "fmt"

var c, python, java bool

func main() {
    var i int
    fmt.Println(i, c, python, java)
}
```

Variables with initializers

```
package main

import "fmt"

var i, j int = 1, 2

func main() {
    var c, python, java = true, false, "no!"
    fmt.Println(i, j, c, python, java)
}
```

Short variable declarations

```
package main

import "fmt"

func main() {
    var i, j int = 1, 2
    k := 3
    c, python, java := true, false, "no!"

    fmt.Println(i, j, k, c, python, java)
}
```

Basic types

```
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
```

Basic types

```
package main
import (
    "fmt"
    "math/cmplx"
)
var (
    ToBe   bool       = false
    MaxInt uint64     = 1<<64 - 1
    z      complex128 = cmplx.Sqrt(-5 + 12i)
)

func main() {
    fmt.Printf("Type: %T Value: %v\n", ToBe, ToBe)
    fmt.Printf("Type: %T Value: %v\n", MaxInt, MaxInt)
    fmt.Printf("Type: %T Value: %v\n", z, z)
}
```

Zero values

```
package main
import "fmt"
func main() {
    var i int
    var f float64
    var b bool
    var s string
    fmt.Printf("%v %v %v %q\n", i, f, b, s)
}
```

Type conversions

```
package main
import (
    "fmt"
    "math"
)
// The expression T(v) converts the value v to the type T.
func main() {
    var x, y int = 3, 4
    var f float64 = math.Sqrt(float64(x*x + y*y))
    var z uint = uint(f)
    fmt.Println(x, y, z)
}
```

Type inference

```
package main

import "fmt"

func main() {
    v := 0.867 + 0.5i // change me!
    fmt.Printf("v is of type %T\n", v)
}
```

Constants

```
package main

import "fmt"

const Pi = 3.14

func main() {
    const World = "世界"
    fmt.Println("Hello", World)
    fmt.Println("Happy", Pi, "Day")

    const Truth = true
    fmt.Println("Go rules?", Truth)
}
```

Functions

```
package main
import "fmt"
func add(x int, y int) int {
    return x + y
}
func main() {
    fmt.Println(add(42, 13))
}
```

Functions

```
package main
import "fmt"
func add(x, y int) int {
    return x + y
}
func main() {
    fmt.Println(add(42, 13))
}
```

Multiple results

```
package main
import "fmt"
func swap(x, y string) (string, string) {
    return y, x
}
func main() {
    a, b := swap("hello", "world")
    fmt.Println(a, b)
}
```

Named return values

```
package main
import "fmt"
func split(sum int) (x, y int) {
    x = sum * 4 / 9
    y = sum - x
    return
}
func main() {
    fmt.Println(split(17))
}
```

For

```
package main
import "fmt"
func main() {
    sum := 0
    for i := 0; i < 10; i++ {
        sum += i
    }
    fmt.Println(sum)
}
```

For (Continue)

```
package main
import "fmt"
func main() {
    sum := 1
    for ; sum < 1000; {
        sum += sum
    }
    fmt.Println(sum)
}
```

For is Go's "while"

```
package main
import "fmt"
func main() {
    sum := 1
    for sum < 1000 {
        sum += sum
    }
    fmt.Println(sum)
}
```

Forever

```
package main
import "fmt"
func main() {
    For {
    }
}
```

If

```
package main
import (
    "fmt"
    "math"
)
func sqrt(x float64) string {
    if x < 0 {
        return sqrt(-x) + "i"
    }
    return fmt.Sprint(math.Sqrt(x))
}

func main() {
    fmt.Println(sqrt(2), sqrt(-4))
}
```

If with a short statement

```
package main
import (
    "fmt"
    "math"
)
func pow(x, n, lim float64) float64 {
    if v := math.Pow(x, n); v < lim {
        return v
    }
    return lim
}
```

```
func main() {
    fmt.Println(
        pow(3, 2, 10),
        pow(3, 3, 20),
    )
}
```

If and else

```
func pow(x, n, lim float64) float64 {
    if v := math.Pow(x, n); v < lim {
        return v
    } else {
        fmt.Printf("%g >= %g\n", v, lim)
    }
    // can't use v here, though
    return lim
}
```

If and else

