Learn the Go Programming Language

For experienced developers or those of an adventurous nature

gotutorial.net
@GoTutorialNet

Matt Nunogawa
@amattn

Lesson 07

Idiomatic Go

`v0.1 draft`

FOUNDATION

Idiomatic Go: The Basics

- go fmt
  - Should be tied into your editor/IDE
- https://golang.org/doc/effective_go.html
  - The canonical "Idiomatic Go" document
- Don't ignore errors!

```
decodedBytes, _ := hex.DecodeString(hexOutput)
```

TIPS

- use the ok when doing type assertions:

```
v := unknownInterface.(TYPE) // will panic at runtime
v, _ := unknownInterface.(TYPE) // hard to debug, code smell
v, ok := unknownInterface.(TYPE)  // 👍
```

- Use make to create slices, maps, and channels, new to create pointers
  - https://golang.org/doc/effective_go.html#allocation_new

```
string_map := make(map[string]string)
slice_of_ints := make([]int, 10, 0)
stopChan := make(chan bool)

intPtr := new(int)
catPtr := NewCat()
```

Iterating

```
// Like a C for loop
for <init>; <condition>; <post> { }

// Like a C while loop
for <condition> { }

// Like a C for(;;)
for { }
```

Grouping

```
// not as good

const configPathName     = "config"
const configPathDefault  = “./config.json"
const configPathUsage    = "path to config file"
const versionPathName    = "version"
const versionPathUsage   = “print version and exit"



var configPath  string
var versionFlag bool
var active      bool
var allEntities []Entity
```

GROUPING

```
// better
const (
    configPathName     = "config"
    configPathDefault  = “./config.json"
    configPathUsage    = "path to config file"
    versionPathName    = "version"
    versionPathUsage   = “print version and exit"
)

var (
    configPath  string
    versionFlag bool
    active      bool
    allEntities []Entity
)
```

Variadic Functions

```
// Not this way:
func addPayload(p Payload)
func addPayloads(ps []Payload)

// Do this instead
func addPayloads(ps ...Payload)
```

Variadic Functions

```
// Not this way:
func addPayload(p Payload)
func addPayloads(ps []Payload)

addPayload(Payload{})
addPayloads(sliceOfPayloads)
addPayloads([]Payload{
    Payload{},
    Payload{},
})
```

Variadic Functions

```
// Do this instead
func addPayloads(ps ...Payload)

addPayloads(Payload{})
addPayloads(sliceOfPayloads...)
addPayloads(Payload{},Payload{})
```

Debugging

- Debugging:
  - prefer log over fmt for logging to standard out
  - use the expvar package
  - wrap your errors

Constructor Debugging Trick

```
type Thing struct {
  didUserConstructor bool
}

// when people properly use the constructor, the
unexported canary boolean will be false
func NewThing() {
  ptr := new(Thing)
  ptr.didUserConstructor = true
  return ptr
}
```

Organization

```
/package/main.go
/package/main_test.go
/package/version.go
```

MAIN.GO

```
package main
import (
    "log"
    "runtime"
    "github.com/amattn/deeperror"
)
func main() {
    log.Println()
    log.Println("Go Version:", runtime.Version())
    log.Println("<NAME>", Version(), “build", BuildNumber())
    log.Println("↳ deeperror  ", deeperror.Version(),
        "build", deeperror.BuildNumber()
    )
    // the truly paranoid can also log os.Environ
    // and some other stuff in runtime, like NumCPU()

    // do stuff
}
```

MAIN_TEST.GO

```
package main

import “testing”

func TestForceFail(t *testing.T) {
    t.Fatal(“just checking test harness”)
}
```

VERSION.GO

```
package main

const (
    internal_BUILD_NUMBER = 9
    internal_VERSION_STRING = "0.2.0"
)

func BuildNumber() int64 {
    return internal_BUILD_NUMBER
}
func Version() string {
    return internal_VERSION_STRING
}
```

