Go by example

1.0

```
Package main

import "fmt"

func main () {

        fmt.Println("Hello")

}
```

Packages :- Every go Program is made up of Packages. Packages Serves as a fundamental organizational unit for source code

Package main :- Entry point of the go Program.

import "fmt" :- imports the fmt Package for formatted I/o

Func main () :- is a Special type of function and is the entry point of executable program.

1.1

Variables

var a int = 10

- b := 20

var c, d int

var is used to declare variable of particular type

var declared without initilization are Zero valued.

the := is a short variable declaration operator, the type of the variable is automatically ~~inf~~ inferred by the go compiler

Data types :

Basic : int, float64, string, bool

composite : array, slice, map, struct

(1.2) Control Statements

```
if x > 10 {
        fmt.Println("greater")
} else {
        fmt.Println("lower")
}
```

we can also use just if ~~else~~

```
if a>5 {
        fmt.Println("ok")
    }
```

~~Hello~~

```
switch day {
case 1 :
        fmt.Println("monday")
case 2 :
        fmt.Println("tues day")
default :
        fmt.Println("ok")
```

(1.3) Loops

for is go's only looping construct

most basic way

~~for~~ i ~~:= 10~~ i := 10

for i<10 {

fmt.Println (i)

i = i + 1 }

classic :-

for i := 10 , i < 10 , i++ {

fmt.Println (i)

}

Range :- to do something N times

for ; := range 10 {

fmt.Println ("range", i)

}

for without a condition will loop repeatedly until you break out of the loop

```
for {
        fmt.Print("loop")
        break
    }
```

You can also skip the loop or continue to the next iteration

```
for i := range 6 {
        if i%2 == 0 {
            continue
        }
        fmt.Println(i)
    }
```

1.4

functions

```
func  add (a int, b int ) int {

        return  a + b
}
```

we can also have multipk return

```
func cal ( a, b int ) (int, int {

        return  a+b , a-b
}
```

go requires explicit returns,

the after making a function you can call it just as you expect, with name (arguments)

```
func  main () {

    c :=  add ( 1, 2)
    ~~fmt.Println (1, 2)~~
      fmt. Println ( c )

}
```

recursion :- function calling itself inside the function

A recursion will have ≥ 1 base case or fall back case.

```
// 5 = 5*4*3*2*1

func fact (a int) int {

    if a == 0 {
        return 1          // Base case
    }
    return a * fact (a-1)

}

func main () {

this fact calls itself until it reaches base case ⇔ fact (0)
```

1.5

Pointers

instead of holding data value directly
it holds the location

```
func mult (value *pint) {
        *value = *value * 10
}

func main () {
        a := 50
        fmt.Println("initial a value : ", a)
        mult (&a)
        fmt.Println("Now a value ", a)
```

* : points at the memory

& :

& ; is used to point the address of the variable

* : is used to access and modify the value of the memory address

1.6

Arrays and Slices

```
var a [3]int .

b := [5]int { 1, 2, 3, 4, 5 }

c := [] int { 1, 2, 3 }      // slice
```

arrays have Fixed length

multi dimension :-

```
e = [2] [2] int {
        { 1, 1 },
        { 2, 2 },
    }
```

Slices can have dynamic length

in slices can use append to add element at the end of the slice

```
s := [] int { 1, 2, 3 }

s = append ( s, 4)
```

maps (Hash table)

```
m := make (map [string] int)
```

```
m ["apple "] = 5
m [" grape "] = 2
```

```
n := map [int] int {0:1, 1:2, 2:3}
```

```
var a int

for i := range 3 {

    a = a + n[i]
}
```

maps are hash table

1.6 structs

```
type student struct {
    rollno int
    name string
    grades float
    pass bool
}
```

this is how you make a struct. a struct is a collection of fields of different data type

you can also use that and make a function around the structs

```
func ( a student ) gradeupdate() {
    if a.grades >= 50 {
        a.pass = true
    } else {
        a.pass = false
    }
}
```

methods and receivex

they allow yev to define behavior

```
func (p person) greet() {
    fmt.Println("Hellow", p.name)
}
```

Interface

~~type ... interface~~

```
type person inteface{
    speak() string
}

type male stauct { }

func (a person) speak() string {
    retuan "woof"
}
```

(1.7) Enums with iota

type Day int

```
const (
        Sunday      Day = iota
        monday
        tuesday
)
fmt.Println (Monday)
```

output : 1

iota starts from 0 and increases by 1 automatically

Enums are special case of sum types. An enum is a type that has a fixed number of possible value.