Golang Cheat Sheet

Dainish Jabeen

June 18, 2023

1 Packages

Go uses packages, which can contain multiple files. **The app will start running in the main application.**

Names exported outside the packages, must use a Capital letter.

Golang will auto connect any package in different files.

Listing 1: Golang basics

```
1
2       package main
3
4       import "fmt"
5
6       func main(){
7               fmt.Println("Hello World")
8       }
```

Can use go run . to run a package but should build for production.

2 Basics

2.1 Types

Listing 2: Golang types

```
1
2         := //Declare and initialize non explicit type
3         >> //Shift bitwise right
4         <<  //Shift bitwise left
5
6         //ARRAY
7         name []string
8         var := []string{"blah","meh"}
9
10        //MAPS
11        map[key type]val type //Dict has to be made
12        m := make(map[string]String)
13
14        m["pi"] = 3.14 //Add to map
15
16        elem, ok = m[key] // check key exists
17
18        // Initialization
19
20        var i int // initializes as 0
21
22        // Constants
23
24        const(
25                x=1 // the type of this can change on context
26        )
27
28        p := &i //point to i
29        *p //value of i, changes will change also change i
30
31        type Name struct{
32                x int
33                y int
34        }
35
36        // Pointers to structs
37
38        v := StructName{1,2}
39
40        p = &v
41        p.x = //Will change the value of v
42
43        // Struct constructors
44
45        v := StructName{x:1} //Others members made 0
```

```
46
47          //SLICES
48
49          //Slices acts as pointers
50
51          a[1:] // slice to end
52          s := a[:3] // slice start to 3
53
54          cap(s) // Capacity, elements in underlying array
55
56          // Dynamically size arrays
57
58          a := make(type, len, cap)
59          append(arr, val)
```

2.2 Functions

Listing 3: Functions

```
1
2       func Name(name type) type{}
3
4       //FUNC PARAMS
5
6       func name(x int,y int)
7       func name(x,y int)
8
9       //NAKED RETURN
10
11      func() (x,y int) {
12              x:=1
13              y:=2
14              return
15      } // Will return x and y
```

2.2.1 Funcs as params

Listing 4: Params

```
1
2       func compute(fn func(float64, float64) float64) float64 {
3       return fn(3, 4)
4       }
5
6       func main() {
7               hypot := func(x, y float64) float64 {
8                       return math.Sqrt(x*x + y*y)
9               }
10              fmt.Println(hypot(5, 12))
11
12              fmt.Println(compute(hypot))
13              fmt.Println(compute(math.Pow))
14      }
```

2.2.2 Funcs as closures

Reference var from outside the body and be bound to that value, which when called includes any prev changes to those vars.
Allows a state of memory via an assigned variable to a func of the internal variables of that func.

Listing 5: Params

```
1
2       func adder() func(int) int {
3               sum := 0
4               return func(x int) int {
5                       sum += x
```

```
6                       return sum
7               }
8       }
9
10      func main() {
11              pos, neg := adder(), adder()
12              for i := 0; i < 10; i++ {
13                      fmt.Println(
14                              pos(i),
15                              neg(-2*i),
16                      )
17              }
18      }
```

2.3 Control

Listing 6: Control

```
1
2       for i:=0;i<10;i++{}
3
4       for x<100 {} //Same as while loop
5
6       if statement; cond {}
7
8       switch statement; val {
9
10              case x: //x same as val == x
11
12              case y: //y same as val == y
13
14              default:
15      }
16
17      defer expr //execute expr at the end of func, can stack defers
18
19      for i,v := range arr {} // Loops through array (can do _,v or i,_)
```

3 Methods

Function with a receiver argument, ie accessed through an object (normally a struct). Generally a type through which the function allows for polymorphism.