Error Handling

```
func doSomething(name string) error {
    err := stepOne(name)
    if err != nil {
        return err
    }
    thing, err := stepTwo(name)
    if err != nil {
        return err
    }
    number, err := stepThree(thing)
    if err != nil {
        return err
    }
    return stepFour(number)
}
```

Error Handling

two clause if statement

```
func doSomething(name stri

    if err := stepOne(name); err != nil {
        return err
    }
    thing, err := stepTwo(name)
    if err != nil {
        return err
    }
    number, err := stepThree(thing)
    if err != nil {
        return err
    }
    return stepFour(number)
}
```

Error Handling Helpers

```
func IsErr(err error, msg ...interface{}) bool {
    if err == nil { return false }
    log.Println(msg...)
    return true
}

func doSomething(name string) error {
    err := stepOne(name)
    if IsErr(err, "stepOne failure", name) {return err}
    num, err := stepTwo(name)
    if IsErr(err, "stepTwo failure", num) {return err}
    log.Println(num)
    return nil
}
```

ERROR WRAPPING

- Allows you to track the “error chain” as error get passed up

```
func doSomething(s string) error {
    i, err := ParseInt(s, 10, 64)
    if err != nil {
    return fmt.Errorf(“3596300981 error parsing int,
%s, %v”, s, err)
    }
    // …
}
```

PANIC

- Should *only* be used in truly exceptional situations where things have gone terribly wrong
- If you are coming from Java/python/etc., panics are much, much more rare than exceptions in those environments
- Ok: during setup/config, where runtime errors can be immediately surfaced. Alternatively, use log.Fatalf()
- https://golang.org/doc/effective_go.html#panic

