Developed by google

Features

Fast compilation

It is simple, safe, conscious

Support for environment adopting pattern

Lightweight processing

Production of statically linked native binaries without external dependencies

Features excluded intentionally

Support for type inherited

Support for method or operator overloading

Support for circular dependencies among packages

Pointer arithmetic

Assertions

Support for generic programming

How program written

With extension .go

Can use vi or vin editor

The go compiler

Install GO in your relevant PC like linux, windows, mac os

Golang

Packages and modules

Packages are gos way of organizing

Programs are written in as one or more packages

Packages are inported from the go package registery

packages should be focused and perform single thing
 - argument passing
 - Drawing graphics
 - Handling http request

Using packages

import "name"
 for ex
import (
 "name"

"namespace/packagename"
)

Can import everything using dot (.)
No need to reference package name in code
Import can be renamed
import (
 . "name"   // Can importeverything using dot
pk "namespace/packagename"   // can rename package name with pk
)

Modules

Modules are the collection of packages
Created by using the go.mod file in the root directory of your project
Can be managed by go cli
Contain information about your project
Dependancies, go versions, package info
All go program have go.mod file

Example module

```
module example.com/practice
go 1.17
require
(
  github.com/alexflint/go-arg v1.4.2
  github.com/fatih/color v1.13.0
)
```
 Hello world program
```
import "fmt"
 func main() {
        fmt.Println("Hello Beautifull world")
 }
```

String

String are slice of byte.

So string are slices

Go will provide various libraries to manipulate string

      unicode

      regexp

      strings

Creating string

var greeting = "Hello World!"

Check the length of string

fmt.Println(len(greeting))

Concating string

```go
package main


import (

    "fmt"

    "strings"

)



func main() {



    greeting := []string{"Hello", "World"}
```

```go
    fmt.Println(strings.Join(greeting, ""))

    fmt.Printf("%+q\n", greeting)

    fmt.Printf("%x\n", greeting)



}
```

HelloWorld

["Hello" "World"]

[48656c6c6f 576f726c64]

Write a program for number of occurrence of single character in string using algorithm?

```go
package main

import "fmt"
func main() {
    char := "a"
    str := "yagnikpokal"
    counter := 0
    for i := 0; i <= len(str)-1; i++ {
        if string(str[i]) == char {
            counter++
        }
    }
    fmt.Println(counter)
}
/*
go run stringsinglecharacteroccurance.go
2
*/
```

Write a program wether string has a character or not using inbuilt function?

```go
package main

import (
    "fmt"
    "strings"
)
```

```go
func main() {
    str := "yagnikpokal"
    present := strings.Contains(str, "y") // Single character present or not
    fmt.Println(present)
    wordpresent := strings.Contains(str, "yag") // Word present or not
    fmt.Println(wordpresent)
    strcount := strings.Count(str, "a")
    fmt.Println(strcount)
}
/*
go run stringcharacterpresentinbuiltfunction.go
true
true
*/
```

Write a program to count the number of character in string?

```go
package main

import (
    "fmt"
    "strings"
)
func main() {
    str := "yagnikpokal"
    charcount := strings.Count(str, "a") // Count the character
    fmt.Println(charcount)
    wordcount := strings.Count(str, "yag") // Count the word
    fmt.Println(wordcount)
}
/*
go run stringcountcharacterpresentinbuiltfunction.go
2
1
*/
```

Write a program to print the number of repeating character in golang?

Write a program to sort the string?

```go
package main

import (
    "fmt"
    "sort"
    "strings"
)

func sortstring(str string) string {
```

```go
    s := strings.Split(str, "") //Convert string to slice
    // fmt.Println(s)
    sort.Strings(s) //Sort the slice
    // fmt.Println(s)
    //fmt.Println("This", strings.Join(s, ""))
    return strings.Join(s, "") //Convert slice to string
}
func main() {
    str := "yagnikpokal"
    sorted := sortstring(str)
    fmt.Println(str)
    fmt.Println(sorted)
}
/*
go run stringsort.go
yagnikpokal
aagikklnopy
*/
```

Write a program weather the string is anagram or not?

```go
package main

import (
    "fmt"
    "sort"
    "strings"
)
func sortstring(str string) string {
    s := strings.Split(str, "")
    sort.Strings(s)
    return strings.Join(s, "")
}
func main() {
    str1 := "yagnikpokal"
    str2 := "pokalyagnik"
    sorted1 := sortstring(str1)
    sorted2 := sortstring(str2)
    if sorted1 == sorted2 {
        fmt.Println("string is anagram")
    } else {
        fmt.Println("string is not anagram")
    }
}
/*
go run stringanagram.go
string is anagram
*/
```

Write a proram wether string is anagram or not using the map?

```go
package main
```

```go
import "fmt"
func isAnagram(s string, t string) bool {
    lenS := len(s)
    lenT := len(t)
    if lenS != lenT {
        return false
    }
    anagramMap := make(map[string]int)
    for i := 0; i < lenS; i++ {
        anagramMap[string(s[i])]++
    }
    for i := 0; i < lenT; i++ {
        anagramMap[string(t[i])]--
    }
    for i := 0; i < lenS; i++ {
        if anagramMap[string(s[i])] != 0 {
            return false
        }
    }
    return true
}
func main() {
    output := isAnagram("abc", "bac")
    fmt.Println(output)
    output = isAnagram("abc", "bc")
    fmt.Println(output)
}
/*
go run stringanagramwithmap.go
true
false
*/
```

Write a program to calculate the number of vowels and consonants in a string?

```go
package main

import "fmt"
func main() {
    str := "yagnikpokal"
    counter := 0
if 'a'<=
    for i := 0; i < len(str)-1; i++ {
        if string(str[i]) == "a" || string(str[i]) == "e" || string(str[i]) == "i" || string(str[i]) ==
"o" || string(str[i]) == "u" {
            counter++
        }
    }
    fmt.Println("vowels are", counter)
    fmt.Println("consonents are", len(str)-counter)
}
/*
go run stringvowelconsonent.go
vowels are 4
consonents are 7
```

```
*/
```

Write a program weather the given character is letter or digit or special character?

```go
package main

import (
    "fmt"
    "unicode"
)
func main() {
    str := "yagnikpokal123#/*"
    for _, i := range str {
        if unicode.IsDigit(i) {
            fmt.Println(string(i), "is Digit")
        } else if unicode.IsLetter(i) {
            fmt.Println(string(i), "s Letter")
        } else {
            fmt.Println(string(i), "special character")
        }
    }
}
/*
go run stringletterdigit.go
y s Letter
a s Letter
g s Letter
n s Letter
i s Letter
k s Letter
p s Letter
o s Letter
k s Letter
a s Letter
l s Letter
1 is Digit
2 is Digit
3 is Digit
# special character
/ special character
* special character

*/
```

Write a program weather the string is rotational or not?

```go
package main

import (
    "fmt"
    "strings"
)
```

```go
func isSubstring(s1, s2 string) bool {
    if !strings.Contains(s1, s2) {
        return false
    }
    return true
}
func rotation(s1, s2 string) bool {
    if len(s1) != len(s2) {
        return false
    } else {
        s1 = s1 + s1
        return isSubstring(s1, s2)
    }
    return true
}
func main() {
    fmt.Println(rotation("yagnik", "agniky"))
}
/*
go run stringrotationalornot.go
true
*/
```

Remove specific character in string?

or

How do you remove all occurrences of a given character from the input string?

```go
package main

import "fmt"
func main() {
    str := "yagnikpokal"
    char := "a"
    str2 := ""
    for i := 0; i < len(str)-1; i++ {
        if string(str[i]) == string(char) {
        } else {
            str2 += string(str[i])
        }
    }
    fmt.Println(str2)
}
/*
go run stringremovechar.go
ygnikpok
*/
```

Constant & iota

iota is used while working with the constants. Once the iota is declared the value can not be changed. Since it declared with respect to constant. iota keyword uses to assign integers to constants.

iota is the reserved keyword in the golang so we can not use this keyword throughout the program it can only use while declaring the constant.

iota in golang is a declaration of the constant sequence while the repeating sequence is used in the constant declaration. it saves time while doing the programming & improves writing efficiency.

There are a few ways to define the iota. The good thing about the iota is we can skip the sequence in the middle of the counter or we can start the sequence from a particular count number.

Let, us take the example of the beautiful beach of the USA using the iota. Example 1 shows the old way of declaring the constant.

Example 1

Go

```go
package main
import "fmt"// Old method of defining the constant
const (
        Malibu   = 0
        Miami    = 1
        Maryland = 2
        Michigan = 3)
func main() {
        fmt.Println(Malibu, Miami, Maryland, Michigan)}
```

```
0 1 2 3
```

Example 2

Go

```
package main
import "fmt"
// Short iota declaration methodconst (
        Malibu    = iota //0
        Miami            //1
        Maryland         //2
        Michigan         //3)
func main() {
        fmt.Println(Malibu, Miami, Maryland, Michigan)}
```

0 1 2 3

Both the above program will declare constant and the value will be Malibu = 0, Miami = 1, Maryland = 2, and so on.

However, while writing the program it is not the case where each and every time we will go in a sequential manner. Sometimes we must have to skip the value.

Where iota comes into the picture and solves the issues. iota having a 2 declaration methods short declaration and long declaration. example 2 mentioned above will be the shorthand declaration method. example 3 uses a long declaration method.

Example 3

Go

```
package main
import "fmt"
// Long iota declaration methodconst (
        Malibu    = iota //0
        Miami     = iota //1
        Maryland  = iota //2
        Michigan  = iota //3)
func main() {
        fmt.Println(Malibu, Miami, Maryland, Michigan)}
```

0 1 2 3

It is also possible to skip the values in iota. In example 4 we added the _ at positions 2 & 3 hence it will skip the second and third positions and jump towards the fourth position.

Example 4

Go

```go
package main
import "fmt"
// Skip the particular constant valueconst (
        Malibu    = iota //0
        _                         // skip the value by adding an underscore
        _                         // skip the value by adding an underscore
        Miami              //3
        Maryland           //4
        Michigan           //5)
func main() {
        fmt.Println(Malibu, Miami, Maryland, Michigan)}
```

0 3 4 5

We can start the value from a particular number let us say we start at 3 in example 5 then iota will skip the values 0,1,2 and jump towards 3 values and continue.

Example 5

Go

```go
package main
import "fmt"
// Start iota from the perticular valueconst (
        Malibu    = iota + 3 //3        Increment the counter by 3 by adding iota + 3
        Miami                //4
        Maryland             //5
        Michigan             //6)
func main() {
        fmt.Println(Malibu, Miami, Maryland, Michigan)}
```

3 4 5 6

In a real-life example iota will be used as a receiver function to more easily work with multiple sequential constants.

Go slices

Go slices are the abstraction over the go array.

Array will allows you to define several data items of same kind

What is the difference between slice and array?

But does not provide increase the size dynamically or to get sub array of its own

Slices overcome this limitation

var numbers []int /* a slice of unspecified size */

/* numbers == []int{0,0,0,0,0}*/

numbers = make([]int,5,5) /* a slice of length 5 and capacity 5*/

Things need to remember while using the slice

Slices are value to reference over array. hence you can not update same slice data using the append function while using the the function to add a value.

Defining the slice

Declare the array without specifying the size will be slice

Alternatively can create the make function too

```go
package main


import (

    "fmt"

)



func main() {

    number := []int{0, 1, 2, 3, 4}

    var number1 = make([]int, 3, 5) //3 is length and 5 is the capacity
here

    fmt.Println(number)
```

```go
        fmt.Println(len(number), cap(number))

        fmt.Println(number1)

        fmt.Println(len(number1), cap(number1))


    var number2 []int //3 is length and 5 is the capacity here


    if number2 == nil {

        fmt.Printf("SLice is nil \n")

        fmt.Println(number2)



    }

}
```

[0 1 2 3 4]

5 5

[0 0 0]

3 5

SLice is nil

[]

Subslice

Subslice allows to create new lice from current slice use upper bound and lower bound limits as per below

[lower-bound:upper-bound]

```go
package main

```

```go
import "fmt"


func main() {

    numbers := []int{0, 1, 2, 3, 4, 5, 6, 7, 8}

    fmt.Println(numbers)

    fmt.Println(numbers[2:3])

    fmt.Println(numbers[:3])

    fmt.Println(numbers[4:])

    numbers1 := make([]int, 0, 5)

    fmt.Println(numbers1)

    number2 := numbers[1:5]

    fmt.Println(number2)

    number3 := numbers[3:]

    fmt.Println(number3)



}
```

[0 1 2 3 4 5 6 7 8]

[2]

[0 1 2]

[4 5 6 7 8]

[]

[1 2 3 4]

[3 4 5 6 7 8]

Slice append and copy

```go
package main

import "fmt"

func main() {

    var number []int // This will create 8 capacity of array

    //If used short hand method then it will not like that

    fmt.Printf("Len = %d , Cap = %d , slice = %v \n", len(number), cap(number), number)


    number = append(number, 0)

    number = append(number, 1)

    number = append(number, 2)

    number = append(number, 3, 4, 5)

    fmt.Printf("Len = %d , Cap = %d , slice = %v \n", len(number), cap(number), number)

    numbers1 := make([]int, len(number), (cap(number))*2)


    copy(numbers1, number)

    fmt.Printf("Len = %d, Cap = %d , slice = %v \n", len(numbers1), cap(numbers1), numbers1)
```

```
}
```

Len = 0 , Cap = 0 , slice = []

Len = 6 , Cap = 8 , slice = [0 1 2 3 4 5]

Len = 6, Cap = 16 , slice = [0 1 2 3 4 5]

Things need to remember while using the slice

Slices are value to reference over array. hence you can not update same slice data using the append function while using the the function to add a value.

Slice are pointers to underlying array

Below code will not work while you try to append a slice in function.

or

Problems with the slice

```go
package main

import "fmt"
func ModifyData(a []int) {
    a[0] = 5
}
func AddData(a []int) {
    a = append(a, 4)
}
func main() {
    a := []int{1, 2, 3} // Slice
    a = AddData(a)        //{1,2,3,4}
    fmt.Println(a)        // 1,2,3,4
    ModifyData(a)         //5,2,3,4
    fmt.Println(a)        // 5,2,3,4
}
/*
go run sliceappendnotworkfunction.go
# command-line-arguments
.\sliceappendnotworkfunction.go:18:6: AddData(a) (no value) used as value
*/
```

The above problem can be solved by 2 way

1) Using the return slice

```
package main

import "fmt"
func ModifyData(a []int) {
    a[0] = 5
}
func AddData(a []int) []int {
    a = append(a, 4)
    return a
}
func main() {
    a := []int{1, 2, 3} // Slice
    a = AddData(a)        //{1,2,3,4}
    fmt.Println(a)        // 1,2,3,4
    ModifyData(a)         //5,2,3,4
    fmt.Println(a)        // 5,2,3,4
}
/*
go run sliceappendworkfunction.go
[1 2 3 4]
[5 2 3 4]
*/
```

2) Using the pointer

```
package main

import "fmt"
func ModifyData(a []int) {
    a[0] = 5
}
func AddData(a *[]int) {
    *a = append(*a, 4)
}
func main() {
    a := []int{1, 2, 3} // Slice
    AddData(&a)           //{1,2,3,4}
    fmt.Println(a)        // 1,2,3,4
    ModifyData(a)         //5,2,3,4
    fmt.Println(a)        // 5,2,3,4
}
/*
go run sliceappendworkfunctionpointer.go
[1 2 3 4]
[5 2 3 4]
*/
```

Function in go

```
package main

import (
```

```go
    "fmt"

)
```

```go
func main() {

    var s string

    fmt.Println("Hello Beautifull world")

    s = passthestring()

    fmt.Println("String from function is", s)

}
```

```go
func passthestring() string {

    return "Goodbye"

}
```

Hello Beautifull world

String from function is Goodbye

```go
package main


import (

    "fmt"

)
```

```go
func main() {

    //var s string

    //var t string

    fmt.Println("Hello Beautifull world")

    s, t := passthestring()

    fmt.Println("String from function is", s, t)

}
```

```go
func passthestring() (string, string) {

    return "Goodbye", "World"

}
```

Hello Beautifull world

String from function is Goodbye World

Function as a pointer

```go
package main


import (

    "fmt"

)
```

```go
func main() {

    var a string

    passString(&a)

    fmt.Println("Value come from function is", a)

}
```

```go
func passString(b *string) {

    *b = "name"

}
```

Value come from function is name

```go
package main

import (

    "fmt"

)

func main() {

    var x int = 5748

    var p *int

    p = &x
```

```
        fmt.Println(x)

        fmt.Println(&x)

        fmt.Println(p)

    }
```

5748

0xc0000aa058

0xc0000aa058

Function array

```
package main

import "fmt"

func main() {

    a := []int{0, 5, 3, 3}

    var j [4]*int

    for i := 0; i < 4; i++ {

        j[i] = &a[i]

    }

    for i := 0; i < 4; i++ {

        fmt.Println(i, *j[i])

    }

}
```
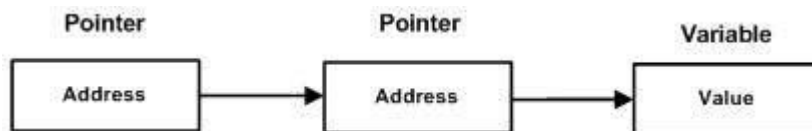
0 0

