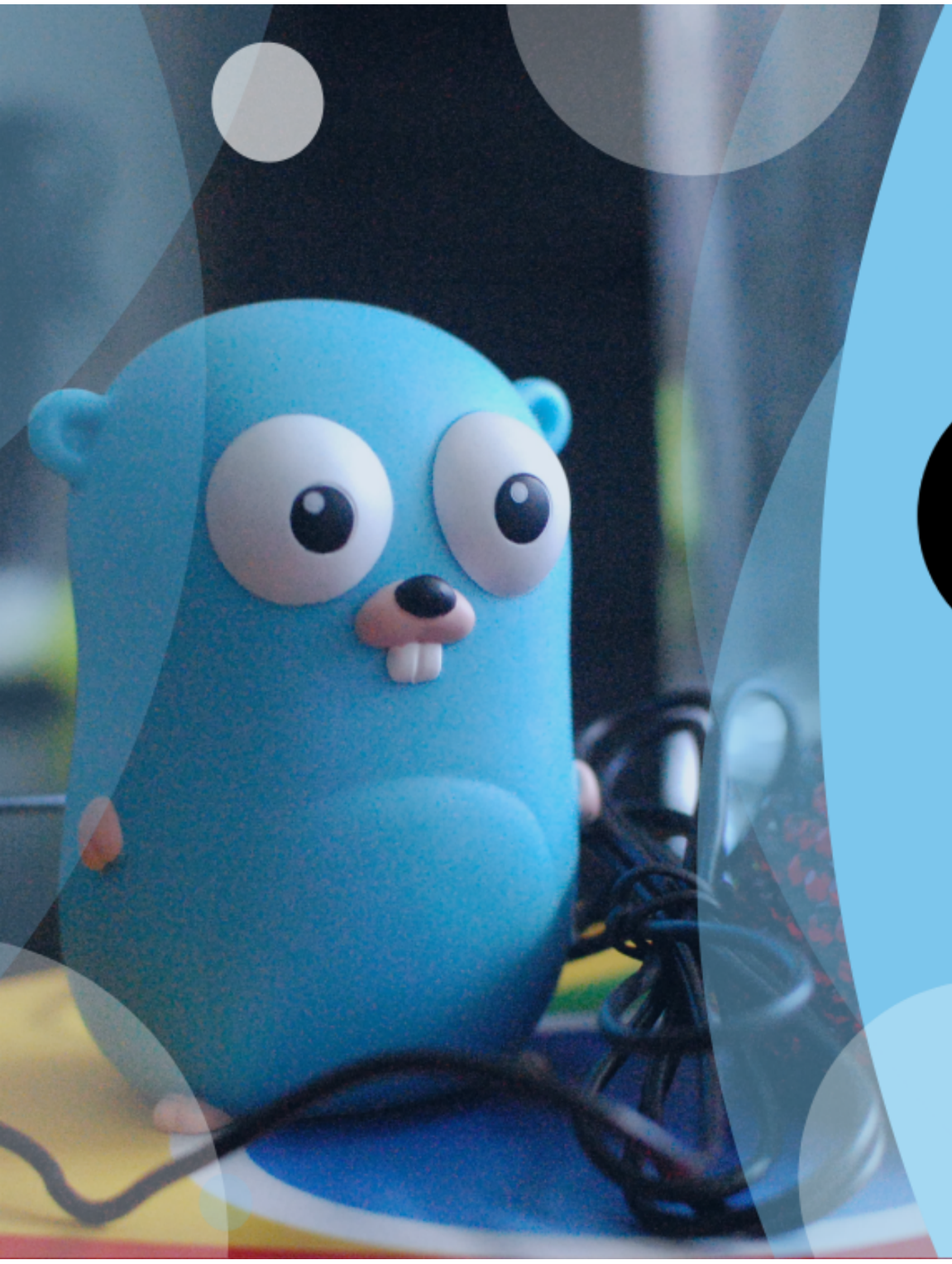

Golang

CHEAT SHEET

pragmaticreviews.com

Golang :: CHEAT SHEET

Installing Golang

Linux

1.- Go to golang.org/dl
2.- Download the installer file
3.- $ sudo tar -C /usr/local -xzf go1.13.5.linux-amd64.tar.gz
4.- $ vim ~/.profile
5.- Append :/usr/local/go/bin to the PATH environment variable.

Building & running Golang apps

```
$ go build
$ go run *.go
```

Mux HTTP Router

- ✓ Package gorilla/mux implements a request router and dispatcher for matching incoming requests to their respective handler.
- ✓ Requests can be matched based on URL host, path, path prefix, schemes, header and query values, HTTP methods or using custom matchers.
- ✓ URL hosts, paths and query values can have variables with an optional regular expression.

