**Variables:**

var variable1 int //declare int type variable "variable1"

fmt.Println(variable1) //print value 0

var variable2 string

fmt.Println(variable2) //print ""

variable1 = 123

variable2 = "hello world"

fmt.Println(variable1, variable2) //print 123 hello world

var variable3 = 123.45

fmt.Println(variable3) //print 123.45

variable4 := false

fmt.Println(variable4) //print false

variable5, variable6 := "multiple", 2

fmt.Println(variable5, variable6) //print multiple 2

var variable7, variable8, variable9 int //declaring multiple variables of same type

variable7, variable8, variable9 = 123, 456, 789

fmt.Println(variable7, variable8, variable9) //print 123 456 789

var(

variable10 int

variable11 string

variable12 bool

) //declaring multiple variables of different type at the same type.

fmt.Println(variable10, variable11, variable12) //print 0 false

variable10, variable11, variable12 = 123, "multiple", true

fmt.Println(variable10, variable11, variable12) //print 123 multiple true

variable13, variable14 := 123, 456

fmt.Println(variable13) //error "variable14 declared and not used"

variable15, \_:= 123, 456 //use of "black identifier"

fmt.Println(variable15) //print 123

**Boolean:**

var is\_Red bool = true

fmt.Printf("%v\n", is\_Red)

**Numeric:**

var int32val int32 = 100

var valint int = 76 //treats as int32 or int64 depends on implementation

fmt.Println(int32val, valint) // print 100 76

var int16val int16 = int64val // error "cannot use int32val <type int32> as type int16 in assignment"

var int16val int16 = int16(int32val)

fmt.Println(int16val) // print 100

var int64val int64 = int32val // error "cannot use int32val <type int32> as type int64 in assignment"

var int64val int64 = int64(int32val)

fmt.Println(int64val) //print 100

var uint8val uint8 = int32val // error "cannot use int32val <type int32> as type uint8 in assignment"

var uint8val uint8 = uint8(int32val)

fmt.Println(uint8val) // print 100

int32val = -100

uint8val = uint8(int32val)

fmt.Println(uint8val) // print 156, as -100 is converting to unsigned integer (using 2's complement)

**String:**

var strval string = "Grow with Golang"

var strcopied string = strval

fmt.Println("original string is:", strval) // prints "original string is: Grow with Golang"

fmt.Println("copied string is:", strcopied) //prints "copied string is: Grow with Golang"

fmt.Println("Address of strval:", &strval)

fmt.Println("Address of strcopied:", &strcopied) // above two lines prints the different memory address of variables

**Pointer:**

var intval int = 12

var pval = &intval

fmt.Println(pval) // prints address of intval

fmt.Println(\*pval) // print 12

\*pval = 78 // change the value of intval through pointer

fmt.Println(intval) // print 78

**Array:**

var arrval[3] string

arrval = ["abc","def","ghi"] // error "syntax error"

arrval[0] = "abc"

arrval[1] = "def"

arrval[2] = "ghi" //will assign the values by accessing it's index

fmt.Println(arrval) //print [abc def ghi]

arrval2 := [3]string{"abc","def","ghi"}

fmt.Println(arrval2) //print [abc def ghi]

**Structures:**

type company struct{

company string

id string

no\_of\_employees int16

}

var comp company //creating variable of type company

var comp2 company

comp.no\_of\_employees = 90 //accessing the members of structure

comp2 = company{"abc pvt. ltd.","SYRP7302",90} // initializing structure

var no\_of\_employees = comp.no\_of\_employees

fmt.Println(no\_of\_employees) //print 90

fmt.Println(comp2.no\_of\_employees) //print 90

fmt.Println(comp2) //print{ abc pvt. ltd. SYRP7302 90}

**Slice:**

slice1 := make([]int,0)

fmt.Println(slice1) //print []

slice2 := make([]int, 3)

slice2[0] = 123

slice2[1] = 456

slice2[2] = 789

fmt.Println(slice2) //print [123, 456, 789]

slice3 := []string{"abc","def","ghi"}

fmt.Println(slice3) //print [abc def ghi]

slice3 = append(slice3, "jkl")

fmt.Println(slice3) //print [abc def ghi jkl]

slice4 := make([]string,5)

copy(slice4, slice3)

fmt.Println(slice4) //print [abc def ghi jkl ]

slice2D := make([][]string, 2,3)

fmt.Println(slice2D) //prints [[] []]

slice2D[0] = []string{"a","b","c"}

slice2D[1] = []string{"d", "e"}

fmt.Println(slice2D) // print [[a b c] [d e]]