Listing 7: Methods

```
1
2       type blah struct{
3               x int
4       }
5
```

```
6         func (v blah) Name() int {} // v can access all members of the struct.
7
8         func (v *blah) Name() int {} //Pointer allows changing of structs.
9
10        // v.Name() interpreted as &v.Name()
```

- Pointers good for performance as copying is avoided
- **Should avoid mixing methods of val and pointers**

3.1 Interfaces

Interfaces are a type of data via you can access a set defined number of methods.
This can:

- Define behavior of an object (a type that uses a chosen interface must also have accompanying methods)
- Decouple code, allow use of methods without needing to know the details of the objects
- polymorphism same func does different things based on interface

Listing 8: Interfaces

```
1
2         type Name interface{
3                 method()
4         }
5
6         interface{} // Empty interface can hold any type
7
8         //Use known type from interface object with assertion
9         val.(type)
10
11        // Type Assertion
12
13        var := interface{} = "h"
14
15        s := var.(string) // Gives the value in the interface
16        s := var.(int) // Causes panic
17
18        s,ok := i.(string) // Check
19
20        switch v := i.(type){
21                case int:
22
23                case string:
24        }
```

Fmt uses Stringer interface, that allows the method String() to be accessed via new types.

4 Error

Error method can be extended to use new types, which is auto called if the return type is error.

Listing 9: Error

```
1
2       // Kind of a struct with only a float stored
3       type ErrNegativeSqrt float64
4
5       func (e ErrNegativeSqrt) Error() string {
6               return fmt.Sprint("cannot Sqrt negative number:",
7               float64(e))
8       }
9
10      func Sqrt(x float64) (float64, error) {
11              if x > 0{
12                      return x*x,nil
13              }else {
14                      return 0,ErrNegativeSqrt(x)
15              }
16      }
17
18      func main() {
19              fmt.Println(Sqrt(2))
20              fmt.Println(Sqrt(-2))
21      }
```

5 Type Parameters

Can use T as a generic type in parameters with a constraint.

Listing 10: Type Parameters

```
1
2       //s can be any type that is comparable
3       func Name[T comparable](s T) int{}
4
5       type[T any] struct{
6               next *List[T]
7               val
8       }
9
10      v := List[int]{nil,3} // Before use need to define T
```

6 Concurrency

6.1 Go routines

Lightweight thread managed by the Go routine.

$$go\ f(x,y,z)$$

Eval of f happens in current routine, execution happens in new routine. Happens in the same address space.

6.2 Channels

Type conduit to send/receive values through the routines.

$$ch <- v,\ v := <- ch$$

The data flows in arrows direction. Like maps and slices need to make before use, send and receive will block until channel is ready. Add length to make buffered lengths.

Sends can close the channel so no more vals will be sent. Use ,ok to check if channel is closed, sending on a closed channel causes a panic.

Listing 11: Channels

```
1
2       func fibonacci(n int, c chan int) {
3       x, y := 0, 1
4       for i := 0; i < n; i++ {
5               c <- x
6               x, y = y, x+y
7       }
8       close(c)
9       }
10
11      func main() {
12              c := make(chan int, 10)
13              go fibonacci(cap(c), c)
14              for i := range c { // Receives until c is closed
15                      fmt.Println(i)
16              }
17      }
```

Select with a channel will block until one of its cases can run (a channel is not empty), unless theres a default.

6.3 Mutual exclusion

Listing 12: Mutal exclusion

```
1
2       var mu sync.Mutex
3
4       mu.Lock
5       mu.Unlock
```

Lock will wait for another lock to unlock before continuing, can use defer to unlock. Locks are queued in order.

7 Testing

A test file name, should end with _test.go.

Listing 13: Test example

```
1
2       package example
3
4       import "testing"
5
6       func TestExample(t *testing.T){
7               if testResult != example {
8                       test.Errorf{"This test has failed "}
9               }
10      }
```

Use go test to run the file.

8 Modules

Dependencies are managed via modules, the go.mod file tracks them.

After adding an import to the package the mod file should be tidied.