1 5

2 3

3 3

Pointer on pointer in go

A pointer to a pointer is a form of chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.


```go
package main


import "fmt"


func main() {

    var a int = 10

    var ptr *int

    var pptr **int


    ptr = &a

    pptr = &ptr

    fmt.Println(a)

    fmt.Println(*ptr)

    fmt.Println(**pptr)
```

```
}
```

10

10

10

Passing pointer to function

```go
package main

import "fmt"

func main() {

    var x int = 100

    var y int = 200



    fmt.Println(x, y)



    swap(&x, &y)

    fmt.Println(x, y)

}

func swap(a *int, b *int) int {

    var tmp int

    tmp = *b

    *b = *a

    *a = tmp
```

```
    return 0

}
```

100 200

200 100

Structor

```go
package main

import (

    "fmt"

    "time"

)


type book struct {

    author      string

    name        string

    revision    int

    yearrelease time.Time

}


func main() {
```

```
    bookdetail := book{

        author:    "yagnik",

        name:      "Golang",

        revision: 3,

    }



    fmt.Println(bookdetail.author)

    fmt.Println(bookdetail.name)

    fmt.Println(bookdetail.revision, bookdetail.yearrelease)



}
```

yagnik

Golang

3 0001-01-01 00:00:00 +0000 UTC

Structor as a function

You can pass a structure as a function argument in very similar way as you pass any other variable or pointer. You would access structure variables in the same way as you did in the above example –

```go
package main

import "fmt"

type book struct {

    author    string

    publisher string

    revision  int
```

```go
}


func main() {

    var book1 book

    var book2 book


    book1.author = "yagnik"

    book1.publisher = "milman"

    book1.revision = 1


    book2.author = "mahamad"

    book2.publisher = "billigine"

    book2.revision = 2

    print_func(book1)

    print_func(book2)


}


func print_func(book_detail book) {

    fmt.Printf("%s \n", book_detail.author)

    fmt.Printf("%s \n", book_detail.publisher)

    fmt.Printf("%d \n", book_detail.revision)
```

```
}
```

yagnik

milman

1

mahamad

billigine

2

Structor as a pointer function

You can define pointers to structures in the same way as you define pointer to any other variable as follows –

var struct_pointer *Books

struct_pointer = &Book1;

struct_pointer.title;

```go
package main

import "fmt"

type book struct {

    author    string

    publisher string

    revision  int

}
```

```go
func main() {

    var book1 book

    var book2 book


    book1.author = "yagnik"

    book1.publisher = "milman"

    book1.revision = 1


    book2.author = "mahamad"

    book2.publisher = "billigine"

    book2.revision = 2

    print_func(&book1)

    print_func(&book2)



}


func print_func(book_detail *book) {

    fmt.Printf("%s \n", book_detail.author)

    fmt.Printf("%s \n", book_detail.publisher)

    fmt.Printf("%d \n", book_detail.revision)



}
```

yagnik

milman

1

mahamad

billigine

2

Datatypes

It is the way that program can interpret the binary numbers

For ex numbers, letters,

Go uses type interferance to determine what type of data it is working with

Signed integer

```
int8 -128 to 127
int16 -32768 to 32767
int             //int and int32 both are 32 bit by default
int32
int64
```

Unsigned integers

```
uint8 0 to 255
uint16 0 to 65535
uint                    //uint and uint32 both are same
uint32
uint64
byte 0 to 255
uintptr 0 to ptr size
```

Other datatypes

```
float32
float64
complex64
complex128
```

bool   true or false

Hello world in go

Package main   /* package declaration */
Import "fmt"   /* preprocessor 8/
Func main()
{
fmt.Println("Hello world")
}

Go program structure
It contains following parts
- Package declaration
- Import packages
- Functions
- Variables
- Statements and expressions
- Comments

Go will runs with packages
Each package has its path and name associated with it

Token in go
Token is either keyword, an identifier, constants, string literature, or a symbol
For ex below statement consists of six tokens
fmt.Println("Hello World!")

For example individual tokens are
Fmt
Println
(
"Hello World"
)

Line separator
fmt.Println("Hello, WOrld")
fmt.Println("I am in go programming world!")

Comments
/* my first program in go */

Identifier
Identifier = letter {letter | unicode_digit}.
Go does not allow the punctuation character such as @, $, %

Go is the case sensitive programming language
Thus Manpower and manpower are 2 different identifiers
Here are some of the acceptable identifiers
mahesh kumar abc move_name a_123
myname50 _temp j a3b9 retvl

Keywords in go
break
default
func
interface
select
case
defer
Go
map
Struct
chan
else
Goto
package
Switch
const
fallthrough
if
range
Type
continue
for
import
return
Var

Whitespaces in Go
It will used in go to describe blanks, tabs, new line characters and comments etc.
Line containing only white spaces possibly with a comments is known as blank line
var age int;
fruit = apples + oranges;   //Get the total fruits
No white spaces is necessary between fruit and = or between = and apples
It is free to include if you wish for readability purposes

GO Datatypes
Boolean Consists of 2 predefined constants a true b false
Derived Arithmetics types, integer types or floating point types

string Sequence of byte It is immutable types Not possible to change the type of the string
numeric pointer, array, structor, union, function, slice, map, channel

Go type conversion
Type conversion is the way to convert one data type to another datatype.
If need to store the long value into simple integer then can type cast long to int
type_name(expression)

```go
package main

import "fmt"

func main() {
    var value1 int = 17
    var value2 int = 5
    var output float32

    output = float32(value1) / float32(value2)
    fmt.Printf("Value of output is %f", output)
}
```
Value of output is 3.400000

Go Array
Go supports data structor called array
Which store fixed sequential bytes of same type of element
Declaration of array
var var_name [size] type.
var var_name [size] type{value1, value2, value3}
var name[3] string
var balance = [5]float32{1.1, 2.3, 5.4, 17.5, 5.2}

var balance = []float32{1.1, 2.3, 5.4, 17.5, 5.2}

var balance[4] = 17.5

```go
package main

import "fmt"

func main() {
```

```go
    var n [11]int
    var i, j int

    for i = 0; i < 10; i++ {

        n[i] = i + 100
    }

    for j = 0; j < 10; j++ {

        fmt.Println(j, n[j])
    }
}
```

```
0 100
1 101
2 102
3 103
4 104
5 105
6 106
7 107
8 108
9 109
```

```go
package main

import (
    "fmt"
)

func main() {
    array := []string{"my", "name", "is", "yagnik"}

    /*
        array = []string
        array[0] = "my"
        array[1] = "name"
        array[2] = "is"
        array[3] = "yagnik"
        fmt.Println("Elements of Array:")
```

```
        fmt.Println("Element 1: ", array[2])
    */
    // printing simple array
    for i := 0; i < 4; i++ {
        fmt.Printf(array[i])
    }
}
```

mynameisyagnik

```go
package main

import "fmt"

func main() {

    // 5 row 2 column
    a := [5][2]int{{0, 0}, {1, 5}, {9, 5}, {6, 2}, {7, 2}}

    for i := 0; i < 5; i++ {
        for j := 0; j < 2; j++ {

            fmt.Printf("a[%d][%d] = %d \n", i, j, a[i][j])
        }

    }
}
```

a[0][0] = 0
a[0][1] = 0
a[1][0] = 1
a[1][1] = 5
a[2][0] = 9
a[2][1] = 5
a[3][0] = 6
a[3][1] = 2
a[4][0] = 7
a[4][1] = 2

Write a program to sort the array?

```go
package main
```

```go
import (
    "fmt"
    "sort"
)
func main() {
    arr := []int{5, 4, 7, 9, 5, 3, 54, 6, 5, 69, 2}
    fmt.Println(arr)
    sort.Ints(arr)
    fmt.Println(arr)

    stringarr := []string{"c", "y", "a"}
    fmt.Println(stringarr)
    sort.Strings(stringarr)
    fmt.Println(stringarr)
}
/*
go run arraysort.go
[5 4 7 9 5 3 54 6 5 69 2]
[2 3 4 5 5 5 6 7 9 54 69]
[c y a]
[a c y]
*/
```

Write a program to insert and delete the element in array?

```go
package main

import "fmt"
func main() {
    arr := []int{10, 52, 32, 46}
    fmt.Println(arr)
    arr[1] = 16 // Copying/inserting the value
    fmt.Println(arr)
    arr = append(arr, 0) // Making space for the new element
    fmt.Println(arr)
    copy(arr[3:], arr[2:]) // copy + Shifting elements
    fmt.Println(arr)
    arr = append(arr[:2], arr[3:]...) //This will delete second element
    fmt.Println(arr)
    index := 3
    arr = append(arr[:index], arr[index+1:]...) //This will delete Third element
    fmt.Println(arr)
}

/*
go run arrayinsertdelete.go
[10 52 32 46]
[10 16 32 46]
[10 16 32 46 0]
[10 16 32 32 46]
[10 16 32 46]
[10 16 32]

*/
```

Write a program to delete the specific index of array?

```go
package main

import "fmt"
func delete_index(arr []int, a int) []int {
    return append(arr[:a], arr[a+1:]...)
    //return arr
}
func main() {
    arr := []int{10, 5, 6, 7, 9, 2}
    fmt.Println(arr)
    arr = delete_index(arr, 2) //Delete the index no 2
    fmt.Println(arr)
}

/*
go run arraydeleteindex.go
[10 5 6 7 9 2]
[10 5 7 9 2]
*/
```

Write a program to reverse the array?

```go
package main

import "fmt"
func main() {
    s := []int{5, 2, 6, 3, 1, 4}
    fmt.Println(s)
    for i, j := 0, len(s)-1; i < j; i, j = i+1, j-1 {
        s[i], s[j] = s[j], s[i]
    }
    fmt.Println(s)
}
/*
go run arrayreverse.go
[5 2 6 3 1 4]
[4 1 3 6 2 5]

*/
```

Write a program to remove the element from the array.

```go
package main

import "fmt"
func main() {
    arr := []int{0, 1, 2, 2, 3, 0, 4, 2}
    num := 2
    fmt.Println(arr)
    for j := 0; j <= len(arr)-1; j++ {
        for i := 0; i <= len(arr)-1; i++ {
            if arr[i] == num {
                arr = append(arr[:i], arr[i+1:]...)
            }
        }
    }
```

```
    }
    fmt.Println(arr)
}
/*
go run arrayremovelement.go
[0 1 2 2 3 0 4 2]
[0 1 3 0 4]
*/
```

Write a program to add all the element of array?

```
package main

import "fmt"
func main() {
    arr := []int{0, 1, 2, 3, 6, 5, 4}
    sum := 0
    for i := 0; i < len(arr); i++ {
        sum += arr[i]
    }
    fmt.Println(sum)
}
/*
go run arraysum.go
21
*/
```

Write a program wether the given number is prime or not?

```
// You can edit this code!
// Click here and start typing.
package main

import (
    "fmt"
    "math/big"
)
func main() {
    const n = 1212121
    if big.NewInt(n).ProbablyPrime(0) {
        fmt.Println(n, "is prime")
    } else {
        fmt.Println(n, "is not prime")
    }
}
/*
go run primenumber.go
1212121 is prime
*/
```

Write a program to find the second largest number from the array?

```
package main

import (
    "fmt"
    "sort"
)
```

```go
func main() {
    arr := []int{0, 2, 3, 5, 4, 15, 13, 6, 7}
    D := arr
    sort.Ints(D)
    fmt.Println(D[len(D)-2])
}
/*
go run arraysecondlargestnumber.go
13
*/
```

Write a program to implement bubble sort algortithem in golang?
or
bubblesort algorithem.
6 2 8 4 10
2 6 8 4 10
2 6 4 8 10
2 4 6 8 10
// Just like a bubble
// It will use only 2 elements and based on that 2 elements find the maximum element and put it left side
Time complexity -->> O(N2)
Space   -->> O(1)
Worst and Average Case Time Complexity: O(N2). The worst case occurs when an array is reverse sorted.
Best Case Time Complexity: O(N). The best case occurs when an array is already sorted.
Auxiliary Space: O(1)

```go
package main

import "fmt"
func bubblesort(arr []int) []int {
    for i := 0; i < len(arr)-1; i++ {
        for j := 0; j < len(arr)-1-i; j++ {
            if arr[j] > arr[j+1] {
                arr[j], arr[j+1] = arr[j+1], arr[j]
            }
        }
    }
    return arr
}
func main() {
    arr := []int{4, 5, 6, 98, 7, 4, 3}
    sorted := bubblesort(arr)
    fmt.Println(arr)
    fmt.Println(sorted)
}
/*
go run sortarraybubblesort.go
[3 4 4 5 6 7 98]
[3 4 4 5 6 7 98]
```

Insertion sort algorithem

Searching algorithms
1) Linear search
2) Binary search
3) Jump search
Linear search
Iterate over the array and find the elements and once find return it.
Time complexity O(n)
Space complexity O(1)

```go
package main

import "fmt"
func linearsearch(arr []int, digit int) int {
    for index, value := range arr {
        if value == digit {
            return index
        }
    }
    return -1
}
func main() {
    arr := []int{1, 2, 3, 5, 7, 8}
    digit := 2
    index := linearsearch(arr, digit)
    fmt.Println(index)
}
/*
go run arraylinearsearch.go
1
*/
```

Binary search
Binary serach only work with sorted arrays.
Method:-
1) Find the middle element
2) If our value is> middle element then divide second half into quarte half
3) If our value is <middle element then divide first half into quarter half
4) If you get value then return it

1, 2, 3, 5, 7, 8 find the element 2
1, 2, 3,   5, 7, 8   // Divide into 2 part
1, 2, 3,   5, 7, 8   // Middle value is 3
1, 2, 3,   5, 7, 8   // Our element is 2 then 2<3 so element is in first half
1, 2, 3   // Find the middle part
1, 2, 3   // Middle part Is 2 our element is also 2 so we found it and return it
2   // 2 is the our element

```go
package main

import "fmt"
func binarySearch(arr []int, s int) int {
    var leftPointer = 0
    var rightPointer = len(arr) - 1
    for leftPointer <= rightPointer {
        var midPointer = int((leftPointer + rightPointer) / 2)
        var midValue = arr[midPointer]
        if midValue == s {
            return midPointer
        } else if midValue < s {
            leftPointer = midPointer + 1
        } else {
            rightPointer = midPointer - 1
        }
    }
    return -1
}
func main() {
    var n = []int{2, 9, 11, 21, 22, 32, 36, 48, 76}
    fmt.Println(binarySearch(n, 32))
}
/*
go run arraybinarysearch.go
5
*/
```

Time complexity O(log n)
Space complexity O(1)

References
https://www.simplilearn.com/coding-interview-questions-article

Go will allows multi dimensional array
var var_name[size1] [size2] [size3] …. [sizen] variable_type
 2D array
var arrayName [x][y] variable_type
initialization of 2D array
a = [3] [4] int{
{0,1,2,3},
{4,5,3,6},
{8,4,3,7}
}

Go pointers
Go tasks easily perform by the pointer

Some cases such as call by reference will not perform without pointer
Every variable has a memory location
Memory location has address and can be accesses by & which is the address of mlocation
A **Pointers are the variable whose value is the address of another memory location**
var var_name *var-type

```go
package main

import "fmt"

func main() {
    var x int = 20
    var y *int
    fmt.Println(y) // THis is nil pointer where we hav not allocated the
adress just we initialized
    y = &x
    fmt.Println(&x)
    fmt.Println(y)
    fmt.Println(*y)


}
```

<nil>
0xc000014088
0xc000014088
20

Passing array to function

void myFunction(param [10]int)
{
.
.
.
}

void myFunction(param []int)
{
.
.
.

}

If you want to pass a single-dimension array as an argument in a function, you would have to declare function formal parameter in one of following two ways and all two declaration methods produce similar results because each tells the compiler that an integer array is going to be received. Similar way you can pass multi-dimensional array as formal parameters.

```go
package main

import "fmt"

func main() {
    a := []int{0, 100, 52, 30}

    x := average(a, 4)
    fmt.Println(x)
}

func average(y []int, size int) float64 {
    var b float64
    var sum int
    for i := 0; i < size; i++ {
        sum += y[i]
    }
    b = (float64)(sum / size)
    return b
}
```

45

Literature
Integer literals
It can be decimal, octal, hexadecimal constant.
0x or 0X for hexa decimal
0 for octal
Nothing for decimal

212   // legal decimal
0213   // octal
0x4b   // hexadecimal
30l   // long
30ul   // unsigned long
215u   // legal unsigned integer

0xFeeL // legal
078   // illlegal octal digit
032UU   // illegal octal digit

Floating-point literature
It is the part of floating point, fractional point and exponent part
3.14159   // legal
31459E-5L   // legal
510E   // illegal
210F   // illegal
.e55   // illegal

String literature in go
"Hello, Dear"
" Hello, \
dear"
"hello,"

Const literature
const var type = value;
const LENGTH = 10
const WIDTH = 5

Go scope rules
  Local variable
  Globle variable
  Firmal parameters

Local variable
Inside the function is called as local variable

```go
import "fmt"

/* global variable declaration */
var g int

func main() {
   /* local variable declaration */
   var a, b int
```

```
   /* actual initialization */
   a = 10
   b = 20
   g = a + b

   fmt.Printf("value of a = %d, b = %d and g = %d\n", a, b, g)
}
```
value of a = 10, b = 20 and g = 30

Globle variable

But local variable inside the main has higher preference hence output will be 10 instead of 20

```
package main

import "fmt"

/* global variable declaration */
var g int = 20

func main() {
   /* local variable declaration */
   var g int = 10

   fmt.Printf ("value of g = %d\n",  g)
}
```
value of g = 10

Formal parameters

Formal parameters says always stick to value in main variable if we used the same value in any other function.