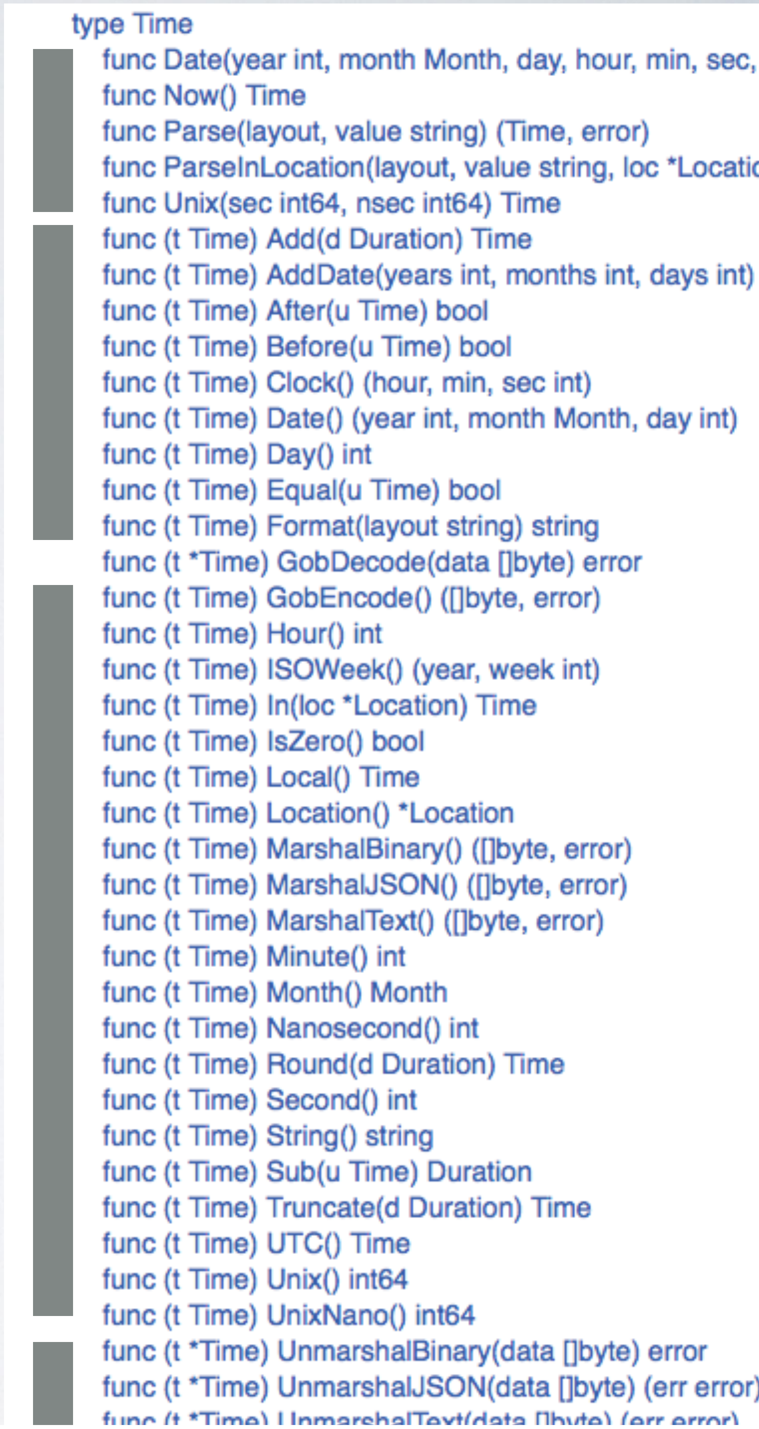
Standard Library

TIME

funcs to create things

methods that use or return a copy of a thing

methods that modify a thing or operate on ptrs to a thing



```
type Time
    func Date(year int, month Month, day, hour, min, sec,
    func Now() Time
    func Parse(layout, value string) (Time, error)
    func ParseInLocation(layout, value string, loc *Locatio
    func Unix(sec int64, nsec int64) Time
    func (t Time) Add(d Duration) Time
    func (t Time) AddDate(years int, months int, days int)
    func (t Time) After(u Time) bool
    func (t Time) Before(u Time) bool
    func (t Time) Clock() (hour, min, sec int)
    func (t Time) Date() (year int, month Month, day int)
    func (t Time) Day() int
    func (t Time) Equal(u Time) bool
    func (t Time) Format(layout string) string
    func (t *Time) GobDecode(data []byte) error
    func (t Time) GobEncode() ([]byte, error)
    func (t Time) Hour() int
    func (t Time) ISOWeek() (year, week int)
    func (t Time) In(loc *Location) Time
    func (t Time) IsZero() bool
    func (t Time) Local() Time
    func (t Time) Location() *Location
    func (t Time) MarshalBinary() ([]byte, error)
    func (t Time) MarshalJSON() ([]byte, error)
    func (t Time) MarshalText() ([]byte, error)
    func (t Time) Minute() int
    func (t Time) Month() Month
    func (t Time) Nanosecond() int
    func (t Time) Round(d Duration) Time
    func (t Time) Second() int
    func (t Time) String() string
    func (t Time) Sub(u Time) Duration
    func (t Time) Truncate(d Duration) Time
    func (t Time) UTC() Time
    func (t Time) Unix() int64
    func (t Time) UnixNano() int64
    func (t *Time) UnmarshalBinary(data []byte) error
    func (t *Time) UnmarshalJSON(data []byte) (err error)
    func (t *Time) UnmarshalText(data []byte) (err error)
```

TIME

```
// http://play.golang.org/p/E_J0TgM0Tu

go_birthday := time.Unix(1257894000, 0)
ts := go_birthday.Unix()

fmt.Println("pretty print:  ”, go_birthday)
fmt.Println("unix timestamp:”, ts)

// pretty print:   2009-11-10 23:00:00 +0000 UTC
// unix timestamp: 1257894000
```