```
func pow(x, n, lim float64) float64 {
    if v := math.Pow(x, n); v < lim {
        return v
    } else {
        fmt.Printf("%g >= %g\n", v, lim)
    }
    // can't use v here, though
    return lim
}
```

Switch

```
package main
import (
    "fmt"
    "runtime"
)
func main() {
    fmt.Print("Go runs on ")
    switch os := runtime.GOOS; os {
    case "darwin":
        fmt.Println("OS X.")
    case "linux":
        fmt.Println("Linux.")
    default:
        // freebsd, openbsd,
        // plan9, windows...
        fmt.Printf("%s.\n", os)
    }
}
```

Switch evaluation order

```
package main
import (
    "fmt"
    "time"
)
func main() {
    fmt.Println("When's Saturday?")
    today := time.Now().Weekday()
    switch time.Saturday {
    case today + 0:
        fmt.Println("Today.")
    case today + 1:
        fmt.Println("Tomorrow.")
    case today + 2:
        fmt.Println("In two days.")
    default:
        fmt.Println("Too far away.")
    }
}
```

Switch with no condition

```
package main

import (
    "fmt"
    "time"
)

func main() {
    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.")
    }
}
```

Defer

```
package main
import "fmt"
func main() {
    fmt.Println("counting")

    for i := 0; i < 10; i++ {
        defer fmt.Println(i)
    }

    fmt.Println("done")
}
```

Exercise: Loops and Functions

Problem:

Write a function that takes an integer n as an argument and returns the sum of all even numbers from 1 to n.

Pointers

```
package main
import "fmt"
func main() {
    i, j := 42, 2701

    p := &i          // point to i
    fmt.Println(*p) // read i through the pointer
    *p = 21          // set i through the pointer
    fmt.Println(i)  // see the new value of i

    p = &j          // point to j
    *p = *p / 37    // divide j through the pointer
    fmt.Println(j) // see the new value of j
}
```

Structs

```
package main
import "fmt"
type Vertex struct {
    X int
    Y int
}

func main() {
    fmt.Println(Vertex{1, 2})
}
```

Struct Literals

```
package main
import "fmt"

type Vertex struct {
    X, Y int
}

var (
    v1 = Vertex{1, 2}  // has type Vertex
    v2 = Vertex{X: 1}  // Y:0 is implicit
    v3 = Vertex{}      // X:0 and Y:0
    p  = &Vertex{1, 2} // has type *Vertex
)

func main() {
    fmt.Println(v1, p, v2, v3)
}
```

Array

```
package main
import "fmt"

func main() {
    var a [2]string
    a[0] = "Hello"
    a[1] = "World"
    fmt.Println(a[0], a[1])
    fmt.Println(a)

    primes := [6]int{2, 3, 5, 7, 11, 13}
    fmt.Println(primes)
}
```

Slices

```
package main
import "fmt"

func main() {
    primes := [6]int{2, 3, 5, 7, 11, 13}

    var s []int = primes[1:4]
    fmt.Println(s)
}
```

Slices are like references to arrays

```
package main

import "fmt"

func main() {
    names := [4]string{
        "John",
        "Paul",
        "George",
        "Ringo",
    }
    fmt.Println(names)

    a := names[0:2]
    b := names[1:3]
    fmt.Println(a, b)

    b[0] = "XXX"
    fmt.Println(a, b)
    fmt.Println(names)
}
```

Slice literals

```
package main

import "fmt"

func main() {
    q := []int{2, 3, 5, 7, 11, 13}
    fmt.Println(q)

    r := []bool{true, false, true,
true, false, true}
    fmt.Println(r)
```

```
s := []struct {
        i int
        b bool
    }{
        {2, true},
        {3, false},
        {5, true},
        {7, true},
        {11, false},
        {13, true},
    }
    fmt.Println(s)
}
```

Slice length and capacity

```
package main
import "fmt"

func main() {
    s := []int{2, 3, 5, 7, 11, 13}
    printSlice(s)