Let us say variable a is declared in global and local both

And same called by the function

Then function will take priority from local only. See the below program

Formal parameters are treated as local variables with-in that function and they take preference over the global variables. For example −

```
package main

import "fmt"

/* global variable declaration */
var a int = 20;
```

```go
func main() {
    /* local variable declaration in main function */
    var a int = 10    // This value is always a preference
    var b int = 20
    var c int = 0

    fmt.Printf("value of a in main() = %d\n",  a);
    c = sum( a, b);
    fmt.Printf("value of c in main() = %d\n",  c);
}
/* function to add two integers */
func sum(a, b int) int {
    fmt.Printf("value of a in sum() = %d\n",  a);
    fmt.Printf("value of b in sum() = %d\n",  b);

    return a + b;
}
```

value of a in main() = 10
value of a in sum() = 10
value of b in sum() = 20
value of c in main() = 30

For loop as while loop

```go
package main

import "fmt"

func main() {
    var i int32
    i = 0
    for i < 5 {

        fmt.Println("This loop runs five time")
```

```
        i++
    }


}
```

This loop runs five time
This loop runs five time
This loop runs five time
This loop runs five time
This loop runs five time

For loop as a do while loop

```go
//There is no do while loop in the go
// There are few ways with the help of for loop we can define do loop
package main

import "fmt"

func main() {
    var i int = 0
    for {
        fmt.Println("This loop will run 5 times", i)
        i++
        if i >= 5 {
            break
        }
    }


}
```

PS C:\Go_WorkSpace\forasdowhile> go run forasdowhile.go
This loop will run 5 times 0
This loop will run 5 times 1
This loop will run 5 times 2
This loop will run 5 times 3
This loop will run 5 times 4

Break statement in go

```go
package main

import "fmt"

func main() {
```

```go
    var i int = 10
    for {
        fmt.Println(i)
        i++
        if i > 15 {
            break
        }

    }
}
```

PS C:\Go_WorkSpace\breakloop> go run break.go
10
11
12
13
14
15

```go
/*package main

import "fmt"

func main() {

    var i int = 10

    for i < 20 {
        i++
        fmt.Println(i)
        //continue
    }
}*/

package main

import "fmt"

func main() {
    /* local variable definition */
    var a int = 10
```

```
    /* do loop execution */
    for a < 20 {
        if a == 15 {
            /* skip the iteration */
            a = a + 1
            continue
        }
        fmt.Printf("value of a: %d\n", a)
        a++
    }
}
```

value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19

Go to statement

```
package main

import "fmt"

func main() {
    learnGoTo()
}
func learnGoTo() {
    fmt.Println("a")
    goto FINISH
    fmt.Println("b")
FINISH:
    fmt.Println("c")
}
```

PS C:\Go_WorkSpace\goto> go run goto.go
a

c

Go range
Range keyword is used to iterate over items of an array, slice, channel or map.
With array and slice it will return the index of the item as integer.
With maps it will return the key of the next key-pair.
Range either return the once value or two.

Range expression	1st Value	2nd Value(Optional)
Array or slice a [n]E	index i int	a[i] E
String s string type	index i int	rune int
map m map[K]V	key k K	value m[k] V
channel c chan E	element e E	none

```go
package main

import "fmt"

func main() {
    // creating a slice
    numbers := []int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    for i := range numbers {
        fmt.Println(numbers[i])
    }

    mystateCitymap := map[string]string{"gujarat": "ahmedabad",
"kernataka": "banglore", "maharastra": "mumbai"}

    //Print map using keys
    for state := range mystateCitymap {
```

```
        fmt.Println("capital city of", state, "is", mystateCitymap[state])
    }

    //Print map using the key value
    for state, city := range mystateCitymap {
        fmt.Println("capital city of", state, "is", city)
    }

}
```

1
2
3
4
5
6
7
8
9
10
capital city of gujarat is ahmedabad
capital city of kernataka is banglore
capital city of maharastra is mumbai
capital city of gujarat is ahmedabad
capital city of kernataka is banglore
capital city of maharastra is mumbai

Function returns the maximum value

```
package main

import "fmt"

func main() {

    var a int = 10
    var b int = 20
    c := max(a, b)
    fmt.Println(c)
}

func max(num1 int, num2 int) int {
    var result int
```

```
    if num1 > num2 {
        result = num1
    } else {
        result = num2
    }
    return result


}
```

20

Swap the value with function and passing 2 values to function and get 2 values from function
```go
package main

import "fmt"

func main() {
    a, b := swap("casey", "jacob")
    fmt.Println(a, b)


}

func swap(value1, value2 string) (string, string) {

    return value2, value1
}
```
jacob casey

Call by value function
```go
package main

import (
    "fmt"
)

func main() {
    var i int = 10
    var j int = 20
    fmt.Println("before swap", i)
    fmt.Println("before swap", j)
```

```
    swap(i, j)
    fmt.Println("after swap", i)
    fmt.Println("after swap", j)
    // output will not change after and before swap
}
func swap(value1, value2 int) int {
    var temp int
    temp = value1
    value1 = value2
    value2 = temp
    return temp
}
```

before swap 10
before swap 20
after swap 10
after swap 20

Call by reference function

```
package main

import (
    "fmt"
)

func main() {
    var i int = 10
    var j int = 20
    fmt.Println("before swap", i)
    fmt.Println("before swap", j)

    swap(&i, &j)
    fmt.Println("after swap", i)
    fmt.Println("after swap", j)
    // output will not change after and before swap
}
func swap(value1 *int, value2 *int) int {
    var temp int
    temp = *value1
    *value1 = *value2
```

```
    *value2 = temp
    return temp
}
```

before swap 10
before swap 20
after swap 20
after swap 10

Regular expression regexp

Regular expression is a special sequence of the character that define a search pattern that used for matching the specific text.
Regular expression is only deal with the string operations.
There are three types of operations performed by the regular expression.
1) Filtering or matching or validating
2) Replacing
3) Find index of matched string
4) Find string
Regexp uses RE2 syntex standard
The MatchString() function reports whether the string passed as a parameter contains any parameters of the regular expression pattern.
To store the complicated regular expressions for reuse later purpose Compile() method parses the regular expressions and returns Regexp object.

Filtering or matching

```go
package main

import (
    "fmt"
    "regexp"
)
func main() {
    str := "geeksforgeeks"
    value, err := regexp.MatchString("geeks", str)
    fmt.Println(value, err)
    value1, err := regexp.MatchString("yagnik", str)
    fmt.Println(value1, err)
}
/*
go run regexpsimple.go
true <nil>
false <nil>
*/
```

Replacing

```go
package main

import (
    "fmt"
    "regexp"
    "strings"
)
func main() {
    re, _ := regexp.Compile(" ")
    replace := re.ReplaceAllString("my name is yagnik", "+")
    fmt.Println(replace)
    // Replace all the characters to uppercase using the function
    re1, _ := regexp.Compile("[aeiou]+")
    replace1 := re1.ReplaceAllStringFunc("My name is yagnik", strings.ToUpper)
    fmt.Println(replace1)
}
/*
go run regexpreplacestring.go
my+name+is+yagnik
My nAmE Is yAgnIk
*/
```

Find index (validating or extracting)

```go
package main

import (
    "fmt"
    "regexp"
)
func main() {
    re, _ := regexp.Compile("geeks")
    str := "geeksforgeeks"
    myIndex := re.FindStringIndex(str) //Shows first index of the charcter matching string
    fmt.Println(myIndex)
    myIndex1 := re.FindAllStringSubmatchIndex("geeks for geeks", -1) // Shows the first and last character
index of the matching string
    fmt.Println(myIndex1)
}

/*
go run regexpfindindex.go
[0 5]
[[0 5] [10 15]]
*/
```

Find string first and last character

```go
package main

import (
    "fmt"
    "regexp"
```

```
)
func main() {
    re2, _ := regexp.Compile("[0-9]+-y.*g") // This will print the string when first char is y and got
second char g
    extract1 := re2.FindString("1994-yagnik_pokal")
    fmt.Println(extract1)
}

/*
go run regexpfindstring.go
1994-yag
*/
```

Referances
https://www.geeksforgeeks.org/what-is-regex-in-golang/
In depth https://www.geeksforgeeks.org/how-to-split-text-using-regex-in-golang/?ref=rp

Go - functions as values

Go programming language provides the flexibility to create functions on the fly and use them as values.

```go
package main

import (
    "fmt"
    "math"
)

func main() {

    getSquareRoot := func(x float64) float64 {
        return math.Sqrt(x)
    }

    fmt.Println(getSquareRoot(9))

}
```
3

Go function closure
Go support anonymous function which can acts as a function closure.
```go
package main
```

```go
import "fmt"

func return_increment() func() int {
    i := 0
    return func() int {
        i += 1
        return i
    }
}

func main() {
    a := return_increment()
    fmt.Println(a())
    fmt.Println(a())
    fmt.Println(a())

    b := return_increment()
    fmt.Println(b())
    fmt.Println(b())
}
```

```
1
2
3
1
2
```

Method in Go
Go programming language supports special types of functions called methods. In method declaration syntax, a "receiver" is present to represent the container of the function. This receiver can be used to call a function using "." operator.
Things need to remember while using method
Method contains receiver's argument where function doesn't this is the **difference.**
Receiver type must be in a same package. Methos will not work in the different package then where defined. If you try to do that then compiler will give error.

For example −

```go
package main
```

```go
import (
    "fmt"
    "math"
)

/* define a circle */
type Circle struct {
    x, y, radius float64
}

/* define a method for circle */
func (circle Circle) area() float64 {
    return math.Pi * circle.radius * circle.radius
}

func main() {
    circle := Circle{x: 0, y: 0, radius: 5}
    fmt.Printf("Circle area: %f", circle.area())
}
```

Circle area: 78.539816

Difference between method and function

Method	Function
It contains a receiver.	It does not contain a receiver.
Methods of the same name but different types can be defined in the program.	Functions of the same name but different type are not allowed to be defined in the program.
It cannot be used as a first-order object.	It can be used as first-order objects and can be passed

Go error handling
Go will prvides pretty simple error handling framework.
With inbuilt error handling type of following declaration
Inthe function we will use return 2 times.

Now will check if there is error then will give that error, if no error then will give the actual return value.
Both time in return value we will give 2 parameters
in error return will return with 0 + error y default
in actual value will return actual value + nil (Because in main we check if it is nil then there is error else not)

```go
package main

import (
    "errors"
    "fmt"
    "math"
)

func Sqrt(value float64) (float64, error) {
    if value < 0 {
        return 0, errors.New("Math: negative number passed to Sqrt")
    }
    return math.Sqrt(value), nil
}
func main() {
    result, err := Sqrt(-1)

    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println(result)
    }

    result, err = Sqrt(9)

    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println(result)
    }
}
```

Math: negative number passed to Sqrt
3

Go recursion

Recursion is the process of repeating the item in selfsimilar way

```go
func recursion(){
recursion()
}

func main(){
recursion()
}
```

Use of factorial in go with recursion

```go
package main

import "fmt"

func factorial(i int) int {

    if i <= 1 {

        return 1
    }
    return i * factorial(i-1)
}

func main() {

    x := factorial(4)
    fmt.Println(x)

}
```

24

```go
package main

import "fmt"

func fibonacci(i int) int {
    if i == 0 {
```

```go
        return 0
    }

    if i == 1 {
        return 1
    }

    return fibonacci(i-1) + fibonacci(i-2)

}

func main() {
    var i int
    for i = 0; i < 10; i++ {
        fmt.Printf("%d \n", fibonacci(i))
    }

}
```

0
1
1
2
3
5
8
13
21
34

Map
Delete function in map

```go
package main

import "fmt"

func main() {
    stateofCityMap := map[string]string{"Gujarat": "Ahmedabad",
"Maharastra": "Mumbai"}

    for state := range stateofCityMap {
```

```
        fmt.Println("state of", state, "is", stateofCityMap[state])

    }
    delete(stateofCityMap, "Gujarat")


    fmt.Println("After delete")


    for state := range stateofCityMap {
        fmt.Println("state of", state, "is", stateofCityMap[state])

    }


}
```

state of Gujarat is Ahmedabad
state of Maharastra is Mumbai
After delete
state of Maharastra is Mumbai

Go Interfaces
Go provides another datatype called as interface
It represents a set of method signatures
The struct data type implements these interface to have a method definition for the method of
the interface
Things need to remember while using interface.
Use of the interface?
1) When you want to create your own type then interface can be used. (Example of area)
2) Go has a different approach to implement the concept of object orientation.Go don't have
class and inheritance.Go fulfill these requirement by powerful **interface.**
3) When use heap then to sort, pop and push elements interface is very use full.

Syntax
type interface_type name{
method name1[return type]
method name2[return type]
method name3[return type]
}

type struct_name struct{
variables
}

// implement interface methods
func (struct_name_variable struct_name) method_name1() [return type]{
```

```
// Method implementation
}

func (struct_name_variable struct_name) method_name() [return type] {
// Method implementation
}
```

```go
package main

import (
    "fmt"
    "math"
)

/* define an interface */
type Shape interface {
    area() float64
}

/* define a circle */
type Circle struct {
    x, y, radius float64
}

/* define a rectangle */
type Rectangle struct {
    width, height float64
}

/* define a method for circle (implementation of Shape.area())*/
func (circle Circle) area() float64 {
    return math.Pi * circle.radius * circle.radius
}

/* define a method for rectangle (implementation of Shape.area())*/
func (rect Rectangle) area() float64 {
    return rect.width * rect.height
}

/* define a method for shape */
func getArea(shape Shape) float64 {
```

```
        return shape.area()
}


func main() {
    circle := Circle{x: 0, y: 0, radius: 5}
    rectangle := Rectangle{width: 10, height: 5}

    fmt.Printf("Circle area: %f\n", getArea(circle))
    fmt.Printf("Rectangle area: %f\n", getArea(rectangle))
}
```

Circle area: 78.539816
Rectangle area: 50.000000

Interface is of 2 types
1) Static
2) Dynamic
The static interface is interface itself
The variable of the interface contains value of the type. Which implments the interface, So the value of type T is called as dynamic value or dynamic type.

Referance,
https://www.geeksforgeeks.org/interfaces-in-golang/
https://www.javatpoint.com/go-interface

Go unit testing
Command to test the unit tests for function
```
go test
```
If needed all tests result then
```
go test -v
```

To run the specific test cases below command is used.
```
go test -run TestNameOfTheFunction
```

If you have a multiple test cases and want to run all of them then use below
```
go test -run .
```

To get all the tests result displayed on console with pass fail
```
go test -v .
```

To get single line pass of all tests
```
go test .
```

To run a test in specific file called render_test.go then use below command. Most of the time this is not a desired way because the package have a dependancy on the other package. And by firing below command other package is not visible.