ENCODING/HEX ([]BYTE/STRING)

```
// http://play.golang.org/p/mRBUCGdw-l

stringInput := "Hello, playground!!!!!"
hexOutput := hex.EncodeToString([]byte(stringInput))
fmt.Println(hexOutput)

decodedBytes, _ := hex.DecodeString(hexOutput)
fmt.Println(string(decodedBytes))

// outputs:
// 48656c6c6f2c20706c617967726f756e642121212121
// Hello, playground!!!!!
```

ENCODING/BASE64 (IO.READER, IO.WRITER)

```
// http://play.golang.org/p/6kcJfBSDs4
// func NewEncoder(enc *Encoding, w io.Writer) io.WriteCloser

input := []byte("foo\x01bar !<>?,./:")
encoder := base64.NewEncoder(base64.URLEncoding, os.Stdout)
encoder.Write(input)
encoder.Close()
fmt.Println()

buf := new(bytes.Buffer)
encoder = base64.NewEncoder(base64.URLEncoding, buf)
encoder.Write(input)
encoder.Close()
readBytes, _ := ioutil.ReadAll(buf)
fmt.Println(string(readBytes))
```

Idiomatic Buffers

```go
// from the io package:

type Reader interface {
    Read(p []byte) (n int, err error)
}
type Writer interface {
    Write(p []byte) (n int, err error)
}
type ReadWriter interface {
    Reader
    Writer
}
type Closer interface {
    Close() error
}
type WriteCloser interface {
    Writer
    Closer
}
```

Idiomatic Buffers

- Typically:
  - for a write buffer, use new(bytes.Buffer)
  - for a read buffer, use one of:
    - bytes.NewBuffer(buf []byte)
    - bytes.NewBufferString(s string)

Idiomatic Readers & Writers

- Don’t need to use buffers
- io.Readers and io.Writers allow you to pipe data through various parts & components

ENCODING/BASE64 (IO.READER, IO.WRITER)

```
// http://play.golang.org/p/6kcJfBSDs4
// func NewEncoder(enc *Encoding, w io.Writer) io.WriteCloser

input := []byte("foo\x01bar !<>?,./:")
encoder := base64.NewEncoder(base64.URLEncoding, os.Stdout)
encoder.Write(input)
encoder.Close()
fmt.Println()

buf := new(bytes.Buffer)
encoder = base64.NewEncoder(base64.URLEncoding, buf)
encoder.Write(input)
encoder.Close()
readBytes, _ := ioutil.ReadAll(buf)
fmt.Println(string(readBytes))
```

CRYPTO/SHA256

```
// http://play.golang.org/p/xrc0xgXZRW

input := []byte("abcdefghijklmnopqrztuvwxyz012345")

// using writer
hasher := sha256.New()
hasher.Write(input)
sum := hasher.Sum([]byte{})
fmt.Println(hex.EncodeToString(sum))

// convenience way
output := sha256.Sum256(input)
fmt.Println(hex.EncodeToString(output[:]))

// prints:
// 1501ba891fda3a810331ca0beacba24f4eaa211480c02a82cb20cd8e9c9a67a7
// 1501ba891fda3a810331ca0beacba24f4eaa211480c02a82cb20cd8e9c9a67a7
```

returns an array, so we convert the array to a slice

Random Numbers

- The math/rand package has good documentation
- Remember to seed!

```
rand.New(rand.NewSource(time.Now().UnixNano()))
```

- There is no UInt64()
  - https://groups.google.com/forum/#!topic/golang-nuts/Kle874lT1Eo
- If you need cryptographically secure random numbers, use crypto/rand

SYNC

- Mutex, Cond, others

- Once

```
var once sync.Once
once.Do(func() {fmt.Println("Only once”)})
```

- WaitGroup

```
var wg sync.WaitGroup
wg.Add(1)
wg.Add(1)
wg.Wait()
```

waiting (blocking) for other goroutines to call wg.Done() twice.