    // Slice the slice to give it zero length.
    s = s[:0]
    printSlice(s)

    // Extend its length.
    s = s[:4]
    printSlice(s)

    // Drop its first two values.
    s = s[2:]
    printSlice(s)
}

func printSlice(s []int) {
    fmt.Printf("len=%d cap=%d %v\n",
len(s), cap(s), s)
}
```

Creating a slice with make

```
package main
import "fmt"

func main() {
    a := make([]int, 5)
    printSlice("a", a)

    b := make([]int, 0, 5)
    printSlice("b", b)

    c := b[:2]
    printSlice("c", c)

    d := c[2:5]
    printSlice("d", d)
}

func printSlice(s string, x []int) {
    fmt.Printf("%s len=%d cap=%d
%v\n",
        s, len(x), cap(x), x)
}
```

Slices of slices

```
package main
import (
    "fmt"
    "strings"
)

func main() {
    // Create a tic-tac-toe board.
    board := [][]string{
        []string{"_", "_", "_"},
        []string{"_", "_", "_"},
        []string{"_", "_", "_"},
    }

    // The players take turns.
    board[0][0] = "X"
    board[2][2] = "O"
    board[1][2] = "X"
    board[1][0] = "O"
    board[0][2] = "X"

    for i := 0; i < len(board); i++ {
        fmt.Printf("%s\n",
strings.Join(board[i], " "))
    }
}
```

Appending to a slice

```
package main
import "fmt"

func main() {
    var s []int
    printSlice(s)

    // append works on nil slices.
    s = append(s, 0)
    printSlice(s)

    // The slice grows as needed.
    s = append(s, 1)
    printSlice(s)

    // We can add more than one element at
a time.
    s = append(s, 2, 3, 4)
    printSlice(s)
}

func printSlice(s []int) {
    fmt.Printf("len=%d cap=%d %v\n",
len(s), cap(s), s)
}
```

Range

```
package main

import "fmt"

var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}

func main() {
    for i, v := range pow {
        fmt.Printf("2**%d = %d\n", i, v)
    }
}
```

Exercise: Slices

Problem:

Write a function that takes a slice of integers and returns the largest difference between any two elements in the slice. If the slice has fewer than two elements, the function should return 0.

Map

```
package main
import "fmt"

type Vertex struct {
    Lat, Long float64
}

var m map[string]Vertex

func main() {
    m = make(map[string]Vertex)
    m["Bell Labs"] = Vertex{
        40.68433, -74.39967,
    }
    fmt.Println(m["Bell Labs"])
}
```

Map literals

```
package main

import "fmt"

type Vertex struct {
    Lat, Long float64
}

var m = map[string]Vertex{
    "Bell Labs": Vertex{
        40.68433, -74.39967,
    },
    "Google": Vertex{
        37.42202, -122.08408,
    },
}

func main() {
    fmt.Println(m)
}
```

Mutating Maps

```
package main

import "fmt"

func main() {
    m := make(map[string]int)

    m["Answer"] = 42
    fmt.Println("The value:", m["Answer"])

    m["Answer"] = 48
    fmt.Println("The value:", m["Answer"])

    delete(m, "Answer")
    fmt.Println("The value:", m["Answer"])

    v, ok := m["Answer"]
    fmt.Println("The value:", v, "Present?", ok)
}
```

Exercise: Map

Implement WordCount. It should return a map of the counts of each “word” in the string s.

You might find ***strings.Fields*** helpful.

Methods

```
package main
import (
    "fmt"
    "math"
)

type Vertex struct {
    X, Y float64
}
```

```
func (v Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}

func main() {
    v := Vertex{3, 4}
    fmt.Println(v.Abs())
}
```

Pointer receivers

```
package main

import (
    "fmt"
    "math"
)

type Vertex struct {
    X, Y float64
}

func (v Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}

func (v *Vertex) Scale(f float64) {
    v.X = v.X * f
    v.Y = v.Y * f
}

func main() {
    v := Vertex{3, 4}
    v.Scale(10)
    fmt.Println(v.Abs())
}
```

Pointers and functions