```
go test render_test.go
```

To check the coverage of the unit test case use below command

```
go test -cover
```

Test your code during the development will expose the bugs
Go's built in function will makes easier to test as you go
It uses the go testing commands and go testing packages
Go supports **automated testing** for unit testing for all go packages.
The function of form necessory is for TDD test driven developement

```
func TestXxx(t *testing.T){}
```

Go unittesting with multiple functions

```go
//File name sum.go
package main

import (
    "fmt"
)
func Sum(a int, b int) int {
    return a + b
}
func Sub(a int, b int) int {
    return a - b
}
func main() {
    D := Sum(5, 65)
    fmt.Println(D)
    E := Sum(65, 6)
    fmt.Println(E)
}
```

Create unit testing

```go
// File name is sum_testing.go
package main

import "testing"
func TestSum(t *testing.T) {
    x := 5
    y := 10
    want := Sum(x, y)
    get := 15
    if get != want {
        t.Errorf("get %d, want %d", get, want)
    }
}
func TestSub(t *testing.T) {
    x := 65
```

```go
        y := 5
        get := Sub(x, y)
        want := 60
        if get != want {
            t.Errorf("get %d, want %d", get, want)
        }
    }


    /*
    To display wether all the test passes or not
    go test
    PASS
    ok      shortenurl/unittest     0.358s

    // Second command if need to display all testcases with result
    go test -v
    === RUN   TestSum
    --- PASS: TestSum (0.00s)
    === RUN   TestSub
    --- PASS: TestSub (0.00s)
    PASS
    ok      shortenurl/unittest     0.401s

    // GO command to check the coverage of the package
    go test -cover
    PASS
    coverage: 33.3% of statements
    ok      shortenurl/unittest     0.341s


    // Go command to check a specific or specific set of functions testing
     go test -v -run "TestSub|TestAdd"
    === RUN   TestSub
    --- PASS: TestSub (0.00s)
    PASS
    ok      shortenurl/unittest     0.340s


    */
```

TDD with Go
In TDD we will write a developer will write the test cases first for enhancement of new features
before writing code
The basic premises of TDD is that you begin with writing failed test cases to be implemented
Then you write most straight full pass code to pass that test cases.
The new code will be reworked or refracted
TDD = Refactoring + TFD
TFD is test fail scenario
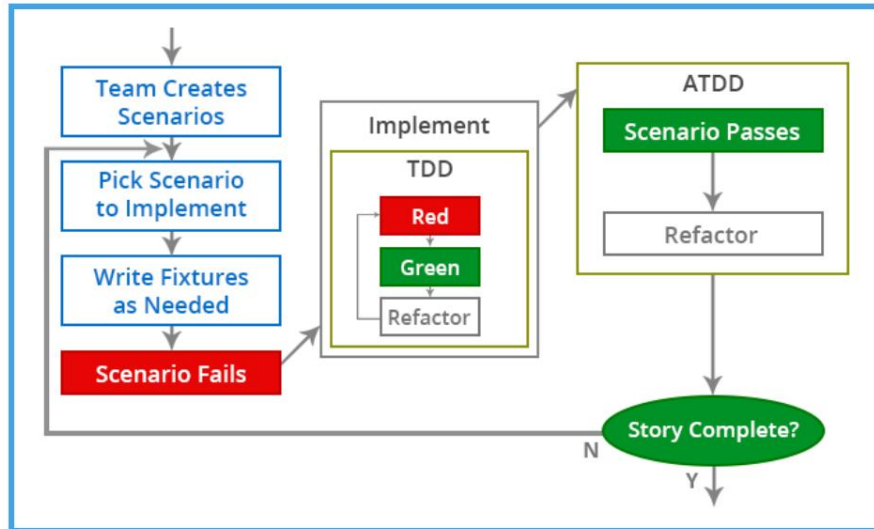
More topics to be covered for the **unit testing**
- Go httptest
- Go test example functions
- Go table driven tests

Go testing referance
https://www.xenonstack.com/blog/test-driven-development-golang
https://zetcode.com/golang/testing/

Go logging
What to log
- Spot bugs in application
- Discover performance problems
- Do the postmortem analysys of outage and security incidents

Some time you needed to log
- Time stamp
- Log level such as debug, error or info
- Contextual data to understand what happen to make it possible to easily reproduce data.

What not to log
- Names
- IP Adress
- Credit card numbers

As per GDPR and HIPPA logging data

Introducing the log package
package main
import "log"

```
func main(){
log.Println("Hello World")
}
2019/12/09 17:21:53 Hello World
```

Logging to a file

The below file will create the log file with the name of text.

```go
package main

import (
    "log"
    "os"
)

func main() {
    file, err := os.OpenFile("logs.txt",
os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0666)
    if err != nil {
        log.Fatal(err)
    }
    log.SetOutput(file)
    log.Println("Hello World")

}
```

```
∨ logging
  Go logging.go
  ≡ logs.txt
```
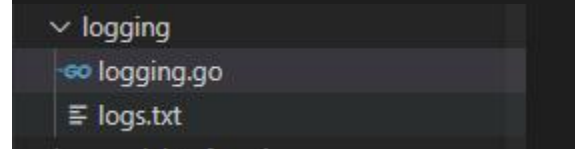
Go Database operations

Importing a database driver & module
Driver for the sql is sql.Open()
There are drivers for the sqlite3 and postgres to

Before executing the below program there are few things need to install
    1) SQL server
mysql-installer-community-8.0.29.0
https://cdn.mysql.com//Downloads/MySQLInstaller/mysql-installer-community-8.0.29.0.msi

2) Microsoft SQL server management studio

SSMS-Setup-ENU

```go
package main
import (
    "database/sql"
    "fmt"

    _ "github.com/go-sql-driver/mysql"
)

func main() {

    db, err := sql.Open("mysql",
"root:root@tcp(127.0.0.1:3306)/employeedb")
    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println("Connection established")

    }
    defer db.Close()
}
```

Connecion established

Go directives

-   **Retract directive**

It means draw back or with draw

Let us assume we publish our module using version control mechanism

In one module suppose did a mistake and released to production with number v0.1.0

After that realise a mistake and publish a new version with v0.2.0

We cant modify the cose in v0.1.0

And there is no way to tell the people that use v0.2.0

This problem will solved by the retract module

Can upgrade module

Can downgrade modules

-   **Go module directives**

Applicable in and after version 1.13 of go

It is the new way of adding libraries called go modules
Go module solves the gopath problems

```go
package main

import (
    "fmt"

    "mymodule/mypackage"
)

func main() {
    fmt.Println("Hello, Modules!")

    mypackage.PrintHello()
}
```

```go
package mypackage

import "fmt"

func PrintHello() {
    fmt.Println("Hello, Modules! This is mypackage speaking!")
}
```

Directory: C:\Go_WorkSpace\projects\mymodules

Mode	LastWriteTime	Length Name
----	------------	------ ----
d-----	10-05-2022 02:18 PM	mypackage
-a----	10-05-2022 02:14 PM	25 go.mod
-a----	10-05-2022 02:18 PM	142 main.go

Taken reference from https://www.digitalocean.com/community/tutorials/how-to-use-go-modules

Adding a remote module as a dependencies

```
go get github.com/spf13/cobra@07445ea

module mymodule

go 1.16

require (
        github.com/inconshreveable/mousetrap v1.0.0 // indirect
        github.com/spf13/cobra v1.1.2-0.20210209210842-07445ea179fc // indirect
        github.com/spf13/pflag v1.0.5 // indirect
)


go get github.com/spf13/cobra@v1.1.1

module mymodule

go 1.16

require (
        github.com/inconshreveable/mousetrap v1.0.0 // indirect
        github.com/spf13/cobra v1.1.1 // indirect
        github.com/spf13/pflag v1.0.5 // indirect
)


go get github.com/spf13/cobra@latest

module mymodule

go 1.16

require (
        github.com/inconshreveable/mousetrap v1.0.0 // indirect
        github.com/spf13/cobra v1.2.1 // indirect
        github.com/spf13/pflag v1.0.5 // indirect
)
```

- **Replace directory**

Replace directory will replace the content of the specific version of the midule from other wheres.

If the version present on the left side of the arrow only that specific version is replaced
Replace directory only applied on the main modules go.mod file, ignored by others
If there is multiple main than it will appy to all
right habd side begin with ./ or ../ then it is local path for replacement

Example:

replace golang.org/x/net v1.2.3 => example.com/fork/net v1.4.5

replace (
    golang.org/x/net v1.2.3 => example.com/fork/net v1.4.5
    golang.org/x/net => example.com/fork/net v1.4.5 //Module with no local path
    golang.org/x/net v1.2.3 => ./fork/net
    golang.org/x/net => ./fork/net   // Local path
)

-   **Exclude directory**
Exclude directory prevents the module version loaded by go command
Before Go 1.16 exclude version was reference by the required directory
Exclude directory only applied in main go.mod it will be ignored by others

ExcludeDirective = "exclude" (ExcludeSpec | "(" newline { ExcludeSpec } ")" newline) .
ExcludeSpec = ModulePath Version newline .

Example:
exclude golang.org/x/net v1.2.3

exclude (
    golang.org/x/crypto v1.4.5
    golang.org/x/text v1.6.7
)

–   **Require directory**
Required directory declares the minimum required version of the given module dependencies.
Go command loads the go.mod file for required version & incorporate requirement from the file
Go command will automatically adds // indirect comments

Which indicates that no package from the required modules is directly imported by any package in main module

RequireDirective = "require" (RequireSpec | "(" newline { RequireSpec } ")" newline) .
RequireSpec = ModulePath Version newline .

Example:
require golang.org/x/net v1.2.3

require (
    golang.org/x/crypto v1.4.5 // indirect
    golang.org/x/text v1.6.7
)

- **Go directives**

Go directive indicates that a module was written assuming the semantic of a given version of go. The version is like 1.9, 1.14 etc

The go directive originally intended to support backward incompatibility changes to the go language.

There have no been incompatible language changes since modules was introduced, go directory still affects new language supports

The go.mod file after 1.17 includes an explicit require directive for each module that provides any package transively import by package or test in main module.

As of the Go 1.17 release, if the go directive is missing, go 1.16 is assumed.

GoDirective = "go" GoVersion newline .
GoVersion = string | ident . /* valid release version; see above */
Example:

go 1.14

GIN Framework

This will introduced the basic of writing the RESTFULL web services API with go and gin web framework.

GIN will simplifies many coding tasks associated with building with web applications, including web services.

Here we will write the GIN to route request, retrieve request, details and marshal json for responses.

Here we also build RESTful API server with 2 end points.

Example project will be a repository of data about vintage jazz record.

- **Design API end point**

API provides an access to a store selling vintage recording the end points.

Hence need to provide endpoints through which a client can get and add album for users.

/album
- GET   Get a list of album, return as JSON
- POST Add a new album from request data sent as json

/albums/:id
- GET   Get an albums by ID, returning album data as json.

Tools
- **Race detection**

Race conditions are most insidious and elusive programming errors. It often long after the code deployed to mass production

Go 1.1 includes the race detection a new tool for finding the race conditions in go code.

It is currently support linux and windows

The race detector is based on the c/C++ thread sanitizer runtime library. Which will used to detect many errorsin googles internal codes.

when -race command-line will be set the compiler instruments all memory access and record, and see how the memory accessed.

Race enables binaries will use 10 times CPU and memory

Commands

go test -race mypkg   // Test the package

go run -race mysrc.go   //Compile and run the program

go build -race mycmd   //build the command

go install -race mypkg //install the package

package main

```go
import "fmt"

func main() {
    done := make(chan bool)
    m := make(map[string]string)
    m["name"] = "world"
    go func() {
        m["name"] = "data race"
        done <- true
    }()
    fmt.Println("Hello,", m["name"])
    <-done
}
```
go run -race racy.go

```go
 func main() {
11     start := time.Now()
12     var t *time.Timer
13     t = time.AfterFunc(randomDuration(), func() {
14         fmt.Println(time.Now().Sub(start))
15         t.Reset(randomDuration())
16     })
17     time.Sleep(5 * time.Second)
18 }
19
20 func randomDuration() time.Duration {
21     return time.Duration(rand.Int63n(1e9))
22 }
23
```

panic: runtime error: invalid memory address or nil pointer dereference
[signal 0xb code=0x1 addr=0x8 pc=0x41e38a]

```
goroutine 4 [running]:
time.stopTimer(0x8, 0x12fe6b35d9472d96)
    src/pkg/runtime/ztime_linux_amd64.c:35 +0x25
time.(*Timer).Reset(0x0, 0x4e5904f, 0x1)
    src/pkg/time/sleep.go:81 +0x42
main.func·001()
    race.go:14 +0xe3
created by time.goFunc
    src/pkg/time/sleep.go:122 +0x48
```

The race detector shows the problem: an unsynchronized read and write of the variable t from different goroutines.

To fix the race condition we change the code to read and write the variable t only from the main goroutine:

```
10  func main() {
11      start := time.Now()
12      reset := make(chan bool)
13      var t *time.Timer
14      t = time.AfterFunc(randomDuration(), func() {
15          fmt.Println(time.Now().Sub(start))
16          reset <- true
17      })
18      for time.Since(start) < 5*time.Second {
19          <-reset
20          t.Reset(randomDuration())
21      }
22  }
```

Here the main goroutine is wholly responsible for setting and resetting the Timer t and a new reset channel communicates the need to reset the timer in a thread-safe way.

Go performance tool

Go has a lot of performance tool available for the CPU utilization and time usage. The common tool is
https://gitlab.com/steveazz-blog/go-performance-tools-cheat-sheet
One of the most convinient method to see is use benchmark tool built in go

go test -bench=. -test.benchmem ./rand/

```
goos: darwin
goarch: amd64
pkg: gitlab.com/steveazz/blog/go-performance-tools-cheat-sheet/rand
cpu: Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz
BenchmarkHitCount100-8            3020          367016 ns/op        269861 B/op
3600 allocs/op
BenchmarkHitCount1000-8            326         3737517 ns/op       2696308 B/op
36005 allocs/op
BenchmarkHitCount100000-8           3       370797178 ns/op      269406189 B/op
3600563 allocs/op
BenchmarkHitCount1000000-8          1      3857843580 ns/op     2697160640 B/op
36006111 allocs/op
PASS
ok      gitlab.com/steveazz/blog/go-performance-tools-cheat-sheet/rand 8.828s
```

Note: -test.benchmem is an optional flag to show memory allocations

Comparing Benchmarks
Go created perf which provides benchstat so that you can compare to benchmark outputs together and it will give you the delta between them.
For example, let's compare the main and best branches.
```
# Run benchmarks on `main`
git checkout main
go test -bench=. -test.benchmem -count=5 ./rand/ > old.txt

# Run benchmarks on `best
git checkout best
go test -bench=. -test.benchmem -count=5 ./rand/ > new.txt

# Compare the two benchmark results
benchstat old.txt new.txt
name            old time/op   new time/op    delta
HitCount100-8       366µs ± 0%     103µs ± 0%  -71.89%  (p=0.008 n=5+5)
```

```
HitCount1000-8      3.66ms ± 0%    1.06ms ± 5%  -71.13%  (p=0.008 n=5+5)
HitCount100000-8     367ms ± 0%     104ms ± 1%  -71.70%  (p=0.008 n=5+5)
HitCount1000000-8    3.66s ± 0%     1.03s ± 1%  -71.84%  (p=0.016 n=4+5)


name            old alloc/op   new alloc/op   delta
HitCount100-8       270kB ± 0%     53kB ± 0%  -80.36%  (p=0.008 n=5+5)
HitCount1000-8      2.70MB ± 0%   0.53MB ± 0%  -80.39%  (p=0.008 n=5+5)
HitCount100000-8     270MB ± 0%     53MB ± 0%  -80.38%  (p=0.008 n=5+5)
HitCount1000000-8   2.70GB ± 0%   0.53GB ± 0%  -80.39%  (p=0.016 n=4+5)


name            old allocs/op  new allocs/op  delta
HitCount100-8       3.60k ± 0%    1.50k ± 0%  -58.33%  (p=0.008 n=5+5)
HitCount1000-8       36.0k ± 0%    15.0k ± 0%  -58.34%  (p=0.008 n=5+5)
HitCount100000-8     3.60M ± 0%    1.50M ± 0%  -58.34%  (p=0.008 n=5+5)
HitCount1000000-8    36.0M ± 0%    15.0M ± 0%  -58.34%  (p=0.008 n=5+5)
```

Notice that we pass the -count flag to run the benchmarks multiple times so it can get the mean of the runs.

Benchmarks

You can generate profiles using benchmarks that we have in the demo project.
CPU:
go test -bench=. -cpuprofile cpu.prof ./rand/

Memory:
go test -bench=. -memprofile mem.prof ./rand/

Go Static code analysis

Static code analysis is the greatest tool to find the issues related to the security, performance, coverage, coding style, and some time even logic running without the running your application.

When invoked with the `-analysis` flag, godoc performs static analysis on the Go packages it indexes and displays the results in the source and package views. This document provides a brief tour of these features.

Type analysis features

`godoc -analysis=type` performs static checking similar to that done by a compiler: it detects ill-formed programs, resolves each identifier to the entity it denotes, computes

the type of each expression and the method set of each type, and determines which types are assignable to each interface type. **Type analysis** is relatively quick, requiring about 10 seconds for the >200 packages of the standard library, for example.

Compiler errors

If any source file contains a compilation error, the source view will highlight the errant location in red. Hovering over it displays the error message.

```
39
40   // Handler implements a WebSocket handler for a client connection.
41   var Handler = websocket.Handler(socketHandler)
42
     undeclared name: websocket
43   // Environ provides an environment when a binary, such as the go tool, is
44   // invoked.
45   var Environ func() []string = os.Environ
```

Identifier resolution

In the source view, every referring identifier is annotated with information about the language entity it refers to: a package, constant, variable, type, function or statement label. Hovering over the identifier reveals the entity's kind and type (e.g. `var x int` or `func f func(int) string`).

```
64   func (c *serverCodec) ReadRequestHeader(r *rpc.Request) error {
65           c.req.reset()
66           if err := c.dec.Decode(&c.req); err != nil {
67                   return err
68           }
69           r.ServiceMethod = c.req.Method
70           field ServiceMethod string
71           // JSON request id can be any JSON value;
```

```
64   func (c *serverCodec) ReadRequestHeader(r *rpc.Request) error {
65           c.req.reset()
66           if err := c.dec.Decode(&c.req); err != nil {
67                   return err   func (*encoding/json.Decoder).Decode(v interface{}) error
68           }
69           r.ServiceMethod = c.req.Method
70
71           // JSON request id can be any JSON value;
```

Clicking the link takes you to the entity's definition.

