02 Variables

↳ main.go

go mod init variables

```
var username string = "nikhil"
fmt.Println(username)
fmt.Printf("variable is of type : %T", username)

var isLoggedIn bool = true

var smallVal uint8 = 255

var smallFloat float32 = 255.45544511254 4
output = 255.45544
```

↓ in case of float64

```
// default values and some aliases

var anotherVariable int
fmt.Println(anotherVariable)
```

↳ 0

```
// implicit type.
var website = "learncodeonline.in"
fmt.Println(website)
```

Walrus operator (:=)

no var used

We can declare variable without using var and type

Eg. name := "nikhil"

This will work properly inside a method or function.

We can't use walrus operator for global variables.

---

const LoginToken string = "xyz2 - - - - abc"

↓

we use first letter as capital because it is a public variable.

Comma Ok syntax and packages

03userinput

↳ main.go

go mod init userinput

```
package main

func main() {
    welcome := "Welcome to user input"
    fmt.Println(welcome)
}
```

bufio - Package

OS package H.W.

```
reader := bufio.NewReader(os.Stdin)
fmt.Println("Enter the rating for our Pizza:")

// comma ok || comma err

input, _ := reader.ReadString('\n')
```

↓ if there is a error it will be stored in _

because there is no try catch in go.

if we want to consider error only, then we can use something like this.

```
_, err
```

Conversions in Golang

04 conversion

↳ main.go

```
go mod init conversion
fmt.Println("Please rate our pizza between 1 and 5")
reader := bufio.NewReader(os.Stdin)
input, _ := reader.ReadString('\n')
fmt.Println("Thanks for rating, ", input)
```

```
numRating, err = strconv.ParseFloat(input, 64)
```

This will give error because in input if we give 4, then input will be "4 \n" so we have to trip it.

panic(mess)
↳ this will end program

```
numRating, err = strconv.ParseFloat(strings.TrimSpace(input), 64)
```

This will not give error

Handling time in Golang.

06myTime
└→ main.go

```
presentTime := time.Now()
fmt.Println(presentTime)
presentTime.Format("01-02-2006 15:04:05 Monday")
createdDate := time.Date(2020, time.August, 12, 23, 23, 0, 0,
                         time.UTC)
```

`go env`

GOOS="windows" go build

Memory Management.

Memory allocation and Deallocation happens automatically

new()

Allocate memory but not INIT

you will get a memory address

zeroed storage

make()

Allocate memory and INIT

you will get a memory address

non-zeroed storage.

Garbage collection automatically
Out of scope or nil

Pointers

```
var ptr *int
var ptr1 *string
```

value of ptr and ptr1 is <nil> when not initialized.

```
myNumber := 23
var ptr = &myNumber
*ptr = *ptr + 2
fmt.Println("New value is: ", myNumber)
```

Array - Less used in Go

```
var fruitList [4] string
```

↑ it is necessary to provide number of element in array.

Slices — Most used in Go

```
var fruitList = []string{}
```

→ this is required to declare slices

one way to declare slices

we donot provide size here

```
fruitList = append(fruitList, "Mango", "Banana")
fruitList = append(fruitList[1:])
```

store "Banana" in fruitList.

```
fmt.Println(fruitList)
```

→ output = ["Banana"]

```
highScores := make([]int, 4)
highScores[0] = 234
highScores[1] = 945
highScores[2] = 465
highScores[3] = 867
```

If we use `highScores[4] = 912` It will give error But

we can use

highScores = append(highScores, 555, 666, 321)

sort.Ints(highScores)

How to remove a value from slice based on index in golang

var course = []string{"reactjs", "javascript", "swift", "python", "ruby"}

var index int = 2

courses = append(courses[:index], courses[index+1:])

It will error

courses = append(courses[:index], courses[index+1:]...)

It will work properly

MAPS.

languages := make(map[string]string) → for declaring a map

languages["JS"] = "Javascript"

languages["RB"] = "Ruby"

languages["PY"] = "Python"

delete(languages, "RB") → for deleting a key

// loops are interesing in golang

```
for key, value := range languages {
    fmt.Printf("For key %v, the value is %v\n",
                key, value)
}
```

Structs in golang

// no inheritance in golang; No super or parent

```
type User struct {
    Name string
    Email string
    Status bool
    Age int
}
```

```
nikhil := User{"Nikhil", "nikhil@g.com", true, 21}
fmt.Println(nikhil)
fmt.Printf("Nikhil details are : %+v\n", nikhil)
```

we can use . operator for accessing member

Eg. nikhil.Name etc

If else in golang


```
loginCount := 23
var result string
if loginCount < 10 {
    result = "Regular user"
} else if loginCount > 10 {
    result = "something else"
} else {
    result = "Exactly equal 10"
}
```