Listing 14: Module Commands

```
1
2       go mod init [path] // Initalise setup
3       go mod tidy // Add imports to mod file
4
5       go mode -replace [mod name] = [path]
6       // replace path of mod to local file
```

Module path is normally: <prefix>/<descriptive-text>

Prefix: The location ie github.com
Descriptive text: Project name

8.1 Workspaces

Use work spaces to control multiple and locally change modules. Allows reference to packages inside a module outside the module in the workspace.

$$go\ work\ init\ ./module_name$$

$$go\ work\ use\ ./module_name$$

Ensure the local module has the same name .mod file as the package name and then replace the workspace path to local.

$$go\ mod\ edit\ -replace\ [modulename]\ =\ [fullpath]$$

9 APIs

Gin is a HTTP web framework written in golang. TBA

9.1 JSON/HTTP

Client errors need to be unmarshalled and then acted upon.

- Decode into struct with 'json: "meh"' markers
- If data comes in as array will need to unmarshal as an interface array

10 Fuzzing

A method of testing with random injections of data. TBA

11 Logging

Using the package Zap.

Listing 15: Macro definition

```
1
2           import (
3           "go.uber.org/zap"
4           )
5
6           logger, _ = zap.NewProduction()
7           logger := logger.Sugar()
8           //Sugar - Less performance intense version
9           //New Production is a built-in preset
10
11          defer logger.Sync()
12          // Will flush any buffered logs
13
14          //NewProduction():
15
16          {"level":"info","ts":1686135419.8329651,
17          "caller":"GoTests/test.go:13","msg":"Meh"}
18
19          //NewDevelopment():
20
21          2023-06-07T11:57:16.804+0100    INFO
22          GoTests/test.go:13      Meh
23
24          //NewExample():
25
26          {"level":"info","msg":"Meh"}
27
28          logger.Info("Hello World")
29          .Error("Not able to reach blog.
30          ",zap.String("url", "codewithmukesh.com"))
31
32          //example with key-val data pairs
33          sugar.Infow("failed to fetch URL",
34          "url", "http://example.com",
35          "attempt", 3,
```

```
36          "backoff", time.Second,)
37
38          //Using fmt.Sprintf
39          sugar.Infof("failed to fetch URL: %s", "http://example.com")
```

Levels:

- DPanic : Causes logger to panic after log (in development)
- Debug
- Error
- Fatal
- Info
- Panic: Causes logger to panic after log
- Warn

Level Endings:

- None: Uses fmt.Sprint
- f: Uses fmt.Sprintf
- ln: Uses fmt.Sprintln
- w: allows extra values to be added in log

12 Prometheus

Host metrics locally:

Listing 16: Macro definition

```
1
2           package main
3
4           import (
5          "net/http"
6
7           "github.com/prometheus/client_golang/prometheus/promhttp"
8           )
9
10          func main() {
11          http.Handle("/metrics", promhttp.Handler())
12          http.ListenAndServe(":2112", nil)
13          }
```

Add counter/gauge:

Listing 17: Macro definition

```
1
2           var (
3           counterExample = promauto.NewCounter(prometheus.CounterOpts{
4                   Name: "Counter_Example",
5                   Help: "The total number of processed events",
6           })
7                   gaugeExample = promauto.NewGauge(prometheus.GaugeOpts{
8                           Namespace: "",
```

```
                        Subsystem: "",
                        Name: "gauge_example",
                        Help: "Gauge example"
                    })

)

counterExample.Inc() // Increment counter
gaugeExample.Set(99.9)
gaugeExample.Inc()
gaugeExample.Dec() //Decrement value
gaugeExample.Add(10) //Add 10
gaugeExample.Sub(10) //Subtract 10

gaugeExample.SetToCurrentTime()
```

Add histogram, counts observations from configurable static buckets in a burst sample. Which can be used to calculate quantiles.
Data points will be separated out into a chosen bucket.