```
34  // Decode reads the next JSON-encoded value from its
35  // input and stores it in the value pointed to by v.
36  //
37  // See the documentation for Unmarshal for details about
38  // the conversion of JSON into a Go value.
39  func (dec *Decoder) Decode(v interface{}) error {
40          if dec.err != nil {
41                  return dec.err
42          }
```

Type information: size/alignment, method set, interfaces

Clicking on the identifier that defines a named type causes a panel to appear, displaying information about the named type, including its size and alignment in bytes, its method set, and its *implements* relation: the set of types T that are assignable to or from this type U where at least one of T or U is an interface. This example shows information about `net/rpc.methodType`.

```
148
149  type methodType struct {
150          sync.Mutex // protects counters
151          method       type info for methodType d
152          ArgType      reflect.Type
153          ReplyType    reflect.Type
154          numCalls     uint
155  }
156
```

Type `methodType`:   (size=136, align=8)
`*methodType` implements `sync.Locker`
method (*methodType) Lock()
method (*methodType) NumCalls() (n uint)
method (*methodType) Unlock()

The method set includes not only the declared methods of the type, but also any methods "promoted" from anonymous fields of structs, such as `sync.Mutex` in this example. In addition, the receiver type is displayed as `*T` or `T` depending on whether it requires the address or just a copy of the receiver value.

The method set and *implements* relation are also available via the package view.

type Server

```
type Server struct {
    // contains filtered or unexported fields
}
```

Server represents an RPC Server.

▼ Implements

> *Server implements net/http.Handler

▼ Method set

```
method (*Server) Accept(lis net.Listener)
method (*Server) HandleHTTP(rpcPath string, debugPath string)
method (*Server) Register(rcvr interface{}) error
method (*Server) RegisterName(name string, rcvr interface{}) error
method (*Server) ServeCodec(codec ServerCodec)
method (*Server) ServeConn(conn io.ReadWriteCloser)
method (*Server) ServeHTTP(w net/http.ResponseWriter, req *net/http.Request)
method (*Server) ServeRequest(codec ServerCodec) error
method (*Server) freeRequest(req *Request)
method (*Server) freeResponse(resp *Response)
method (*Server) getRequest() *Request
method (*Server) getResponse() *Response
method (*Server) readRequest(codec ServerCodec) (service *service, mtype *methodTy
method (*Server) readRequestHeader(codec ServerCodec) (service *service, mtype *me
method (*Server) register(rcvr interface{}, name string, useName bool) error
method (*Server) sendResponse(sending *sync.Mutex, req *Request, reply interface{}
```

Go API DOCS

Why API doc is needed?

To aware the functionality of the API to new developers.

Which tool we needed?

We will use swag tool.

Installation of swag tool

Go to the main directory of the project where your rest api based project is there.

Go to the terminal and fire below commands

```
go get -u github.com/swaggo/swag/cmd/swag
```

```
go get -u github.com/swaggo/http-swagger
go get -u github.com/alecthomas/template
```

This 3 commands will do the necessory installation of swag, http and templates

Routes

I have my app's endpoints as follows:

```
user := r.Group("/user")
```

```
{
```

```
 user.GET("/", controller.GetUsers)
```

```
 user.POST("/", controller.CreateUser)
```

```
 user.GET("/:id", controller.GetUserByID)
```

```
}
```

I will be documenting these endpoints in this artcile.

Models

I have a User model as:

```
type User struct {

  ID          BinaryUUID      `json:"id"`

  Name          string          `json:"name"`

  Email          string          `json:"email"`

  Phone          string          `json:"phone"`

  Address      string          `json:"address"`

  UN          sql.NullString `json:"user_num"
swaggertype:"string"`

}
```

Above, I have `ID` and `UN` fields of **customized data types**. Swag

supports customized data type. In case of field `ID`, the marshallings

and unmarshallings are written in `binary_uuid.go` file [check the example repo in GitHub]. Since, the data type `sql.NullString` is imported from `"database/sql"`, the corresponding field i.e. `UN` requires `swaggertype` tag so that `Swag` can support these kinds of data type.

Handlers

I have three handlers for three endpoints as follows:

```
// GetUsers ... Get all users
```

```
func GetUsers(c *gin.Context) {
```

```
var user []model.User
```

```
err := model.GetAllUsers(&user)
```

```
if err != nil {
```

```
 c.JSON(http.StatusNotFound, gin.H{"error": err.Error()})
```

```go
    return

}

c.JSON(http.StatusOK, gin.H{"data": user})

}

// CreateUser ... Create User

func CreateUser(c *gin.Context) {

var user model.User

if err := c.BindJSON(&user); err != nil {

    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
```

```go
	return


}




err := model.CreateUser(&user)




if err != nil {




	c.JSON(http.StatusInternalServerError, gin.H{"error":
err.Error()})




	return




}




c.JSON(http.StatusOK, gin.H{"message": "success"})
```

```go
}


// GetUserByID ... Get the user by id


func GetUserByID(c *gin.Context) {


id := c.Params.ByName("id")


userID, err := model.StringToBinaryUUID(id)


if err != nil {


 c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})


 return
```

```go
}

var user model.User

err = model.GetUserByID(&user, userID)

if err != nil {

    c.JSON(http.StatusNotFound, gin.H{"error": err.Error()})

    return

}

c.JSON(http.StatusOK, gin.H{"data": user})
```

```
}
```

Integrate Swag to App

General API info

To integrate Swag into the App, we just need to write some

annotations/ comments/docstring or whatever you want to call it. It's

really just bunch of comments before specific API function, which is

used to generate the *Swagger* docs.

Before we get to describing individual API endpoints, we need to first

write general description for our whole project. This part of

annotations lives in the `main` package, right before the `main` function:

```go
package main
        ...
// @title User API documentation
// @version 1.0.0
// @host localhost:5000
// @BasePath /user
func main() {
        ....
}
```

titile: Document title

version: Version

description, **termsOfService**, **contact** ... These are some statements, so don't write them.

host, **BasePath:** If you want to directly swagger to debug the API, these two items need to be filled in correctly. The former is the port of the service document, ip. The latter is the base path, like mine is "/user". BasePath is also not required.

In the original document there `issecurityDefinitions.basic`, `securityDefinitions.apikey`. These are all used for authentication.

API Operation annotations

Now that we have added project-level documentation, let's add documentation to each individual API.

```go
// GetUsers ... Get all users
// @Summary Get all users
// @Description get all users
// @Tags Users
// @Success 200 {array} model.User
// @Failure 404 {object} object
// @Router / [get]
func GetUsers(c *gin.Context) {
    ...
}
// CreateUser ... Create User
// @Summary Create new user based on paramters
// @Description Create new user
// @Tags Users
// @Accept json
// @Param user body model.User true "User Data"
// @Success 200 {object} object
// @Failure 400,500 {object} object
// @Router / [post]
func CreateUser(c *gin.Context) {
    ...
}
// GetUserByID ... Get the user by id
// @Summary Get one user
// @Description get user by ID
// @Tags Users
// @Param id path string true "User ID"
// @Success 200 {object} model.User
// @Failure 400,404 {object} object
// @Router /{id} [get]
func GetUserByID(c *gin.Context) {
    ...
}
```

These comments will appear in the corresponding position of the API document. Here we mainly talk about the following parameters in detail:

Tags

Tags are used to group APIs.

Accept

The received parameter type, support form (`mpfd`) , JSON(`json`), etc., more in the table below.

Produce

The returned data structure is generally`json`, Other support is as follows:

Param

The parameters, from front to back are:

```
// @Param name body string true "Username" default(user)
```

```
// @Param email formData string true "Email"
```

```
@Param 1.Parameter name 2.Parameter type 3.Parameter data type
4.Required 5.Parameter description 6.Other attributes
```

Success

Specify the data for a successful response. The format is:

```
// @Success 1.HTTP response code {2.Response parameter type}
3.Response data type 4.Other description
```

Failure

Same as Success.

Router

Specify routing and HTTP method. The format is:

```
// @Router /path/to/handle [HTTP method]
```

No need to include a basic path.

Generate

Finally, it's time to generate the docs! All you need is one command —

```
swag init
```

This command needs to be ran from directory where `main` is. This command will create package called `docs`, which includes both *JSON* and *YAML* version of our docs.

If you need to update your API annotation or add more endpoints, all you need to is go for the command `swag init` . No need to delete or work on previous docs package. Everything will be updated by Swag itself.

We should see a similar output, if you are curious, you can navigate
to `docs` Catalog and view `swagger.json` file.

Swagger UI

This step is very simple. All we do here is import `httpSwagger` Library,
and the huge documentation we generated. And remember, the import
might be change as per your project requirements and package
installations.

```
import (
 _ "Cyantosh0/go-swag/docs"
 ginSwagger "github.com/swaggo/gin-swagger"
 "github.com/swaggo/gin-swagger/swaggerFiles"
)
```

In addition to specifying routes for all APIs, we must also define a
main route to use Swagger UI to serve the `PathPrefix` method.

```
r := route.SetupRouter()
r.GET("/swagger/*any", ginSwagger.WrapHandler(swaggerFiles.Handler))
r.Run(":" + os.Getenv("SERVER_PORT"))
```

Finally, once we are done with all the APIs, and it's time take them for a spin. To run the app, navigate to your project directory, and run the following commands:

```
go run main.go
```

You can see your work coming to life by loading the swagger UI at http://localhost:5000/swagger/index.html [Here, my app is running in

PORT 5000]

If everything goes well, we should be seeing a UI like below:

Here, we can check our API endpoints.

AWS Cloud with golang
Lambda
It has only virtual functions
It is limited by the times, so short execution
Run on demand
Scaling is automated
Benefits of lambda
Easy price:- Pay per request and per compute time
Free tier of 1000,000 lambda request, 400,000 GBs compute time
It is integrated with whole AWS suite of services
Event Driven Functions get evoked when needed

Integrated with many programming langauges
Easy monitor through cloud watch
Easy to get more resources per functions (upto 10 GB of RAM)

Practical use of the Lambda
Thumbnail creation,
New image in S3 -->>Trigger lambda -->>New thumbnail in S3-->>Metadata in
dynamodv with image size, namedate etc

Things need to remember while using the lambda,
Memory size -->>128 MB to 1040 MB
Timeout -->> 15 Minutes

Hands on with lambda,
There are 2 method by which we can add code to lambda functions.
How you add a program/function in lamda,
Build and compile the code for linux machine
Zip it and upload into source code section
Note that if used python or similar langauges then it must be directly compilabale on
AWS portal. But golang will not supported to compile direct on the portal. So we need to
make machine executable binaries and then need to send to the lambda.

To test the lambda hello word code use the AWS portal Link
1) Add a codec in VS compile binary and upload zip to AWS portal
To check the information go to cloud watch and you can see the logs for the same.

You can make a trigger from the following items,
Alexa IoT
AWS iot
apache kafka
Dynamo DB
S3
SNS and many more

```go
package main

import (
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
)
type Request struct {
    ID     float64 `json:"id`
    Value string  `json:"value`
```

```
}
type Responce struct {
    Message string `json:"message"`
    ok      bool   `json:"ok"`
}
func Handler(request Request) (Responce, error) {
    return Responce{
        Message: fmt.Sprintf("Process request ID: %f", request.ID),
        ok:       true,
    }, nil
}
func main() {
    lambda.Start(Handler)
}

/*
GOOS=linux go build -o main    // This will generate the main executable for linux and push it to source
code of lambda on AWS console portal
// Can test the respince on AWS console portal

*/
```

Amazon EC2
Contanirization
There is virtusl server in cloud
Limited by RAM and CPU
Continuosly running
You have to pay even if there is no load

Go Concurrency
In a normal situation code is execute line by line, one line at a one time.
Concurrency allows multiple lines to be execute.
There are 2 types of concurrent code
1) Threaded :- The code will runs in parallel based on number of cpu cores.
2) Asynchronous :- Code can be pause and resume executions (While pausing other
code can be executed)
Go will choose automatically appropriate method for both above mentioned.
Synchronisation is needed while working with concurrency.
Go routin and channels are a lightweight built in features for managing concurrency and
communication between several function executing at a same time.
This way once can execute the code that outside if the main program.
Go has below keywords likeoncurre

go

chan

Types of channels

1) Buffered :- Can send a data upto the capacity even without a reader.

2) Unbuffered :- Unbuffered channels will be block when sending untill reader is available.

Fact about the channel :-

1) You can convert **bidirectional channel to unidirectional** but viceversa not possible

2) **Use of unidirectional channel :-**   The unidirectional channel can be used to provide the type safety of the program so that program gives less error.

or

unidirectional channel can be used only when you want to create the channel that can send or receive the data.

3) **Zero valued channel :-** The zero value of channel is nil

4) **Blocking send and receive :-** When data is sent to the channel the control is blocked in the send statement untill other goroutine reads from that channel.
Similarly when channel receive the data from go routine the read statement block until another go routine statement.

5) **Length of the channel :-** Length indicates the number of value queued in channel buffer.

```go
// Go program to illustrate how to
// find the length of the channel

package main
import "fmt"
// Main function
func main() {
    // Creating a channel
    // Using make() function
    mychnl := make(chan string, 6)
    mychnl <- "GFG"
    mychnl <- "gfg"
    mychnl <- "Geeks"
    mychnl <- "GeeksforGeeks"
    // Finding the length of the channel
    // Using len() function
    fmt.Println("Length of the channel is: ", len(mychnl))
}

/*
Output
go run channelsize.go
Length of the channel is:  4
*/
```

6) **Capacity of the channel :** Capacity will be size of the buffer.

```go
// Go program to illustrate how to
// find the length of the channel
```

```
package main
import "fmt"
// Main function
func main() {
    // Creating a channel
    // Using make() function
    mychnl := make(chan string, 6)  // Capacity will be 6 here
    mychnl <- "GFG"
    mychnl <- "gfg"
    mychnl <- "Geeks"
    mychnl <- "GeeksforGeeks"
    // Finding the length of the channel
    // Using len() function
    fmt.Println("Capacity of the channel is: ", cap(mychnl))
}
/*
Output
go run channelcapacity.go
Capacity of the channel is:  6
*/
```

FIFO order:- Messages on the channel are FIFO order always
Unbuffered channel

```
package main

import "fmt"
func sendfromchannel(ch chan int) {
    ch <- 47
}
func main() {
    ch := make(chan int)
    go sendfromchannel(ch)
    fmt.Println(<-ch)
}
/*
go run channel.go
42
*/
```

Select statement in channel :-
When need to Send/Receive the data from multiple channels then it will be really helpful.
Select statement is just like a switch statement without the input parameter.
The select statement is used in the channel to perform a single operation out of the
multiple operation provided by the case block.

```
package main    // This is not a perfect example need to check more on this

import (
    "fmt"
    "time"
)
// Main function
```

```go
func main() {
    one := make(chan int)
    two := make(chan int)
    for {
        select {
        case o := <-one:
            fmt.Println("One", o)
        case t := <-two:
            fmt.Println("two", t)
        default:
            fmt.Println("No data to receive")
            time.Sleep(50 * time.Millisecond)
        }
    }
}
```

Write a program that will continuously take the data from the channel and print from 2 different functions?

```go
package main // This is not a perfect example need to check more on this

import (
    "fmt"
    "time"
)
func server1(channel1 chan string) {
    for {
        time.Sleep(3 * time.Second)
        channel1 <- "Echo from server1"
    }
}
func server2(channel2 chan string) {
    for {
        time.Sleep(3 * time.Second)
        channel2 <- "Echo from server2"
    }
}
// Main function
func main() {
    channel1 := make(chan string)
    channel2 := make(chan string)
    go server1(channel1)
    go server2(channel2)
    for {
        select {
        case o := <-channel1:
            fmt.Println(o)
        case t := <-channel2:
            fmt.Println(t)
        default:
            // fmt.Println("No data to receive")
            // time.Sleep(3 * time.Second)
        }
    }
}
```

```
/*



*/
go run selectstatement.go
Echo from server2
Echo from server1
Echo from server2
Echo from server1
…·… and so on
/*
```

Size of the channel
Directional and unidirectional channel

Concurrency with go routine

```go
package main

import (
    "fmt"
    "time"
)

func timesThree(number int) {

    fmt.Println(number * 3)

}

func main() {
    fmt.Println("We are executing a go routine")
    go timesThree(3)
    fmt.Println("Done!")
    time.Sleep(time.Second)


}
```

PS F:\Training\Golang\Program> go run concurrency.go

We are executing a go routine
Done!
9

We have successfully run the concurrency execution
Main program will creates go routine for executing timesThree function
There for fmt.Println("Done!") will executes before go routine

But, what if we need some value returning from that function to continue with our main
function.
Thats where channel comes and save the day.

```go
package main

import (
    "fmt"
)

func timesThree(number int, ch chan int) {
    result := number * 3
    fmt.Println(number * 3)
    ch <- result

}

func main() {
    fmt.Println("We are executing a goroutine")
    ch := make(chan int)
    go timesThree(3, ch)
    result := <-ch
    fmt.Printf("The result is: %v", result)

}
```

We are executing a goroutine
9
The result is: 9

Write a program to use unbuffered channel?

```go
package main
```

```go
import (
    "fmt"
    "time"
)
func listentochannel(ch chan int) {
    for {
        i := <-ch
        fmt.Println(i)
        time.Sleep(1 * time.Second)
    }
}
func main() {
    ch := make(chan int)
    go listentochannel(ch)
    for i := 0; i <= 10; i++ {
        ch <- i
    }
    close(ch)
}
/*
go run channelunbuffered.go
0
1
2
3
4
5
6
7
8
9
10
*/
```

Write a program for buffered channel?