There is a special syntax

```
if num := 3; num < 10 {
    fmt.Println("Num is less than 10")
} else {
    fmt.Println("Num is NOT less than 10")
}

if err != nil {

}
```

Switch case


```
rand.Seed(time.Now().UnixNano())
diceNumber := rand.Intn(6) + 1

switch diceNumber {
    case 1:
        fmt.Println("Dice value is 1 and you can open")
    case 2:
        fmt.Println("You can move 2 spot")
    default:
        fmt.Println("What was that!")
}
```

fallthrough are used in switch statement

LOOPS

```
vowels := []string{"A", "E", "I", "O", "U"}

for d:=0; d< len(vowels); d++ {
    fmt.Println(vowels[d])
}
```

```
for i := range vowels {
    fmt.Println(vowels[i])
}
```

```
for index, vowel := range vowels {
    fmt.Printf("index is %v and value is %v \n", index, vowel)
}
```

```
rougueValue := 1

for rougueValue < 10 {
    fmt.Println("Value is: ", rougueValue)
    rougueValue++
}
```

We can use goto

```
goto name

name:
    fmt.Println("Hello world")
```

```
package main

func main() {
    greeter()
    result := adder(3,5)
    fmt.Println("Result is: ", result)
    proRes, myMessage := proAdder(2,5,8,7,3)
}

func adder(valOne int, valTwo int) int {
    return valOne + ValTwo
}

func proAdder(values ...int) (int, string) {
    totalVal := 0
    for _, val := range values {
        totalVal += val
    }
    return totalVal, "Hello from proAdder"
}
```

Methods.

```
type User struct {
    Name string
    Email string
    Status bool
}
```

To declare a method for a struct.

```
func (u User) GetStatus() {
    fmt.Println("~~User status~~ Is user active", u.Status)
}

func (u *User) NewEmail() {
    u.Email = "test@email.com"
    fmt.Println("New email is : ", u.Email)
}
```

Here we are passing reference not a copy, so we are using pointer.

Defer statements

① defer keyword puts an element to the last of function.

② If there are many defer ; then they will execute in LIFO manner.

```
func main () {
    defer fmt.Printf (" Hello ")
    fmt. Printf (" World ")
}
```

Output : ⇒

World Hello.

```
func main () {
    defer fmt.Printf (" Hello ")
    fmt.Printf (" World ")
    defer fmt.Printf (" From ")
    defer fmt.Printf (" Nikhil ")
}
```

Output : ⇒ World Nikhil From Hello

Working With Files

```
func main() {
    fmt.Println("Welcome to files")
    content := "This is content of the file"
    file, err := os.Create("./myfile.txt")
    if err != nil {
        panic(err)
    }
    length, err := io.WriteString(file, content)
    if err != nil {
        panic(err)
    }
    fmt.Println("length is: ", length)
    defer file.Close()
    readFile("./myfile.txt")
}
```

```
func readFile(filename string){
    ~~data, err := ioutil.ReadFile(filename)~~
    data, err := ioutil.ReadFile(filename)
    // this data is in byte form.
    if err != nil {
        panic(err)
    }
    fmt.Println("Text data inside the file is \n",
                data)
    // This will give data in byte form
    fmt.Println("Text data inside the file is \n",
                string(data))
}

func checkNilErr(err error){
    if err != nil {
        panic(err)
    }
}
```

Handling web request in golang.

```
const url = "Https://lco.dev"

func main() {
    fmt.Println("LCO web request")

    response, err := http.Get(url)
    if err != nil {
        panic(err)
    }
    fmt.Printf("Response is of type: %T\n", response)
    defer response.Body.Close() // caller's responsibility to close the connection

    databytes, err := ioutil.ReadAll(response.Body)
    if err != nil {
        panic(err)
    }
    content := string(databytes)
    fmt.Println(content)

}
```

Handling URL in GOLANG

```
const urlstring = "https://lco.dev:3000/learn?coursename=reactjs
                   &paymentid=any3216sac"

func main(){
    // parsing the url
    result, err := url.Parse(url)
    checkNilErr(err)
    fmt.Println(result.Scheme)    // https
    fmt.Println(result.Host)      // lco.dev:3000
    fmt.Println(result.Path)      // /learn
    fmt.Println(result.Path())    // 3000
    fmt.Println(result.RawQuery)  // coursename=reactjs&paymentid=

    qparams := result.Query()     // gives key value pair
    fmt.Printf("The type of query params are: %T\n", qparams)
    fmt.Printf(qparams["coursename"])

    for _, val := range qparams{
        fmt.Println("Param is: ", val)
    }

    partsOfUrl := &url.URL{
        Scheme: "https",
        Host: "lco.dev",
        Path: "/tutcss",
        RawPath: "user=hitesh"
    }
    anotherURL := partsOfURL.String()  // we can also wrap up
    fmt.Println(anotherURL)            // inside string to convert it
}
```

Here we are passing a reference