Listing 18: Macro definition

```

        temps := prometheus.NewHistogram(prometheus.HistogramOpts{
                Name:    "pond_temperature_celsius",
                Help:    "The temperature of the frog pond.",
                Buckets: prometheus.LinearBuckets(20, 5, 5),
                // Start at 20, 5 buckets, each 5 centigrade wide.
        })

        // Simulate some observations.
        for i := 0; i < 1000; i++ {
                temps.Observe(30 +
                math.Floor(120*math.Sin(float64(i)*0.1))/10)
        }

        //Example output

        histogram: <
  sample_count: 1000
  sample_sum: 29969.50000000001
  bucket: <
    cumulative_count: 192
    upper_bound: 20
  >
  bucket: <
    cumulative_count: 366
    upper_bound: 25
  >
  bucket: <
    cumulative_count: 501
    upper_bound: 30
```

```
31    >
32    bucket: <
33      cumulative_count: 638
34      upper_bound: 35
35    >
36    bucket: <
37      cumulative_count: 816
38      upper_bound: 40
39    >
```

13 Cobra - CLI

Cobra is a CLI framework.

Built around commands (actions), args (things) and flags (modifiers).

APPNAME COMMAND ARG –FLAG

Structure:
appName/
– cmd/
— files.go
– main.go

The main files will simply initalise Cobra.

13.1 Example Command

Listing 19: Example command

```
1
2         //Placed in file called ex.go in package cmd
3
4         var exCmd = &cobra.Command{
5                 Use:   "ex",
6                 Short: "Hugo is a very fast static site generator",
7                 Long: `A Fast and Flexible Static Site Generator built
8                 with love by spf13 and friends in Go.
9                 Complete documentation is available at
10                https://gohugo.io/documentation/`,
11
12                Run: func(cmd *cobra.Command, args []string) {
13                  // Do Stuff Here
14                },
15        }
16
17        func Execute() {
18                if err := rootCmd.Execute(); err != nil {
19                fmt.Fprintln(os.Stderr, err)
20                os.Exit(1)
```

```
21                  }
22          }
```

Listing 20: main.go

```
1
2           func main() {
3                   cmd.Execute()
4           }
```

13.2 Sub commands

Can create sub commands using the Add command function:

13.3 Flags

Two types: Persistent flag (available to that command and all subcommands), local command (only applies to that specific command).

Listing 21: Flags

```
1
2           var Verbose bool
3
4           //Persistent
5
6           rootCmd.PersistentFlags().BoolVarP(&Verbose, "verbose",
7           "v", false, "verbose output")
8           rootCmd.Flags().StringVarP(&Verbose, "verbose",
9            "v", false, "verbose output")
10
11          // Flag required
12          rootCmd.MarkFlagRequired("verbose")
13          rootCmd.MarkPersistentFlagRequired("verbose")
14
15          // Flags required together
16          rootCmd.MarkFlagsRequiredTogether("username", "password")
17
18          // Flags mutually exclusive
19          rootCmd.MarkFlagsMutuallyExclusive("json", "yaml")
```

13.4 Arguments

Using the Args field in cobra.Command(), can add validators:

- NoArgs - report error if any args used
- MinimumNArgs(int) - report an error if less than N positional args are provided.
- MaximumNArgs(int) - report an error if more than N positional args are provided.
- ExactArgs(int) - report an error if there are not exactly N positional args.
- OnlyValidArgs - report an error if there are any positional args not specified in the ValidArgs field of Command, which can optionally be set to a list of valid values for positional args.

MatchAll() is used to combine validators.

Can also create custom validator funcs.

13.5 Help

If using subcommands, cobra will auto generate command help, will give list of all subcommands.

Can also get help on specific commands by using command help subcommand.

13.6 Version

cmd.SetVersionTemplate()

13.7 Cobra Generator

Use: **cobra-cli init** to create new cli app.

Edit details in cmd/root.go.

Use: **cobra-cli add command** to create new command files. Add -p 'parentCmd' for a subcommand.

14 Viper

A configurator (TBA)