```go
package main

import (
    "fmt"
    "time"
)
func listentochannel(ch chan int) {
    for {
        i := <-ch
        fmt.Println(i)
        time.Sleep(1 * time.Second)
    }
}
func main() {
    ch := make(chan int, 5)
    go listentochannel(ch)
    for i := 0; i <= 10; i++ {
        ch <- i
    }
    close(ch)
}
```

```
/*
go run channelbuffered.go
0
1
2
3
4
5
*/
```

Once the main program executes the goroutines, it waits for the channel to get some data before continuing, therefore fmt.Println("The result is: %v", result) is executed after the goroutine returns the result. This doesn't mean that the main program will wait for the full goroutine to execute, just until the data is served to the channel.
ONCE function in concurrency
sync.Once is a type in the Go standard library that is used to ensure that a piece of code is executed only once, even if multiple goroutines are trying to execute it concurrently.

Syntx is
var once sync.Once

once.Do(function name)

```
package main

import (
    "fmt"
    "sync"
)
var globalVar int
var once sync.Once
func initialize() {
    globalVar = 42
}
func main() {
    go once.Do(initialize)
    go once.Do(initialize)
    once.Do(initialize)
    fmt.Println(globalVar) // prints 42
}
/*
Here we initialise variable 2 times but it will do only one time
go run onceinitialisevariable.go
42
*/
```

Write a program to use once function
or
call function multiple time and show the use of once function in go.

```go
package main

import (
    "fmt"
    "sync"
)
func setup() {
    fmt.Println("Initializing shared resource...")
}
func main() {
    var once sync.Once
    once.Do(setup)
    once.Do(setup)
    once.Do(setup)
}
/*
go run onceprint.go
Initializing shared resource...
*/
```

The role of this function is just call your code once only.
One of the most design pattern is the singletone design pattern.
Package sync provides basic synchronization primitives such as mutual exclusion locks.
Other than the Once and WaitGroup types, most are intended for use by low-level
library routines. Higher-level synchronization is better done via channels and
communication.

```go
package main

import (
    "fmt"
    "sync"
)
type DbConnection struct {}
var (
    dbConnOnce sync.Once
    conn *DbConnection
)
func GetConnection() *DbConnection {
    dbConnOnce.Do(func() {
        conn = &DbConnection{}
        fmt.Println("Inside")
    })
    fmt.Println("Outside")
    return conn
}
```

```go
func main()  {
    for i := 0; i<5; i++ {
        _ = GetConnection()
    }
}
/*
go run once.go
Inside
Outside
Outside
Outside
Outside
Outside
*/
```

Singletone design pattern can also be implemented by using the mutex lock as below.

```go
package main

import (
    "fmt"
    "sync"
)
type DbConnection struct{}
var (
    mut   sync.Mutex
    conn *DbConnection
)
func GetConnection() *DbConnection {
    // Lock and unlock the entire GetInstance function
    mut.Lock()
    defer mut.Unlock()
    if conn == nil {
        fmt.Println("Inside")
        conn = &DbConnection{}
    }
    fmt.Println("Outside")
    return conn
}
func main() {
    for i := 0; i < 5; i++ {
        _ = GetConnection()
    }
}
/*
go run oncewithmutex.go
Inside
Outside
Outside
Outside
Outside
Outside
*/
```

Resource pool with concurrency

Resource pool is a pattern used in concurrent programming.
These will help to manage pool of resources that are shared among the multiple consumers.
The purpose of resource pool is to improve the performance and scalability by allowing the large number of resources to be shared with ,multiple consumers.
- Creating the fixed number or a pool of things for use.
- It is commonly used to constraint things that are expensive.

Resource pool contains number of resources such as databse connections, sockets, file handlers etc
These resources are created and initilised in advance, and are then available for consumer for example worker go routine and thread.

```go
package main

import (
    "fmt"
    "sync"
)
```

```go
type Resource struct {
    ID int
}
```

```go
type ResourcePool struct {
    mu        sync.Mutex
    resources chan *Resource
}
```

```go
func NewResourcePool(numResources int) *ResourcePool {
    rp := &ResourcePool{
        resources: make(chan *Resource, numResources),
    }
    for i := 0; i < numResources; i++ {
        rp.resources <- &Resource{ID: i}
    }
    return rp
}
func (rp *ResourcePool) Get() *Resource {
    return <-rp.resources
}
func (rp *ResourcePool) Put(r *Resource) {
    rp.resources <- r
}
func main() {
    rp := NewResourcePool(5)
    res := rp.Get()
    fmt.Println("Got resource:", res.ID)
    rp.Put(res)
    fmt.Println("Returned resource:", res.ID)
}
/*
```

```
go run resourcepoolwithchannel.go
Got resource: 0
Returned resource: 0
*/
```

Note that in this example I've used a simple channel and a mutex, but in real life it's better to use more advance libraries like pool package in golang to achieve the same purpose.

Worker pool with concurrency
Golang supports worker pool as a pattern.
Worker pool is designed in which fixed number of m workers (Go routines) works on n task in work queue (Go channel).
The work resides in queue untill a worker finish its current task and pull a new one.
How worker pool works
1) Creating the channel to hold the task to be processed
2) Creating a number of go routine to pull a task from the channel and process them
3) Sending the tasks to the channels to be processed by worker go routine.

Application of the worker pool
- **Data processing:** When you have a large dataset that needs to be processed, you can use a worker pool to divide the dataset into smaller chunks and process them concurrently.
- **Network requests:** When you need to make a large number of network requests, you can use a worker pool to handle the requests concurrently and avoid overloading the network.
- **Image processing:** When you need to process a large number of images, you can use a worker pool to divide the images into smaller chunks and process them concurrently.
- **File processing:** When you need to process large number of files, you can use a worker pool to divide the files into smaller chunks and process them concurrently.
- **Parallel computing:** When you have a complex algorithm that can be broken down into smaller tasks and performed concurrently.

```go
package main

import (
    "fmt"
    "time"
)
func worker(id int, jobs <-chan int, results chan<- int) {
    for j := range jobs {
        fmt.Println("worker", id, "processing job", j)
        time.Sleep(time.Second)
        results <- j * 2
```

```go
        }
}
func main() {
    job := make(chan int, 10)
    result := make(chan int, 10)
    for w := 1; w <= 2; w++ {
        go worker(w, job, result)
    }
    for j := 1; j <= 9; j++ {
        job <- j
    }
    close(job)
    for a := 1; a <= 9; a++ {
        <-result
    }
}
/*
go run workerpool.go
worker 2 processing job 1
worker 1 processing job 2
worker 1 processing job 3
worker 2 processing job 4
worker 2 processing job 5
worker 1 processing job 6
worker 1 processing job 7
worker 2 processing job 8
worker 2 processing job 9
*/
```

Write a program to perform 10 tasks use 2 workers. Use waitgroup, 2 channel to perform the task.

```go
package main

import (
    "fmt"
    "sync"
)
func worker(id int, jobs <-chan int, results chan<- int, wg *sync.WaitGroup) {
    defer wg.Done()
    for j := range jobs {
        fmt.Println("worker", id, "processing job", j)
        //time.Sleep(time.Second)  this will run a job sequnceially since we use the 1 second time to
cmplete the go routine
        results <- j * 2
    }
}
func main() {
    job := make(chan int, 10)
    result := make(chan int, 10)
    var wg sync.WaitGroup
    for w := 1; w <= 2; w++ {
        wg.Add(1)
        go worker(w, job, result, &wg)
    }
    for j := 1; j <= 9; j++ {
```

```
        job <- j
    }
    close(job)
    for a := 1; a <= 9; a++ {
        <-result
    }
    wg.Wait()
    fmt.Println("All worker have been processed")
}
/*
go run workerpoolwaitgroup.go
worker 2 processing job 1
worker 2 processing job 3
worker 2 processing job 4
worker 2 processing job 5
worker 2 processing job 6
worker 2 processing job 7
worker 2 processing job 8
worker 2 processing job 9
worker 1 processing job 2
All worker have been processed
*/
```

Write a program that use single channel, waitgroup, 3 worker complete the 5 tasks using worker pool.

```go
package main

import (
    "fmt"
    "sync"
)
func worker(id int, tasks <-chan int, wg *sync.WaitGroup) {
    defer wg.Done()
    for task := range tasks {
        fmt.Printf("Worker %d processing task %d\n", id, task)
    }
}
func main() {
    tasks := make(chan int)
    var wg sync.WaitGroup
    // Create a fixed number of worker goroutines
    for i := 1; i <= 3; i++ {
        wg.Add(1)
        go worker(i, tasks, &wg)
    }
    // Send tasks to the task channel
    for i := 1; i <= 5; i++ {
        tasks <- i
    }
    close(tasks)
```

```
    // Wait for all the worker goroutines to finish
    wg.Wait()
    fmt.Println("All tasks have been processed")
}
/*
 go run workerpoolwaitgroup3worker5tasksinglechannel.go
Worker 3 processing task 1
Worker 3 processing task 4
Worker 3 processing task 5
Worker 2 processing task 3
Worker 1 processing task 2
All tasks have been processed
*/
```

Atomics in golang
- Golang supports special fuction called atomic.
- Used to synchronise the memory while working with the go routine.
- The purpose of this function is to reduce the use of samer memory location when we called multiple go routines.
- Package atomic provides low-level atomic memory primitives useful for implementing synchronization algorithms.
- These functions require great care to be used correctly. Except for special, low-level applications, synchronization is better done with channels or the facilities of the sync package. Share memory by communicating; don't communicate by sharing memory.

It will not work with int it will needed uint32/uint64/int64/int32/uintptr/type value/pointer/bool

Basic operation for which it can be used is
1) Add
2) Compare
3) load
4) store
5) swap

Use of atomic operations in Golang
Atomic operations are used when we need to have a shared variable between different goroutines which will be updated by them. If the updating operation is not synchronized then it will create a problem that we saw.

Wrtie a program to increment counte 1000 times using the atomic
or
Write a program that will call 1000 go routine with atomic sync package.

```
package main

import (
```

```
    "fmt"
    "sync"
    "sync/atomic"
)
func main() {
    var counter uint64
    var wg sync.WaitGroup
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go func() {
            atomic.AddUint64(&counter, 1)
            wg.Done()
        }()
    }
    wg.Wait()
    fmt.Println(counter)
}
/*
go run atomic1000goroutine.go
1000
*/
```

There are a few things to keep in mind when using the atomic package in Go:

● The atomic package only provides a limited set of operations, such as atomic increments and exchanges. If you need to perform more complex operations, you may need to use other synchronization mechanisms, such as mutexes.
● The atomic operations are implemented using low-level CPU instructions, which means they are very fast but also potentially platform-dependent. You should be aware of this when writing code that needs to be portable across different architectures.
● The atomic package is intended for use in low-level code, such as in the implementation of higher-level synchronization primitives. It is not designed for use as a general-purpose synchronization mechanism.
● You should be careful to ensure that the variables you are using with the atomic package are properly aligned in memory. On some architectures, unaligned access to atomic variables can cause performance issues or even crashes.
● If you are using the atomic package to synchronize access to a variable from multiple goroutines, you should be aware of the possibility of contention. Contention can occur when multiple goroutines try to access the same atomic variable simultaneously, which can lead to performance issues.

Ways to wait the multiple go routines?
or
How many ways are there to wait the multiple go routine?
or
how to sync the multiple go routine?

1) Wait group
2) sync.once   // Not sure how to do it with this function
3) Workers pool

Developing a webservices
- Download source code from https://github.com/tsawler/bookings-udemy/releases/tag/v21
- Unzipp into you workspace
- Go to visual studio
- File>openfolder
- Go to main directory
- Run the command go run main.go
- It will run the application on your browser

Open your browser and type URL
http://localhost:8080/

And you got a Hello World! on the browser

Database operation
Download postgresql
https://www.enterprisedb.com/postgresql-tutorial-resources-training?uuid=db55e32d-e9f0-4d7c-9aef-b17d01210704&campaignId=7012J000001NhszQAC

Download DBeaver
https://dbeaver.io/download/
Create a database called test in dbviewer

Code is as below & taken from

https://att-c.udemycdn.com/2021-04-05_23-12-41-ccb5b133039198cd672e083dafee1c72/original.zip?response-content-disposition=attachment%3B+filename%3Dtest_connect.zip&Expires=1652886735&Signature=P-ADFLUqNG4i6xJIUwB2ukNITzHjJ3TTroisK2V7MOSsiKeC7eC7FxMw3dVCUEfeaMKi454dR~d~m~naWyZpuvFibWMU84GutwSxxIxpoDOg~EWY8lp~1l5ng6fJTHGKhu0kmKGdn8DJJJBDnNAlC5lfoGvVtsbg9VLnoHMPL24~56EEPjkqnNFteKOtm-GvSVaQz7lQZPwJQVnmCDRzv1oe0VaAozR4iKD1JQWgXYRER20ERT~50-5PycL73A~bj2rVh9qMVaOBqaFU1vLwaLTGguDm4LVV-jXSDI0blymO7mtRy4nW5QLOFC8gblmHhoOlL~UObnxe8o20uHtLhg__&Key-Pair-Id=APKAITJV77WS5ZT7262A

```go
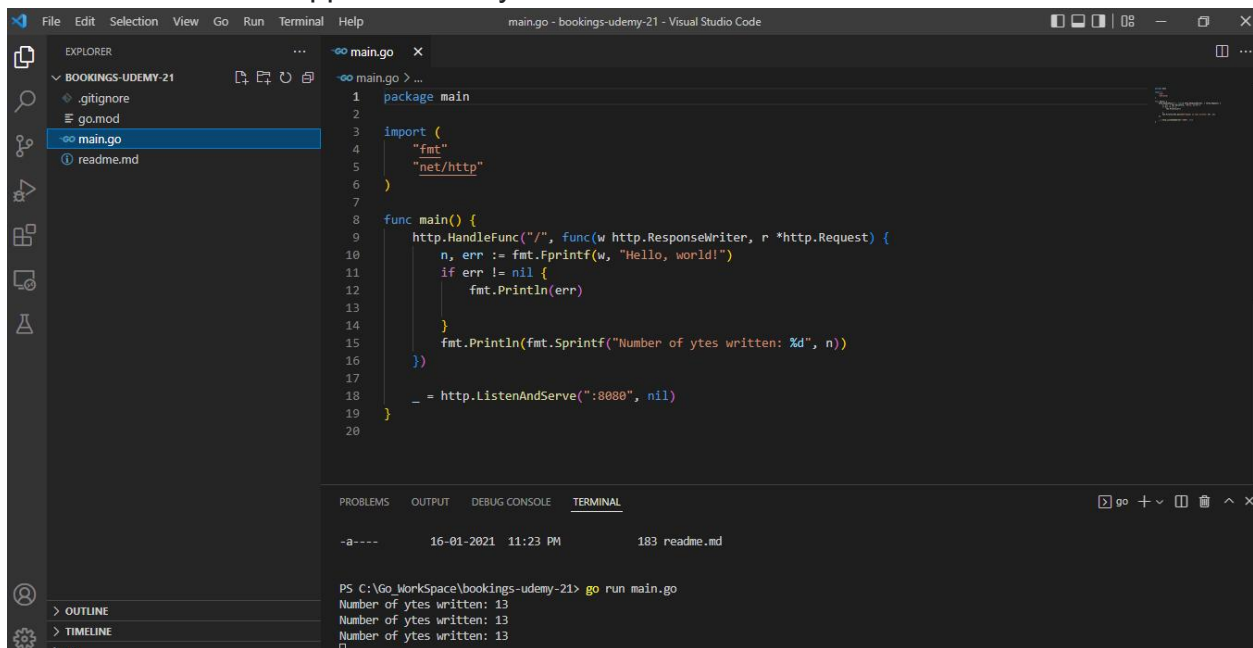package main

import (
    "database/sql"
    "fmt"
    "log"

    _ "github.com/jackc/pgx/v4/stdlib"
)

func main() {
    // connect to a database
    conn, err := sql.Open("pgx", "host=localhost port=5432 dbname=test
user=yagnik.pokal password=yagnik@2017")
    if err != nil {
        log.Fatal(fmt.Sprintf("Unable to connect: %v\n", err))
    }
    defer conn.Close()

    log.Println("Connected to database!")

    // test my connection
    err = conn.Ping()
    if err != nil {
        log.Fatal("Cannot ping database!")
    }

    log.Println("Pinged database!")

    // get rows from table
    err = getAllRows(conn)
    if err != nil {
        log.Fatal(err)
    }

    // insert a row
    query := `insert into users (first_name, last_name) values ($1, $2)`
    _, err = conn.Exec(query, "Jack", "Brown")
    if err != nil {
        log.Fatal(err)
```

```go
    }

    log.Println("Inserted a row!")