**Map:**

var mapval2 = make(map[int]string)

mapval2[1] = "WORLD"

mapval2[2] = "INDIA"

fmt.Println(mapval2) //prints map[1:WORLD 2:INDIA]

mapval := make(map[string]int)

mapval["first"] = 16

mapval["second"] = 32

fmt.Println(mapval) //prints map[first:16 second:32]

**Channel:**

var channelval chan int

fmt.Println(channelval) //print undefined channel as <nil>

channelval2 := make(chan string)

go func() { channelval2 <- "DONE" }() //passing value to function through channel

msg := <-channelval2 //reading channel

fmt.Println(msg) //print DONE

**Numeric constant:**

const untypedConstInt = 123 //untyped integer constant

const untypedConstFLoat = 3.97 //untyped floating-point constant

const typedConstFloat float64 = 4.87 //typed floating point constant

const untypedConstComplex = 56+2i //untyped complex constant

const typedConstComplex complex64 = 56+2i //typed complex constant

const typedConstComplex1 complex64 = untypedConstInt //typed complex constant

fmt.Println(typedConstComplex1) //print 123+0i

const untypedConstrune = 'G'

fmt.Println(untypedConstrune) //print unicode value of 'G' 71

const untypedConstrune1 rune = untypedConstInt

fmt.Println(untypedConstrune1) //print 123

**String constants:**

const untypedConstString = "Golang is a great language" //untyped string constant

const typedConstString string = "Golang is a great language"//typed string constant

type myString string //creating new type having same structure as string

const customName myString = untypedConstString //assigning untyped data

const customName myString = typedConstString //error in assigning typed data “conversion not allowed”

**Boolean constants:**

const untypedConstBoolean = true //untyped boolean constant

const typedConstBoolean bool = false //typed boolean constant

type myBool bool //creating new type having same structure as bool

const customName myBool = untypedConstBoolean //assigning untyped data

const customName1 myBool = typedConstBoolean // error in assigning typed data “conversion not allowed”

**Iota:**

const // iota resets to zero

(

C0 = iota // here iota is zero, increments after coming out from this line

\_ // increments by one, iota is 1 here

C1 = iota // iota is 2

)

fmt.Println("C0:", C0) // print 0

fmt.Println("C1:", C1) // print 2

const // resets iota to zero

(

//comments // doesn't consider this line, so iota would be same.

C2 = iota

)

fmt.Println("C2:", C2) //print 0

**Format specifiers:**

package main

import "fmt"

func main() {

var str = "hello"

var intval = 78

var floatval = 56.8789

var complexval = 67+6i

var boolval = true

var pointerval = &str

fmt.Printf("example of %%v, value is: %v \n",intval)

fmt.Printf("example of %%#v, value is: %#v \n",str)

fmt.Printf("example of %%T, value is: %T \n",intval)

fmt.Printf("example of %%, used to print %%\n\n")

fmt.Printf("example of %%t, value is: %t\n\n", boolval)

fmt.Printf("example of %%b, value is: %b\n", intval)

fmt.Printf("example of %%c, value is: %c\n", intval)

fmt.Printf("example of %%d, value is: %d\n", intval)

fmt.Printf("example of %%o, value is: %o\n", intval)

fmt.Printf("example of %%O, value is: %O\n", intval)

fmt.Printf("example of %%q, value is: %q\n", intval)

fmt.Printf("example of %%x, value is: %x\n", intval)

fmt.Printf("example of %%X, value is: %X\n", intval)

fmt.Printf("example of %%U, value is: %U\n\n", intval)

fmt.Printf("example of %%b, value is: %b\n", floatval)

fmt.Printf("example of %%e, value is: %e\n", floatval)

fmt.Printf("example of %%E, value is: %E\n", complexval)

fmt.Printf("example of %%f, value is: %f\n", floatval)

fmt.Printf("example of %%F, value is: %F\n", floatval)

fmt.Printf("example of %%g, value is: %g\n", floatval)

fmt.Printf("example of %%G, value is: %G\n", floatval)

fmt.Printf("example of %%x, value is: %x\n", floatval)

fmt.Printf("example of %%X, value is: %X\n\n", floatval)

fmt.Printf("example of %%s, value is: %s\n", str)

fmt.Printf("example of %%q, value is: %q\n", str)

fmt.Printf("example of %%x, value is: %x\n", str)

fmt.Printf("example of %%X, value is: %X\n\n", str)

fmt.Printf("example of %%p, value is: %p \n", pointerval)

}