SYNC/ATOMIC

```
LoadUint64(addr *uint64) (val uint64)
StoreUInt64(addr *int64, val int64)

AddUint64(addr *uint64, delta uint64) (new uint64)
SwapUint64(addr *uint64, new uint64) (old uint64)
CompareAndSwapUint64(addr *uint64, old, new uint64)
(swapped bool)
```

OTHER STDLIB

```
encoding, crypto // other standards
archive/tar & archive/zip
compress/gzip, lzw, bzip2, & more
math/big
strconv
io, io/ioutil, path, path/filepath // file I/O
net, net/http & net/url
text/template & html/template
```

Other packages

- Other repos, usually maintained by core Go team. Contains things that are too new, or move too fast or just don't belong in stb lib.

```
code.google.com/p/go.crypto
code.google.com/p/go.exp  //experimental and deprecated
stuff)
code.google.com/p/go.image
code.google.com/p/go.net
code.google.com/p/go.text
code.google.com/p/go.tools  // compiler, linter,
playground, godoc, blog, etc. )
```

Making Command Line Tools

The Basics of Making Command Line Tools

- making command line tools
  - flag, args
  - config
  - version
  - os.signal

Flag package: Args

- Use the flags package

```
// http://play.golang.org/p/whm8qWNOwN

import "flag"
var (
    countFlag int
    prefixFlag string
)
```

name

default

help text

```
func init() {
    flag.IntVar(&countFlag, ”count", 1234, “number of iterations”)
    flag.StringVar(&prefixFlag, ”prefix", “”, “prefix to append to output")
}

func main() {
    flag.Parse()

    for i:=0; i < count; i++ {
        output := prefixFlag + processOutput()
    }
    // …
}
```

FLAG PACKAGE: ARGS

- Use the flags package

```
// http://play.golang.org/p/knGnJNHh8q

import "flag"
var (
    countFlag = flag.Int(”count", 1234, “number of iterations”)
    prefixFlag = flag.String(”prefix", “”, “prefix to append to output")
)

func main() {
    flag.Parse()

    for i:=0; i < count; i++ {
        output := prefixFlag + processOutput()
    }
    // …
}
```

pointers!

CONFIG: FLAG TO A PATH

```
var confPath string

func init() {
    flag.String("config", “./config.json”, “path to config file")
}

func main() {
    flag.Parse()

    // read config file
    bytes, err := ioutil.ReadFile(configPath)
    if err != nil {
        log.Fatalln(
            "error reading configfile:",
            configPath, "err:", err
        )
    }
    // parse config file

    // …
}
```

- - VERSION

```
versionFlag = bool
func init() {
    flag.BoolVar(&versionFlag, “version”, false, “print version
and exit”)
}
func main() {
    log.Println()
    log.Println("Go Version:", runtime.Version())
    log.Println(“cli_tool", Version(),
        “build", BuildNumber()
    )
    flag.Parse()
    if versionFlag {
        os.Exit(0)
    }
    // do stuff
}
```

SIGNALS

```
func main() {
    // …  setup and other …

    // Use a buffered channel or risk missing the signal
    // if we're not ready to receive when the signal is sent.
    sigChan := make(chan os.Signal, 1)
    signal.Notify(sigChan, os.Interrupt, os.Kill)

    serviceStopChan := make(chan bool)
    go StartService(serviceStopChan)

       for {
            select {
                case <-sigChan:
                    StopService()
                case <- serviceStopChan:
                    return
           }
        }
}
```

syscall package has os-specific signals

More Info

- std lib
  - http://golang.org/pkg/flag/
- other
  - https://github.com/codegangsta/cli

Testing

TESTING

- http://golang.org/pkg/testing/
- Basic template:

```
package main

import "testing"

func TestXxx(t *testing.T) {
    …
    if x == bad {
        t.Error(“expected x is good, but got”, x)
    }
    if y == 0 {
        t.Error(“expected non-zero y, got”, y)
    }
}
func TestYyy(t *testing.T) {
    t.Fatal(“This will always fail")
}
```