    // get rows from table again
    err = getAllRows(conn)
    if err != nil {
        log.Fatal(err)
    }

    // update a row
    stmt := `update users set first_name = $1 where id = $2`
    _, err = conn.Exec(stmt, "Jackie", 5)
    if err != nil {
        log.Fatal(err)
    }

    log.Println("Updated one or more rows")

    // get rows from table again
    err = getAllRows(conn)
    if err != nil {
        log.Fatal(err)
    }

    // get one row by id
    query = `select id, first_name, last_name from users where id = $1`

    var firstName, lastName string
    var id int

    row := conn.QueryRow(query, 1)
    err = row.Scan(&id, &firstName, &lastName)
    if err != nil {
        log.Fatal(err)
    }
    log.Println("QueryRow returns", id, firstName, lastName)

    // delete a row
    query = `delete from users where id = $1`
```

```go
        _, err = conn.Exec(query, 6)
    if err != nil {
        log.Fatal(err)
    }


    log.Println("Deleted a row!")

    // get rows from table again
    err = getAllRows(conn)
    if err != nil {
        log.Fatal(err)
    }


}

func getAllRows(conn *sql.DB) error {
    rows, err := conn.Query("select id, first_name, last_name from users")
    if err != nil {
        log.Println(err)
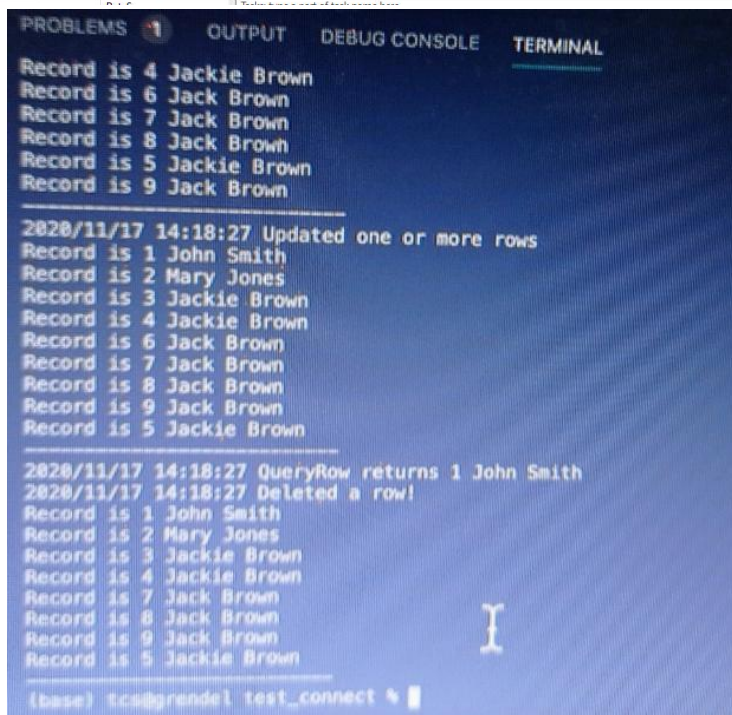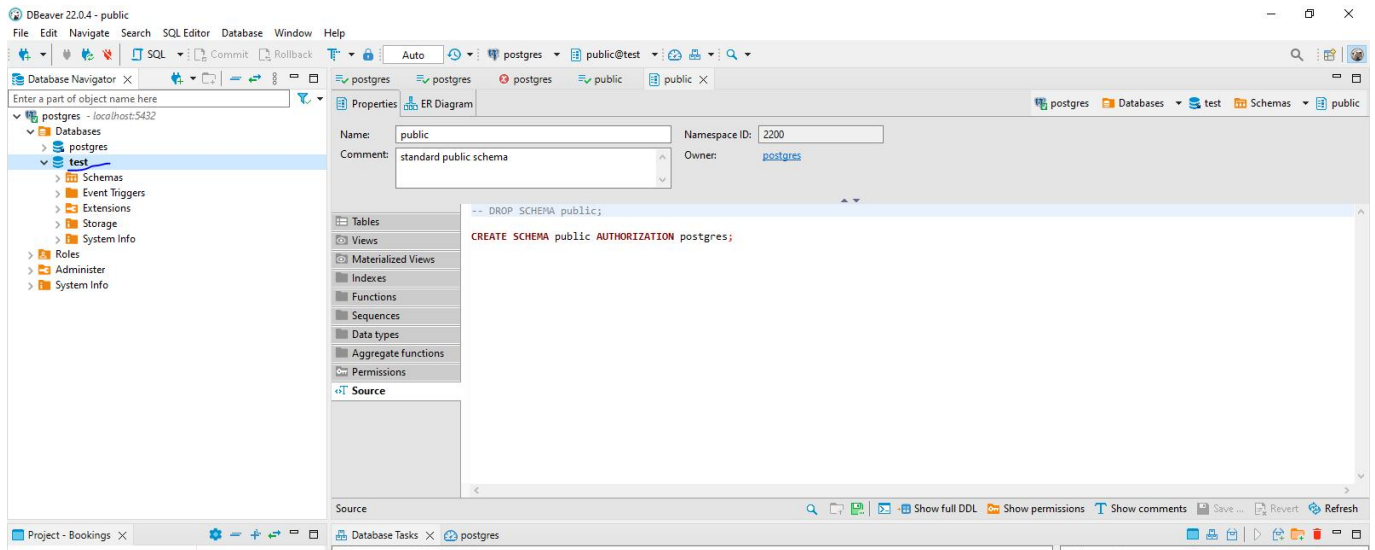        return err
    }
    defer rows.Close()

    var firstName, lastName string
    var id int

    for rows.Next() {
        err := rows.Scan(&id, &firstName, &lastName)
        if err != nil {
            log.Println(err)
            return err
        }
        fmt.Println("Record is", id, firstName, lastName)
    }

    if err = rows.Err(); err != nil {
        log.Fatal("Error scanning rows", err)
    }

    fmt.Println("---------------------------")
```

```
    return nil
}
```


Build webaaplication
Dowbload code from below
https://github.com/yagnikpokal/golang/tree/main/Baseapp

Extract in a particular directory
Import extracted folder in VS
Import necessary packages
Debug and run the application in VS

Application will available on port 8080
Open your browser and type URL
http://localhost:8080/

Check on below link for demonstration
https://youtu.be/bUwsVwwE8ZA

Question and answers
Mention the advantages of golang?
How to declare the multiple type of variable in single line?
What are builtin support in go?
Why does golang developed?
Why should I want to learn the go programming langauge?
Go is functional or OOP?
How to perform testing in golang?
How to compare struct in golang?

Does go have optional parameters?
What is rune in go?
What are function closures?
What are rvalue and lvalue?
What are golang pointer?
What is string literature?
Formate a string without printing?

What do you understand by type assertion in go?
or
What happen if I don't mention concret type T(t := I.(T))?
This is called type assertion. It takes the interface value and retrives specified explicit
data type.
The type assertion is the process to extract the value of interface.
t := I.(T)
I is the interface value
T is conceret type
t is the variable value assign from type T
What happended if interface I doesn't have concrete type T?
The statement will result in panic error.

How to check if interface has concrete type T or not?
or
How to check wether the type assertion is completed or not?
t, isSuccess := I.(T)

t will get underlying value
isSuccess will get true or false
If it is false then it has a type T and value of t =0. So no panic error.

How would you check the type of variable in runtime?
can use %T with fmt.Printf("The type is %T",V)

What is type switch in go?
In a Go interface type switch is used to compare the concrete type of the interface with
multiple type of case statement.
It has one difference then the actual switch which is case has type not a values.

```go
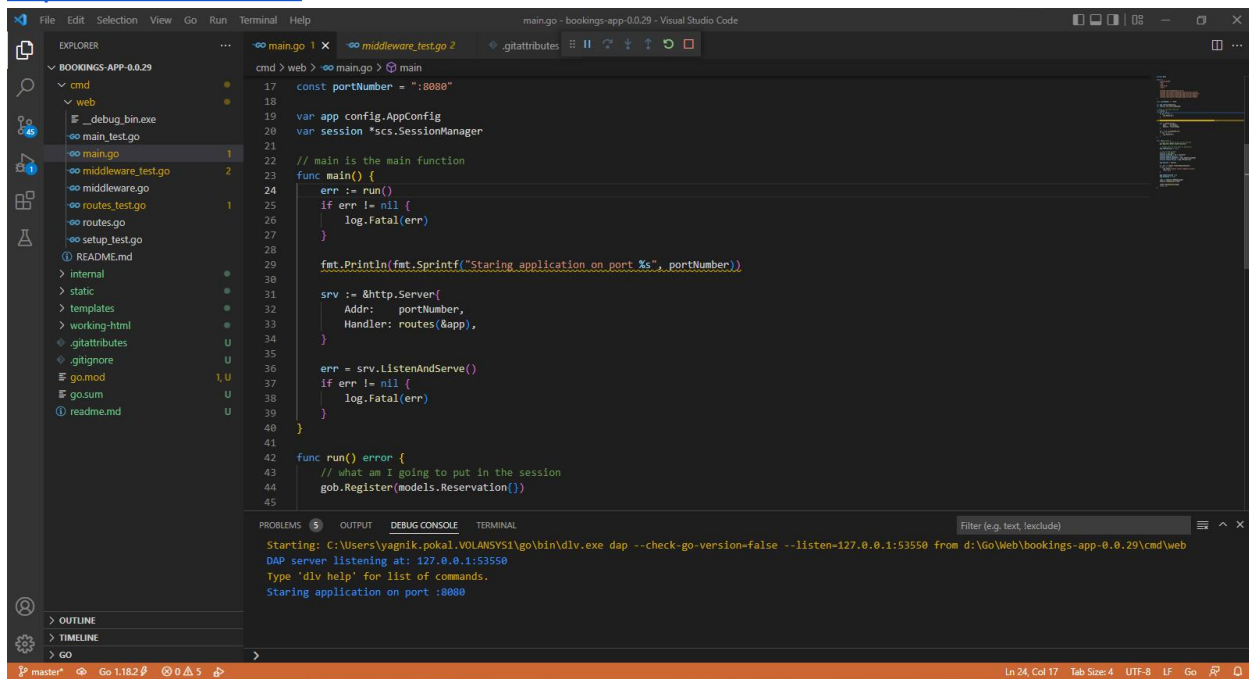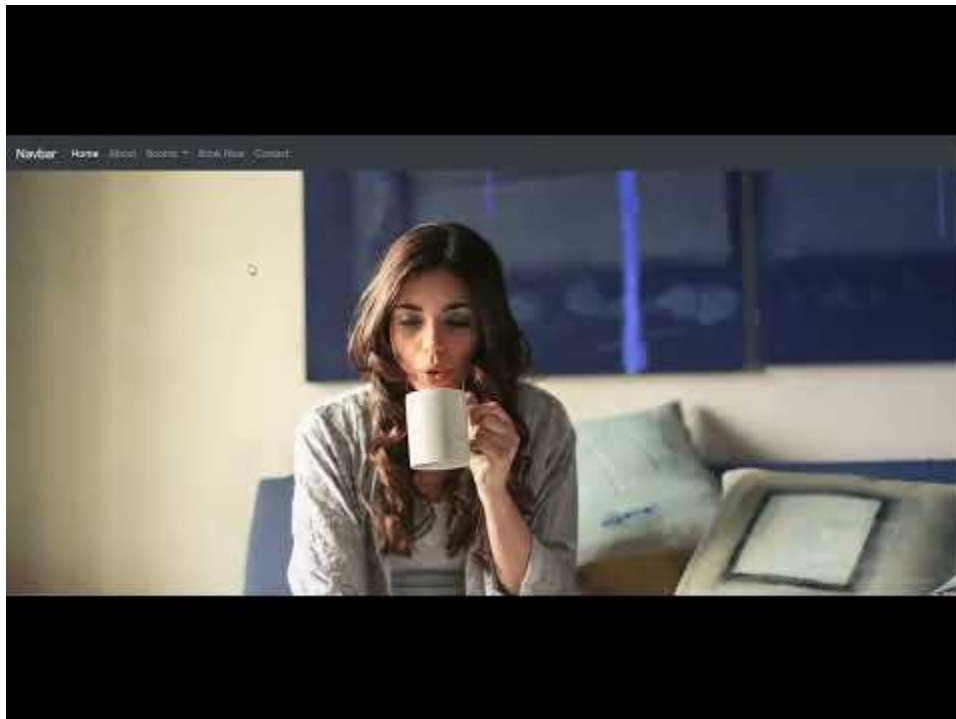// Go program to illustrate type switch
package main

import "fmt"
func myfun(a interface{}) {
    // Using type switch
    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")
    }
}
// Main method
func main() {
    myfun("GeeksforGeeks")
    myfun(67.9)
    myfun(true)
}
/*
go run typeswitchinterface.go

Type: string, Value:  GeeksforGeeks

Type: float64, Value:  67.9
*/
```

What are decision making statement in go?
How to declare if as while in go?
What is GOROOT and GOPATH environment variables in go?

What is structure in go?
Why do we used break statement?
Why do we used continue statement?
WHy do we use Goto statement?
What kind of conversion is supported in go?
What is cGO in golang?

What is grpc?
What is graphql?
Graphql is the query langauge for the api
It provides alternative to the rest
- It is designed to get the client data in flexible way.
- In a graphql api Client specify the data it needs and server returns tht data in shape. This allows the client to request exactly the data it needs, and nothing more
- Which make the api more efficient and reduce the amount of data over the network.

Write a program to sort the array?
What is length and capacity of slice?
Write a program that do a subslice and calculate the length and capacity for the same?
Write a program to parse the JSON?
What is slice?
What is array?
What is iota?
iota will use to assign integer value to the constant while declaring constant by using short hand method

```go
package main

import "fmt"
const (
    north = iota
    south
    east
    west
)
func main() {
    fmt.Println(north, south, east, west)
}
/*
Output
0 1 2 3
*/
```

What is variadic function?
The function that will takes variable number of arguments then it is called as variadics
function

```go
package main

import "fmt"
func sum(a ...int) int {
    sum := 0
    for _, j := range a {
        sum += j
    }
    return sum
}
func main() {
    x := []int{1, 2, 3, 5, 7, 9, 6}
    y := sum(x...)
    fmt.Println(y)
}
/*Output
33     */
```

What is method and how it is different then normal function?
How to check wether the key is present or not in golang?
we can use if statement to check wether key is present or not
if value, isPresent := mymap[key]; isPresent {
// Do something if key is present
}
This will help when you know the key and if you want to see wether it is present or not in
the map.

How to import the package from the folder?
What is init function?
What is new and make function?

How garbage collection works in go?
Using mark and sweep algorithm. It will contiguously check on the program that wether
if there is any unused variable or not and if it find the unused variable which will not
used in future then it will remove value of that particular memory address and and free
up space.
GO will do refreshing of the above activity at a certain time interval and that's how it will
do a garbage collection.
What is channel?
Difference between buffered and unbuffered channel?

Select and switch statement in golang?

What is waitgroup?

What is panic error?

How to handle exception in golang?

When anything unexpected happen then error will comes. that unexpected is known as the exceptions.

With the help of that error developer came to know what to debug and where to start.

Error handling in golang is very easy.

Go functions return errors as a second return value.

```go
package main

import (
    "errors"
    "fmt"
)
func mul(i int, j int) (int, error) {
    if i == 0 || j == 0 {
        return 0, errors.New("0 can not be multipled")
    }
    return i * j, nil
}
func main() {
    d, err := mul(2, 0)
    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println(d)
    }
}

/*
go run errornew.go
0 can not be multipled
*/
```

 Ignoring the error golang

```go
package main

import (
    "errors"
    "fmt"
)
func returnError() (int, error) { // declare return type here
    return 42, errors.New("Error occured!") // return it here
}
func main() {
    v, e := returnError()
    if e != nil {
        fmt.Println(e, v) // Error occured! 42
    }
}
/*
```

```
go run errorwithvalue.go
Error occured! 42
*/
```

Creating a custome error

```go
package main

import (
    "fmt"
    "os"
)
type MyError struct{}
func (m *MyError) Error() string {
    return "boom"
}
func sayHello() (string, error) {
    return "", &MyError{}
}
func main() {
    _, err := sayHello()
    if err != nil {
        fmt.Println(err)
        os.Exit(1)
    }
}

/*
go run errorcustome.go
boom
exit status 1
*/
```

Panic errors
Panic errors are of 2 types
1) Generated by the go compiler/langauge
2) Custom panic
Generated by the compiler mean **compiler** will give these errors runtime. like below

```go
package main

import "fmt"
func main() {
    d := []string{
        "Yagnik",
        "Pokal",
    }
    fmt.Println(d[2])  // We use only 0 and 1 index, 2 index is not preset so it will gove error
}

/*
go run errorpaniccompiler.go
panic: runtime error: index out of range [2] with length 2
```

```
goroutine 1 [running]:
main.main()
        D:/Training/Go/GIT/Go/golang/Learning/Programs/errorpaniccompiler.go:10 +0x1b
exit status 2
*/
```

Custom panic

```go
package main

func foo() {
    panic("There is something wrong")
}
func main() {
    foo()
}



/*
go run errorpanicbuiltinfunction.go
panic: THere is something wrong

goroutine 1 [running]:
main.foo(...)
        D:/Training/Go/GIT/Go/golang/Learning/Programs/errorpanicbuiltinfunction.go:4
main.main()
        D:/Training/Go/GIT/Go/golang/Learning/Programs/errorpanicbuiltinfunction.go:7 +0x27
exit status 2
*/
```

Panic with defer

```go
package main

import "fmt"
func foo() {
    defer panic("There is something wrong!")
    fmt.Println("hello from the deferred function!")
}
func main() {
    foo()
}

/*
go run errorpaniccompiler.go
hello from the deferred function!
panic: There is something wrong!

goroutine 1 [running]:
main.foo.func1()
        D:/Training/Go/GIT/Go/golang/Learning/Programs/errorpaniccompiler.go:7 +0x2a
main.foo()
        D:/Training/Go/GIT/Go/golang/Learning/Programs/errorpaniccompiler.go:9 +0xb5
main.main()
        D:/Training/Go/GIT/Go/golang/Learning/Programs/errorpaniccompiler.go:11 +0x17
```

```
exit status 2
*/
```

Recover panic error
The recover function relies on the value of the error to make determinations as to whether a panic occurred or not.
Panics have a single recovery mechanism—the recover builtin function.

