Run the go file go run main.go

go build – to just compile the code (exe file)

./main

go run- compile and run the file

go fmt – Formats all the code in each file in the current directory

go install – compiles and installs a package

go get – downloads raw source code from someone elses package

go test – runs any tests associated with the current project.

Package (many related files of .go)

2 types of package

Executable(exe) and Reusable (code used as helper. Good place to put reusable logic)

Word main is used to create an executable file

Import fmt math standard packages

Can import reusable packages also calculator, uploader

func is short for function in other languages

func main(arguments){

code to be executed

}

Go is a static typed language

bool, string, int , float64

array – fixed length list

slice – an array that can grow or shrink

everything should be in same data type

creating an extra type called deck

go run main.go deck.go

func(d deck) print(){  
}

d deck is a receiver (d acts like a this)

any variable of the type deck gets access to the method print

this or self is not allowed

slice[startIndex:endIndex+1]

can return multiple values from a function

Type Conversion

[]byte(“Hi there”)

Byte-type we want

“Hi there”- is value we have

Testing

To make a test file create a new file ending \_test.go

Run go test in command line

Function name should start with the word Test

func TestNewDeck()

Type ZeroValue

string “”

int 0

float 0

bool false

for slice without pointers we can update the code s[0]=”Bye”

Value types – int, float,string,bool,structs

Reference types- slices,maps,channels,pointers,functions

Interfaces

Type bot interface{

Args return types

getGreeting(string,int)(string,error)

}

Interfaces are implicit

Are a contract to help us manage types

Body io.ReadCloser – interface

Reader interface is for connecting multiple types to a common data type called byte[] that anyone can work with

**ROUTINES**

go routine executes code line by line

blocking call freezes the code routine

go checkLink(link) – for the blocking code so they are executed almost concurrent(if one thread is blocked. Another one is picked up and worked on)

go uses by default only one cpu core

cpu core

scheduler and routine

multiple cores lead to parallelism

channel needs an input to it and also print the output

receiving messages is also like blocking