How to install Mux

```
$ go get github.com/gorilla/mux
```

How to use Mux

```
// Import the Mux library
import "github.com/gorilla/mux"
// Create a new Mux router instance
router := mux.NewRouter()
// Handle HTTP requests
router.HandleFunc("/", func(w
http.ResponseWriter, r *http.Request) {
                fmt.Fprintln(w, "Up and running...")
        }).Methods(“GET”)
// Start the HTTP server listening on port 8080
http.ListenAndServe(“:8080”, router)
```

Go commands

Complete go command list

```
bug         start a bug report
build       compile packages and dependencies
clean       remove object files and cached files
doc         show documentation for package or
symbol
env         print Go environment information
fix         update packages to use new APIs
fmt         gofmt (reformat) package sources
generate    generate Go files by processing source
get         add dependencies and install them
install     compile and install
packages/dependencies
list        list packages or modules
mod         module maintenance
run         compile and run Go program
test        test packages
tool        run specified go tool
version     print Go version
vet         report likely mistakes in packages
```

Golang Structs

```
type Author struct {
  ID    int64  `json:"id"`
  firstName string `json:"firstname"`
  fastName string `json:"lastname"`
}

type Post struct {
  ID    int64  `json:"id"`
  title string `json:"title"`
  text  string `json:"text"`
  author Author
}
```

Golang Functions

```
func add(x int, y int) int {
        return x + y
}
```

Golang Interfaces

Sample Interface

```
type PostRepository interface {
        Save(post *entity.Post) (*entity.Post, error)
        ...
}
```

How to implement an Interface

```
type repo struct{}

//NewPostRepository creates a new repo
func NewPostRepository() PostRepository {
        return &repo{}
}

func (*repo) Save(post *entity.Post) (*entity.Post,
error) {
        ...
}
```

Goroutines

- ✓ Goroutines are lightweight threads (managed by Go, not O.S. threads).
- ✓ go f(a, b) starts a new goroutine which runs f.

```
// Just a function (can be used as a goroutine)
func doStuff(s string) {
}

func main() {
    // using a named function in a goroutine
    go doStuff("foobar")

    // using an anonymous inner function in a
goroutine
    go func (x int) {
        // function body goes here
    }(42)
}
```

Golang :: CHEAT SHEET

Golang Channels

```
ch := make(chan int) //create a channel of type int
ch <- 42             // Send a value to the channel ch.
v := <-ch            // Receive a value from ch
```

```
// Sending to a nil channel blocks forever
var ch chan string
ch <- "Blocks!!!"
// fatal error: all goroutines are asleep - deadlock!
```

```
// Receiving from a nil channel blocks forever
var ch chan string
fmt.Println(<-ch)
// fatal error: all goroutines are asleep - deadlock!
```

```
// Sending to a closed channel panics
var ch = make(chan string, 1)
ch <- "Hello, Open channel!"
close(ch)
ch <- "Hello, Closed channel!"
// panic: send on closed channel
```

```
// Receiving from a closed channel returns zero
value immediately
var ch = make(chan int, 2)
ch <- 1
ch <- 2
close(ch)
for i := 0; i < 3; i++ {
    fmt.Printf("%d ", <-ch)
}
// 1 2 0
```

Golang Pointers

```
p := Point{1, 2}  // p is a Point
q := &p           // q is a pointer to a Point
r := &Point{1, 2} // r is also a pointer to a Point
// The type of a pointer to a Point is *Point
var s *Point = new(Point) // new creates a pointer
to a new struct instance
```

Errors in Golang

- ✓ There is no exception handling.
- ✓ Functions that might produce an error just declare an additional return value of type Error.

Error Interface

```
type error interface {
    Error() string
}
```

A function that may return an error

```
func doSomething() (int, error) {
}

func main() {
    result, err := doSomething()
    if err != nil {
        // handle error
    } else {
        // no errors, use result
    }
}
```

Maps in Golang

```
var m map[string]int
m = make(map[string]int)
m["key"] = 42
fmt.Println(m["key"])
delete(m, "key")

elem, ok := m["key"] // test if key "key" is present
and retrieve it, if so
// map literal
var m = map[string]Vertex{
    "Bell Labs": {40.68433, -74.39967},
    "Google":    {37.42202, -122.08408},
}
// iterate over map content
for key, value := range m {
}
```

Reflection in Golang

Type Switch

- ✓ A type switch is like a regular switch statement, but the cases in a type switch specify types (not values), and those values are compared against the type of the value held by the given interface value.

```
func doIt(val interface{}) {
  switch v := val.(type) {
  case int:
    fmt.Printf("Twice %v is %v\n", v, v*2)
  case string:
    fmt.Printf("%q is %v bytes long\n", v, len(v))
  default:
    fmt.Printf("I don't know about type %T!\n", v)
  }
}

func main() {
  doIt(88)
  doIt("a string")
  doIt(false)
}
```

Arrays/Slices in Golang

```
var a [10]int // declare an int array with length 10.
Array length is part of the type!
a[3] = 42     // set elements
i := a[3]     // read elements

var a []int   // declare a slice - similar to an array,
but length is unspecified
var a = []int {1, 2, 3, 4}  // declare and initialize a
slice (backed by the array given implicitly)
a := []int{1, 2, 3, 4}                  // shorthand
chars := []string{0:"a", 2:"c", 1: "b"}  // ["a", "b",
"c"]
```