**Regex in Golang**:

A Regular Expression (or RegEx) is a special sequence of characters that defines a search pattern that is used for matching specific text.

There’s a built-in package for regular expressions, called the regexp package which contains all list of actions like filtering, replacing, validating, or extracting.

The MatchString() function reports whether the string passed as a parameter contains any match of the regular expression pattern.

**Syntax:**

func MatchString(pattern string, s string)

**Program:**

package main

import (

"fmt"

"regexp"

)

func main() {

str := "geeksforgeeks"

match1, err := regexp.MatchString("geeks", str)

fmt.Println("Match: ", match1, " Error: ", err)

str2 := "ComputerScience"

match2, err := regexp.MatchString("geeks", str2)

fmt.Println("Match: ", match2, "Error: ", err)

match3, err := regexp.MatchString("geek(s", str2)

fmt.Println("Match: ", match3, "Error: ", err)

}

Output:

Match: true Error: <nil>

Match: false Error: <nil>

Match: false Error: error parsing regexp: missing closing ): `geek(s`

**Using Compile() method in Regex:**

To store complicated regular expressions for reuse later, Compile() method parses a regular expression and returns a Regexp object if successful which can be used to match the text.

Syntax:

func Compile(expr string) (\*Regexp, error)

**Program:**

package main

import (

"fmt"

"regexp"

"strings"

)

func main() {

re, \_ := regexp.Compile("geek")

str := "I love geeksforgeeks"

// returns the slice of first

// and last index

match := re.FindStringIndex(str)

fmt.Println(match)

str2 := "I love computer science"

// prints an empty slice

// as there is no match

match2 := re.FindStringIndex(str2)

fmt.Println(match2)

// finds the first or leftmost

// match to a given pattern.

re2, \_ := regexp.Compile("[0-9]+-v.\*g")

// matches one or more numbers followed

// by v and any number of characters upto g

match3 := re2.FindString("20024-vani\_gupta")

fmt.Println(match3)

// returns a slice of all successive

// matches of the expression

match4 := re.FindAllStringSubmatchIndex("I'am a geek at"+

" geeksforgeeks"1)

fmt.Println(match4)

// returns a copy and replaces

// matches with the replacement string

re3, \_ := regexp.Compile(" ")

match5 := re3.ReplaceAllString("All I do"+

" is code everytime.", "+")

fmt.Println(match5)

// returns a copy in which all matches are

// replaced by return value of function

re4, \_ := regexp.Compile("[aeiou]+")

match6 := re4.ReplaceAllStringFunc("All I do"+

" is code everytime.", strings.ToUpper)

fmt.Println(match6)

}

**Output:**

[7 11]

[]

20024-vani\_g

[[7 11] [15 19] [23 27]]

All+I+do+is+code+everytime.

All I dO Is cOdE EvErytImE.

**Go types**

**Structures in Golang**

A structure or struct in Golang is a user-defined type that allows to group/combine items of possibly different types into a single type

**Declaring a structure**

type Address struct {

name string

street string

city string

state string

Pincode int

}

**Program: Using a dot(.) operator**

package main

import "fmt"

// defining the struct

type Car struct {

Name, Model, Color string

WeightInKg float64

}

// Main Function

func main() {

c := Car{Name: "Ferrari", Model: "GTC4",

Color: "Red", WeightInKg: 1920}

// Accessing struct fields

// using the dot operator

fmt.Println("Car Name: ", c.Name)

fmt.Println("Car Color: ", c.Color)

c.Color = "Black

fmt.Println("Car: ", c)

}

Output:

Car Name: Ferrari

Car Color: Red

Car: {Ferrari GTC4 Black 1920}

**Pointers to a struct**

Go programming language or Golang is a variable which is used to store the memory address of another variable.