TABLE DRIVEN TESTING: SLICES

- https://code.google.com/p/go-wiki/wiki/TableDrivenTests
- http://dave.cheney.net/2013/06/09/writing-table-driven-tests-in-go

```
func TestXxx(t *testing.T) {
    inputs := []string{“one”, “two”, “twenty-two”,…}
    expecteds := []int{1, 2, 22,…}
    if len(inputs) != len(expected) {
        t.Fatal(“C’MON!”)
    }
    for i, input := range inputs {
        candidate := word2Int(input)
        if candidate != expecteds[i] {
            t.Errorf(
                “table_idx:%d candidate != exp, %v != %v”,
                i, candidate, expecteds[i],
            )
        }
    }
}
```

Table Driven Testing: Structs

```
type InsExp struct {
    input string
    expected int
}
func TestXxx(t *testing.T) {
    table := []InsExp{
        InsExp{“one”, 1},
        InsExp{“two”, 2},
        InsExp{“twentytwo”, 22},
        InsExp{“twenty two”, 22},
        InsExp{“twenty-two”, 22},
    }
    for i, inexp := range table {
        candidate := word2Int(inexp.input)
        if candidate != inexp.expected {
            t.Errorf(“table_idx:%d candidate != exp, %v != %v”, i,
candidate, inexp. expected)
        }
    }
}
```

Table Driven Testing: Structs

- Can get more sophisticated:

```
type EndpointTestCase struct {
    inputURL string
    expectedCode int
    expectedBody []bytes
    expectedHeaders http.Header
}
```

- Can use external files to populate tables
    - external files can be generated, maintained by other teams, etc.

HTTP Testing

- net/http/httptest
- ResponseRecorder is a http.ResponseWriter implementation that is useful for recoding how a handler handles a request. Allows relatively easy unit testing of handlers.
- Server is a full on http server designed to do end to end tests in the testing infrastructure.

HTTP Testing: httptest.Server

create test server

```
// https://golang.org/pkg/net/http/httptest/#example_Server

ts := httptest.NewServer(http.HandlerFunc(func(w
http.ResponseWriter, r *http.Request) {
     fmt.Fprintln(w, "Hello, client")
     }))
defer ts.Close()

res, err := http.Get(ts.URL)
if err != nil {
     t.Fatal(err)
}
bodyBytes, err := ioutil.ReadAll(res.Body)
res.Body.Close()
if err != nil {
     t.Fatal(err)
}

fmt.Printf("%s", bodyBytes)
```

test server has it's own URL, with a special test port that you GET

inspect the response statusCode, body, headers, etc.

QuickCheck

```
import “testing/quick"

func TestOddMultipleOfThree(t *testing.T) {
  f := func(x int) bool {
    y := OddMultipleOfThree(x)
    return y%2 == 1 && y%3 == 0
  }
  if err := quick.Check(f, nil); err != nil {
    t.Error(err)
  }
}

func TestString(t *testing.T) {
  f := func(s string) bool {
    err := ProcessString(s)
    return err == nil
  }
  if err := quick.Check(f, nil); err != nil {
    t.Error(err)
  }
}
```

inject random ints

inject random strings

ANTI-IDOMATIC

- panic: only for truly unrecoverable situations
- syscall: many low-level, os specific primitives
  - ⚠️: Typically use alternatives in os, net, etc.
- unsafe:
  - Used to work around the type system
  - ⚠️: Typically use reflect or go/* packages

Thank You, Credits & License

http://gotutorial.net
@GoTutorialNet

- I owe many many, thanks to the many authors of Go and to Rob Pike in particular.
- These slides are Copyright 2013-2014 Matthew Nunogawa

Matt Nunogawa
@amattn