```
package main

import (
    "fmt"
    "math"
)

type Vertex struct {
    X, Y float64
}

func Abs(v Vertex) float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

```
func Scale(v *Vertex, f float64) {
    v.X = v.X * f
    v.Y = v.Y * f
}

func main() {
    v := Vertex{3, 4}
    Scale(&v, 10)
    fmt.Println(Abs(v))
}
```

Methods and pointer indirection

```
package main
import "fmt"

type Vertex struct {
    X, Y float64
}

func (v *Vertex) Scale(f float64) {
    v.X = v.X * f
    v.Y = v.Y * f
}
func ScaleFunc(v *Vertex, f float64) {
    v.X = v.X * f
    v.Y = v.Y * f
}
```

```
func main() {
    v := Vertex{3, 4}
    v.Scale(2)
    ScaleFunc(&v, 10)

    p := &Vertex{4, 3}
    p.Scale(3)
    ScaleFunc(p, 8)

    fmt.Println(v, p)
}
```

Interface

```go
package main
import (
    "fmt"
    "math"
)
type Abser interface {
    Abs() float64
}
func main() {
    var a Abser
    f := MyFloat(-math.Sqrt(2))
    v := Vertex{3, 4}
    a = f  // a MyFloat implements Abser
    a = &v // a *Vertex implements Abser
    a = v  // Error
    fmt.Println(a.Abs())
}
```

```go
type MyFloat float64

func (f MyFloat) Abs() float64 {
    if f < 0 {
        return float64(-f)
    }
    return float64(f)
}

type Vertex struct {
    X, Y float64
}

func (v *Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

Interfaces are implemented implicitly

```
package main

import "fmt"

type I interface {
    M()
}

type T struct {
    S string
}

// This method means type T implements the
interface I,
// but we don't need to explicitly declare
that it does so.
func (t T) M() {
    fmt.Println(t.S)
}

func main() {
    var i I = T{"hello"}
    i.M()
}
```

Interface values with nil underlying values

```
package main
import "fmt"

type I interface {
    M()
}
type T struct {
    S string
}
func (t *T) M() {
    if t == nil {
        fmt.Println("<nil>")
        return
    }
    fmt.Println(t.S)
}
```

```
func main() {
    var i I
    var t *T
    i = t
    describe(i)
    i.M()

    i = &T{"hello"}
    describe(i)
    i.M()
}
func describe(i I) {
    fmt.Printf("(%v, %T)\n", i, i)
}
```

The empty interface

```
package main

import "fmt"

func main() {
    var i interface{}
    describe(i)

    i = 42
    describe(i)

    i = "hello"
    describe(i)
}
```

```
func describe(i interface{}) {
    fmt.Printf("(%v, %T)\n", i, i)
}
```

Type assertions

```
package main
import "fmt"

func main() {
    var i interface{} = "hello"
    s := i.(string)
    fmt.Println(s)
    s, ok := i.(string)
    fmt.Println(s, ok)
    f, ok := i.(float64)
    fmt.Println(f, ok)
    f = i.(float64) // panic
    fmt.Println(f)
}
```

Stringers

```
package main
import "fmt"

type Person struct {
    Name string
    Age  int
}
func (p Person) String() string {
    return fmt.Sprintf("%v (%v years)", p.Name, p.Age)
}
func main() {
    a := Person{"Arthur Dent", 42}
    z := Person{"Zaphod Beeblebrox", 9001}
    fmt.Println(a, z)
}
```

Exercise: Stringer

Make the IPAddr type implement fmt.Stringer to print the address as a dotted quad.

For instance, ***IPAddr{1, 2, 3, 4}*** should print as "1.2.3.4".

Error

```
package main
import (
    "fmt"
    "time"
)

type MyError struct {
    When time.Time
    What string
}
func (e *MyError) Error() string {
    return fmt.Sprintf("at %v, %s",
        e.When, e.What)
}
```

```
func run() error {
    return &MyError{
        time.Now(),
        "it didn't work",
    }
}

func main() {
    if err := run(); err != nil {
        fmt.Println(err)
    }
}
```

Exercise: Error

Implement Mod should return a divide by zero error value when given a zero.