Program:

// the pointer to struct

package main

import "fmt"

type Employee struct {

firstName, lastName string

age, salary int

}

func main() {

// taking pointer to struct

emp8 := &Employee{"Sam", "Anderson", 55, 6000}

fmt.Println("First Name: ", emp8.firstName)

fmt.Println("Age: ", emp8.age)

}

Output:

First Name: Sam

Age: 55

-------------------------------------------------------------------------------------------

**Pointers in Golang**

Pointers in Go programming language or Golang is a variable that is used to store the memory address of another variable

**Declaring a pointer:**

var pointer\_name \*Data\_Type

Program:

package main

import "fmt”

func main()

var x int = 5748

var p \*int

p = &x

fmt.Println("Value stored in x = ", x)

fmt.Println("Address of x = ", &x)

fmt.Println("Value stored in variable p = ", p)

}

Output:

Value stored in x = 5748

Address of x = 0x414020

Value stored in variable p = 0x414020

**2.Declaration and initialization of the pointers can be done into a single line.**

Type inference concept is used in this method. There is no need to specify the data type during the declaration. Here we will not use the \* operator.

**Example:**

var s \*int = &a

**Program:**

package main

import "fmt"

func main() {

var y = 458

var p = &y

fmt.Println("Value stored in y = ", y)

fmt.Println("Address of y = ", &y)

fmt.Println("Value stored in pointer variable p = ", p)

}

**Dereferencing the Pointer**

\* operator is also termed as the dereferencing operator. It is not only used to declare the pointer variable but also used to access the value stored in the variable which the pointer points to which is generally termed as indirecting or dereferencing.

**Program:**

// Golang program to illustrate the

// concept of dereferencing a pointer

package main

import "fmt"

func main() {

var y = 458

var p = &y

fmt.Println("Value stored in y = ", y)

fmt.Println("Address of y = ", &y)

fmt.Println("Value stored in pointer variable p = ", p)

fmt.Println("Value stored in y(\*p) = ", \*p)

}

**Changing the value of the pointer or at the memory location instead of assigning a new value to the variable.**

Program:

package main

import "fmt"

func main() {

var y = 458

var p = &y

fmt.Println("Value stored in y before changing = ", y)

fmt.Println("Address of y = ", &y)

fmt.Println("Value stored in pointer variable p = ", p)

fmt.Println("Value stored in y(\*p) Before Changing = ", \*p)

\*p = 500

fmt.Println("Value stored in y(\*p) after Changing = ",y)

}

Output:

Value stored in y before changing = 458

Address of y = 0x414020

Value stored in pointer variable p = 0x414020

Value stored in y(\*p) Before Changing = 458

Value stored in y(\*p) after Changing = 500

**Interfaces in Golang:**

In Go language, the interface is a custom type that is used to specify a set of one or more method

Interface is abstract, so you are not allowed to create an instance of the interface. But you are allowed to create a variable of an interface type and this variable can be assigned with a concrete type value that has the methods the interface requires.

**Syntax:**

type interface\_name interface{

// Method signatures

}

**Program:**

package main

import "fmt"

type tank interface {

Tarea() float64

Volume() float64

}

type myvalue struct {

radius float64

height float64

}

func (m myvalue) Tarea() float64 {

return 2\*m.radius\*m.height +

2\*3.14\*m.radius\*m.radius

}

func (m myvalue) Volume() float64 {

return 3.14 \* m.radius \* m.radius \* m.height

}

func main() {

var t tank

t = myvalue{10, 14}

fmt.Println("Area of tank :", t.Tarea())

fmt.Println("Volume of tank:", t.Volume())

}

Output:

Area of tank : 908

Volume of tank: 4396

**Type Assertion**

type assertion is a process to extract the values of the interface.

**Syntax:**

a.(T)

**Program:**

package main

import "fmt"

func myfun(a interface{}) {

value, ok := a.(float64)

fmt.Println(value, ok)

}

func main() {

var a1 interface {

} = 98.09

myfun(a1)

var a2 interface {

} = "GeeksforGeeks"

myfun(a2)

}

**Output:**

98.09 true

0 false

**Type Switch:**

In Go interface, type switch is used to compare the concrete type of an interface with the multiple types provide in the case statements.

**Program:**

package main

import "fmt"

func myfun(a interface{}) {

switch a.(type) {

case int:

fmt.Println("Type: int, Value:", a.(int))

case string:

fmt.Println("\nType: string, Value: ", a.(string))

case float64:

fmt.Println("\nType: float64, Value: ", a.(float64))

default:

fmt.Println("\nType not found")

}

}

func main() {

myfun("GeeksforGeeks")

myfun(67.9)

myfun(true)

}

**Output:**

Type: string, Value: GeeksforGeeks

Type: float64, Value: 67.9

Type not found

----------------------------------------------------------------------------------------------------