Go **from** Beginner

for (assumed) Beginners

made by Made

Gather your will

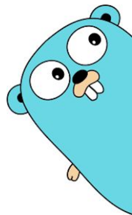
Get your laptop

Visit tour.golang.org

Fin! Thanks :)

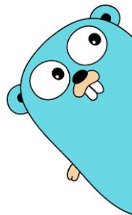
Fin! Thanks :)

Disclaimer: As I would talk more about the go language and skip explaining basic computer science, some terms may only be familiar with those with programming background



Interesting Basics

- Need to have Package (put it at the beginning of the file)
  - (usually) Same folder same package



Interesting Basics

- Need to have Package (put it at the beginning of the file)
    - (usually) Same folder same package
- Export / unexport interface ≈ public / private function,variables

Interesting Basics

- Need to have Package (put it at the beginning of the file)
  - (usually) Same folder same package
- Export / unexport interface ≈ public / private function,variables
- (be fancy) named return value

Interesting Basics

- Need to have Package (put it at the beginning of the file)
  - (usually) Same folder same package
- Export / unexport interface ≈ public / private function,variables
- (be fancy) named return value
- defer
  - wait for surrounding steps finished before executing in LIFO

Interesting Basics

- Need to have Package (put it at the beginning of the file)
  - (usually) Same folder same package
- Export / unexport interface ≈ public / private function,variables
- (be fancy) named return value
- defer
  - wait for surrounding steps finished before executing in LIFO
- Pointers * & ≈ object reference (java) (demo)
  - Memory address of a value
  - Set pointer to a variable -- p := &(variable)
  - Set value through pointers -- *p = value

Interesting Basics

- Need to have Package (put it at the beginning of the file)
  - (usually) Same folder same package
- Export / unexport interface ≈ public / private function,variables
- (be fancy) named return value
- defer
  - wait for surrounding steps finished before executing in LIFO
- Pointers * & ≈ object reference (java)
  - Memory address of a value
  - Set pointer to a variable -- p := &(variable)
  - Set value through pointers -- *p = value
- Struct ≈ ***strict*** descriptors

Interesting Basics

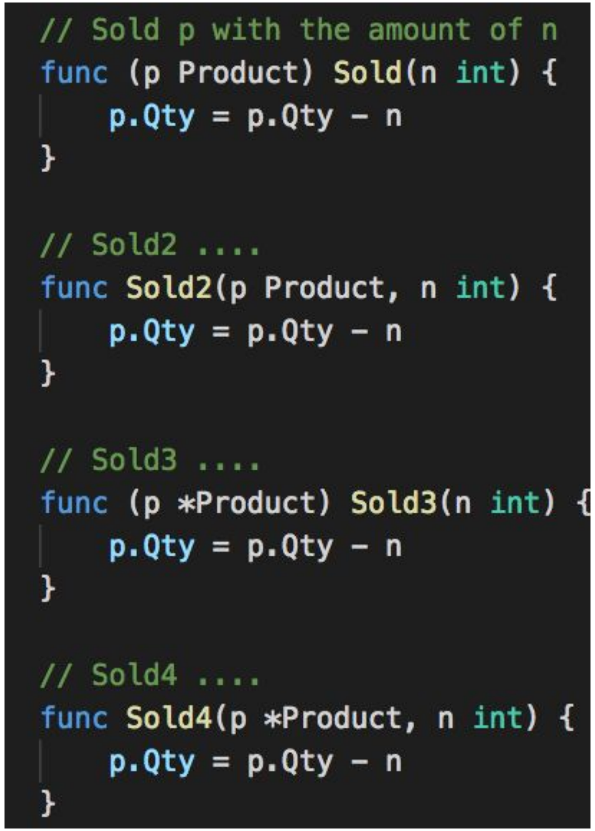
- Need to have Package (put it at the beginning of the file)
  - (usually) Same folder same package
- Export / unexport interface ≈ public / private function,variables
- (be fancy) named return value
- defer
  - wait for surrounding steps finished before executing in LIFO
- Pointers * & ≈ object reference (java)
  - Memory address of a value
  - Set pointer to a variable -- p := &(variable)
  - Set value through pointers -- *p = value
- Struct ≈ ***strict*** descriptors
- Array `[3]int{1, 2, 3}` / Slice `[]int{1, 2, 3}`

More Basics (with Demo)

- Struct - Pointer - null value for JSON (quiz)

More Basics (with Demo)

- Struct - Pointer - null value for JSON
- Methods ≈ function with *receiver* (quiz)
    - because Go doesn't have class :(
    - Value or Pointer?



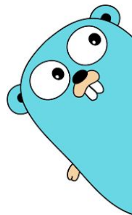
```go
// Sold p with the amount of n
func (p Product) Sold(n int) {
    p.Qty = p.Qty - n
}

// Sold2 ....
func Sold2(p Product, n int) {
    p.Qty = p.Qty - n
}

// Sold3 ....
func (p *Product) Sold3(n int) {
    p.Qty = p.Qty - n
}

// Sold4 ....
func Sold4(p *Product, n int) {
    p.Qty = p.Qty - n
}
```

More Basics (with Demo)

- Struct - Pointer - null value for JSON
- Methods ≈ function with *receiver*
  - because Go doesn’t have class :(
  - Value or Pointer?
- Interface (demo)
  - ≈ like common Interface without ‘implements’ keywords
  - simpler(?) implementation

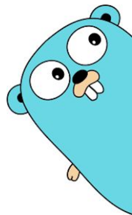
More Basics (with Demo)

- Struct - Pointer - null value for JSON
- Methods ≈ function with *receiver*
  - because Go doesn’t have class :(
  - Value or Pointer?
- Interface
  - ≈ like common Interface without ‘implements’ keywords
  - simpler(?) implementation
- Assert ≈ cast `interface{}` to a data type / struct (demo)
  - Because can not do *interface.value*



Not so Basic

- Goroutines (demo) (quiz)
  - Light weight thread for concurrency
  - return immediately, don’t await until the function is finished
- Channels
  - Pipe for goroutine communication
  - Send & Receive block next process
  - Useful if the next process need return value for thread functions before continue



A little More Knowledge

- Error Handling
  - Always check for error
- Test
  - File *_test.go
  - import “testing”
  - `go test`
- Go Mod: https://blog.golang.org/using-go-modules
  - Collection of packages with go.mod as root
  - Define module path and its dependency requirement (automatically)

Nice Tips

- Install plug-in on your editor to auto-format your go files

+

- Type strict
- Concurrency/Threading
- Many best practices & communities
- Go’s standard lib is enough
- Build in unit-test
- Standardized culture

-

- Type strict
- Parsing JSON becomes difficult, especially if the JSON is too dynamic

Useful References

- General Cheat Sheets: https://github.com/a8m/golang-cheat-sheet
- Data Types Cheat Sheets: https://github.com/emirpasic/gods
- Goroutine & Channel: https://golangbot.com/goroutines/ | https://golangbot.com/channels/
- Packages lib: https://github.com/avelino/awesome-go
- Medium
- StackOverflow
- Google

Fin! Thanks :)

It’s real, Qs?

Footnotes

Named return value

```
func split(a, b int) (x, y int) {
    x = a + b
    y = a - b
    return
}
```

Back to Basics

Defer example

```
f := createFile("/tmp/defer.txt")
    defer closeFile(f)
    writeFile(f)
```

Back to Basics