```go
package main

import (
    "fmt"
    "log"
)
func main() {
    divideByZero()
    fmt.Println("we survived dividing by zero!")
}
func divideByZero() {
    defer func() {
        if err := recover(); err != nil {
            log.Println("panic occurred:", err)
        }
    }()
    fmt.Println(divide(1, 0))
}
func divide(a, b int) int {
    return a / b
}

/*
go run errorpanicrecover.go
2022/12/05 12:45:48 panic occurred: runtime error: integer divide by zero
we survived dividing by zero!
*/
```

Referances
https://www.digitalocean.com/community/tutorials/handling-panics-in-go

What is use of the defer keyword?
What is rest api? Advantages of rest api?
Rest is stands for representational state transfer.
It is an architectectureal style that defines defines a set of constraints to be used for creating web services.
REST API is the way of accessing the web services in a simple and flexible way without having any processing.

A request is sent from client(UI Uniform interface) to server in the form of web URL as HTTP GET, POST, PUT, DELETE request.
After that response is comes anything like HTML, XML, Image, JSON
But now a days JSON is the most popular formate of being used in web services.

Advantages of REST
1) Easily scalable
2) Decrese complexity
3) Improoved performance
4) Very commonly used
5) Stateless
6) Resource caching

1) **Easily scalable :** As there is no need to store any information any server can handles any client request.Thus many concurrent requests can be processed by deploying API to multiple server.
2) **Decrease complexity :** As state synchronization is not needed it reduce complexity.
3) **Improved performance :** Server does not need to keep track of client which increase the performance.

4) **Stateless** means every HTTP request happens in the complete isolation.
REST statelessness means being free from the application state.
Application state is the information stored at the server side to identify incoming requests and previous interactions.
5) **Resource cache** All the resouces are should allow caching unless explicitly that cache is not possible.
6) **Layered system** REST allows architecture composed of muliple layer of servers
7) **Code on demand** Most of the time server send JSON, however when necessary server can send executable code to client.

Disadvantages of REST
1) long request time and too much datasize: The request size becomes very big many time as it contains all the information about the request and previous transaction.
2) Security: lots os aspects like
  1) HTTPs

2) blocking IPs from unlnown IP adresses and domains
3) Validating URIs
4) Block of unexpected large payload
5) logging request
6) Investigating failures
3) Authentication
4) Request and data
5) API testing
6) Define error code and messages

What is restful in rest API?
Restful is a services that uses the REST API.

What is CRUD operations?
C create -->> POST
R read -->> GET
U update -->>PU
D delete -->>delete

What is RESTAPI doc?
REST API doc is the document which is given by service services by the serverside. it
has a details like
1) size of the payload
2) maximum hit of API
3) how response will get
4) how authentication will work

References
https://www.techtarget.com/searchapparchitecture/definition/RESTful-API
https://www.youtube.com/watch?v=lsMQRaeKNDk
https://www.youtube.com/watch?v=E0Qqpn8ymko

Where rest stores the data of state?
On the client side

Difference between rest api and graphql and advantages and disadvantages?
What is directional and unidirection channel in golang?

If you use a linux PC and you want to compile a binary for the windows how to do that and viceversa?
Write a program for number increment and return error if there is negative number using panic?

```go
package main

import "fmt"
func sum(a int, b int) int {
    if a <= 0 || b <= 0 {
        panic("The number is negative")  //Panic error
    }
    return a + b
}
func main() {
    D := sum(5, 6)
    fmt.Println(D)
}
```

Write a program for number increment and return error if there is negative number?

```go
package main

import (
    "errors"
    "fmt"
)
var negativenumber = errors.New("The number is negative")
func Sum(a int, b int) (int, error) {
    if a < 0 || b < 0 {
        return 0, negativenumber
    }
    return a + b, nil
}
func main() {
    D, error := Sum(5, 65)
    if error != nil {
        fmt.Println(error)
    } else {
        fmt.Println(D)
    }
}

/*
go run sum.go
Error : The number is negative
*/
```

Write a program that will take slice as a struct and sort the salery of the eployees?
Is there any relationship between buffered, unbuffered and directional, nondirectional channel?
What is environment variable in go?
What is package in go?

In the below code I have added _ while importing the function what is the meaning of that?
import (
_ go/golang/master
)
What is interface in golang?
What is goroutine?
Write a program to use 1000 goroutine and increment the counter and check of there is race condition or not?
Below program is not increment 1000 go routines. SInce there is race condition. DUe to that it will give different values each and every time.
Race condition can be solved by 2 ways
1) Using mutex
2) Using Channels
3) Using the atomic
While langauge given a features of channel that doesn't mean that every time we will use the channel. Based on need we can use mutex and channels.
In general **channel** can be used when go routine needes to be communicate with each other and,
Mutex when only one go routine should be access the critical section of the code.

```go
package main

import (
    "fmt"
    "sync"
)
var counter = 0
func increment(wg *sync.WaitGroup) {
    counter = counter + 1
    wg.Done()
}
func main() {
    var w sync.WaitGroup
    for i := 0; i < 1000; i++ {
        w.Add(1)
        go increment(&w)
    w.Wait()
    fmt.Println(counter)
}
/*
Output
go run 1000goroutine.go
990

// Check the race condition
go run -race 1000goroutine.go
==================
WARNING: DATA RACE
*/
```

Suppose I have a race detection condition in go how to solve it? If it is solvable how to use that with mutex?

or

What is mutex and semaphore in golang?

The name mutex itself a mutual exclusion. That means while accessing single variable by 2 processes we put a mutual lock and at a time only one can be access the value of the variable the second will wait till the end of process.

Write a program that increment the counter with 1000 go routine and use mutex lock?

Or

Solve the race detection problem(1000 go routine) with the mutex lock?

Using the function on fly

```go
package main

import (
    "fmt"
    "sync"
)
func main() {
    var counter uint64
    var wg sync.WaitGroup
    var mu sync.Mutex
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go func() {
            mu.Lock()
            counter++
            mu.Unlock()
            wg.Done()
        }()
    }
    wg.Wait()
    fmt.Println(counter)
}
/*
go run mutex1000goroutine.go
1000
go run -race mutex1000goroutine.go
1000
*/
```

By writing the function

```go
package main

import (
    "fmt"
    "sync"
)
```

```go
var counter = 0
func increment(m *sync.Mutex, wg *sync.WaitGroup) {
    m.Lock()
    counter = counter + 1
    m.Unlock()
    wg.Done()
}
func main() {
    var m sync.Mutex
    var wg sync.WaitGroup
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go increment(&m, &wg)
    }
    wg.Wait()
    fmt.Println(counter)
}
/*
output
mutex1000goroutinefunction.go
1000
// Check race detections in go
go run -race mutex.go
1000
*/
```

Write a program that increment the counter with 1000 go routine and use channel?
Or
Solve the race detection problem(1000 go routine) with the channel?
Using function on fly

```go
package main

import (
    "fmt"
    "sync"
)
var counter = 0
func main() {
    var wg sync.WaitGroup
    mychan := make(chan bool, 1)
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go func() {
            mychan <- true
            counter++
            <-mychan
            wg.Done()
        }()
    }
    wg.Wait()
    fmt.Println(counter)
}
```

```
/*
go run -race channel1000goroutine.go
1000
*/
```

Using the function

```go
package main

import (
    "fmt"
    "sync"
)
var counter = 0
func increment(wg *sync.WaitGroup, mychan chan bool) {
    mychan <- true
    counter = counter + 1
    <-mychan
    wg.Done()
}
func main() {
    var wg sync.WaitGroup
    mychan := make(chan bool, 1)
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go increment(&wg, mychan)
    }
    wg.Wait()
    fmt.Println(counter)
}

/*
Output
go run channel1000goroutinefunction.go
1000

// Check the race detection
go run -race channel1000goroutinefunction.go
1000

*/
```

Referances
https://golangbot.com/mutex/

What is reflection go?
How can we swap variables in go?

```go
package main
```

```
import "fmt"
func swap(a, b int) (int, int) {   //Variables can be stopped by just writing below syntax a, b = b, a
    a, b = b, a
    return a, b
}
func main() {
    x := 5
    y := 7
    c, d := swap(x, y)
    fmt.Println(c, d)
}
```

What is static and dynamic variables?
Write a program that iterate over a string and print all the element of the string?
or
Write a program that iterate over a string and print the number of time repeating elements?
or
write a program to print only repeating character in string?

```
// Click here and start typing.
package main

import (
    "fmt"
    "strings"
)
func main() {
    name := "yagnikpokal"
    var mymap = make(map[string]int)
    for I, j := range name {
        fmt.Println(I, string(j))
        singlecharacter := string(j)
        sumcharacter := strings.Count(name, singlecharacter)
        //fmt.Println(string(j), "repeated", sumcharacter, "times")
        mymap[string(j)] = sumcharacter
    }
    fmt.Println(mymap) // To print all the values
    for key, element := range mymap {
        if element >= 2 {
            fmt.Println(key, "Present", element, "times") // To print only repeating character
        }
    }
}
*/
go run stringprintrepeatingcharacter.go
0 y
1 a
2 g
3 n
4 i
5 k
6 p
```

```
7 o
8 k
9 a
10 l
map[a:2 g:1 i:1 k:2 l:1 n:1 o:1 p:1 y:1]
a Present 2 times
k Present 2 times*/
```

How many ways are there to print the the repeating element in the string?
What is protobuf?
protobuf is the googles language neurals, platform neural, extensible serialising the struct data.
Think like xml data but smaller, faster and simpler. Protocol buffer supports generated code in java, python, objective-c,c++. How ever proto3 version also supports kotline, dart, go, ruby, c#.
Protocol buffer is used to communicate the inter-server communications and archival storage data in servers.

What are the advantages of protobuf?
1. It take input as a struct and data can be transferable in serialized manner forward and back word compatibility.
2. Protocol buffers can be extended with the new informtion without invalidating existing data or requrining the code to be updated.
3. It removes language specific runtime library. It will create the .proto file.

What are disadvantages of protobuf?
4. Supports size of few megabytes so when it is comes to transfer speed of more then few mbps then it will not work.
What are the steps to use the protobuf with go?
● Define the message formates in .proto file
● Use the protobuf compiler to compile the protobuf files
● Use the go protocol buffer API to write and read messages.

What is the use of mongo db?
What is regexp?
Does go supports inheritance?
Go doesn't supports inheritance. However we can use composition, embedding, interfaces to reuse and polymorphism.
Does go is case sensitive?
Yes Go is case sensitive language.

List the operators in golang?
List the datatype?
Scope of variable?
What is golang workspace?
How to return multiple values from function?
Is the usage of the global variable in programs implementing go routines recommnded?
What are the uses of the empty struct?
Empty struct is used when we want to save memory. This is because it does not consume any memory for the value.
What is syntex for the empty struct?

```
package main

import (
    "fmt"
    "unsafe"
)
func main() {
    a := struct{}{}
    fmt.Println(unsafe.Sizeof(a))
}
```

How can we copy slice and map?
In GO are there any good error handling practices?
Which is safer in concurrent data access? channels or maps?
Channel is safe while using concurrent data access since it will provides locking mechanism.
How can you sort a custome struct with the help of an example?
What do you understand by shadowing in go?
What do you understand by rune and byte datatype? How they are represented?
Byte and rune are both integer datatype.
Byte is uint8 and rune is int32
Byte will represents ASCII charcater and rune will represnts unicode character with UTF-8
Rune is also called as codepoint and also can be a numaric value.For example 0x61 in hexadecimal corresponds to rune literature a.

Golang programs
Write a program to swap a variable in a list? or Write a program to swap array or slice?

```
package main

import "fmt"
func swap(sw []int) {
    for a, b := 0, len(sw)-1; a < b; a, b = a+1, b-1 {
        sw[a], sw[b] = sw[b], sw[a]
    }
}
func main() {
```

```
    x := []int{3, 2, 4, 5}
    swap(x)
    fmt.Println(x)
}


// Output
[5 4 2 3]
```

Write a program to swap 2 variables in golang?

```go
package main

import "fmt"
func main() {
    x := 3
    y := 5
    fmt.Println("Before swap ", x, y)
    x, y = y, x
    fmt.Println("After swap ", x, y)
}
//Output
Before swap  3 5
After swap  5 3
```

Write a go program that find factourial of the given number?

```go
package main

import "fmt"
func factorial(a int) int {
    if a == 1 || a == 0 {
        return a
    }
    return a * factorial(a-1)
}
func main() {
    D := factorial(5)
    fmt.Println(D)
}
/*
Output
120
*/
```

Write a program to find the nth of fibonacci series?
Write a program for checking the character present or not in a string?
Write a program to compare the 2 slices of the byte?
Is it possible that if do not use wait group and still multiple go routine can not go into race condition or block each others.
Write a program to calculate my age.

```go
package main

import (
    "fmt"
    "time"
)
func main() {
    currentTime := time.Now()
    myBirth := "1994-mar-05"
    layout := "2006-Jan-02"
    myBirthdate, _ := time.Parse(layout, myBirth)
    age := currentTime.Sub(myBirthdate).Hours() / 8760 // 8760 is convert hour to year for non leap years
    fmt.Println(age)
}
/* Output
1994-03-05 00:00:00 +0000 UTC
28.66931682145781
*/
```

Write a program wether character is present or not in given string.

```go
package main

import "fmt"
func main() {
    myname := "yagnikpokal"
    character := "g"
    for _, j := range myname {
        if string(j) == character {
            fmt.Println("g is present in the string")
        }
    }
}

/*
Output
g is present in the string
*/
```

What is defer keyword in golang and how it is usefull?
Defer key word used to run a operations after function is completed.
Things need to remember
1) in a defer keyword the last function executes first let us say in a below code three will print first then two and the one
it will use to cleanup the resources,reset data and to close the files etc

```go
package main

import "fmt"
func main() {
    fmt.Println("Begning")
    defer fmt.Println("One")
```

```go
    defer fmt.Println("Two")
    defer fmt.Println("Three")
    fmt.Println("End")
}

/*
Output
go run defer.go
Begning
End
Three
Two
One
*/
```

Write a program wether the paranthesis is valid or not?

Write a program wether the string is palindrome or not?

```go
package main

import "fmt"
func main() {
    original_string := "madam1"
    var reverse_string = ""
    for i := len(original_string) - 1; i >= 0; i-- {
        reverse_string += string(original_string[i])
    }
    if original_string == reverse_string {
        fmt.Println("Palindrome")
    } else {
        fmt.Println("Not Palindrome")
    }
}

/*
go run palindrome.go
Not Palindrome
*/
```

Write a program to check wether paranthesis is valid or not?
or
Valid Parentheses circle bracket

```go
package main

import "fmt"
func checkValidString(s string) bool {
    size := len(s)
    stack := make([]byte, size)
    top := 0
    for i := 0; i < size; i++ {
        c := s[i]
        switch c {
        case '(':
```

```go
            stack[top] = c + 1 // '('+1 is ')'
            top++
        case ')':
            if top > 0 && stack[top-1] == c {
                top--
            } else {
                return false
            }
        }
    }
    return top == 0
}
func main() {
    input := "(*))"
    fmt.Println(checkValidString(input))
}
/*
go run parenthesis_valid_circular.go
false
*/
```

Write a program to check wether paranthesis is valid or not(circle + square + curly)?
or
Valid Parentheses circle + square + curly bracket

```go
package main

import "fmt"
func checkValidString(s string) bool {
    size := len(s)
    stack := make([]byte, size)
    top := 0
    for i := 0; i < size; i++ {
        c := s[i]
        switch c {
        case '(':
            stack[top] = c + 1 // '('+1 is ')'
            top++
        case '[', '{':
            stack[top] = c + 2
            top++
        case ')', ']', '}':
            if top > 0 && stack[top-1] == c {
                top--
            } else {
                return false
            }
        }
    }
    return top == 0
}
func main() {
    input := "{(yag)[]}"
    fmt.Println(checkValidString(input))
```

```
}
/*
go run parenthesis_valid_circularsquerecurly.go
true
*/
```

How man types of sorting algorithms are there?
1) Bubble sort
2) Quick sort
3) ballon sort
4) merge sort
5) radix sort

What is difference between stack and array?
Stack follows LIFO pattern wheres array uses index wise pattern for data stroing.

Data structor and algorithms
Stack
Stack is a linear data structor which follows the particular order in which operations are performed.
Stacks are variable size you can add a data as much as needed and can remove as needed.
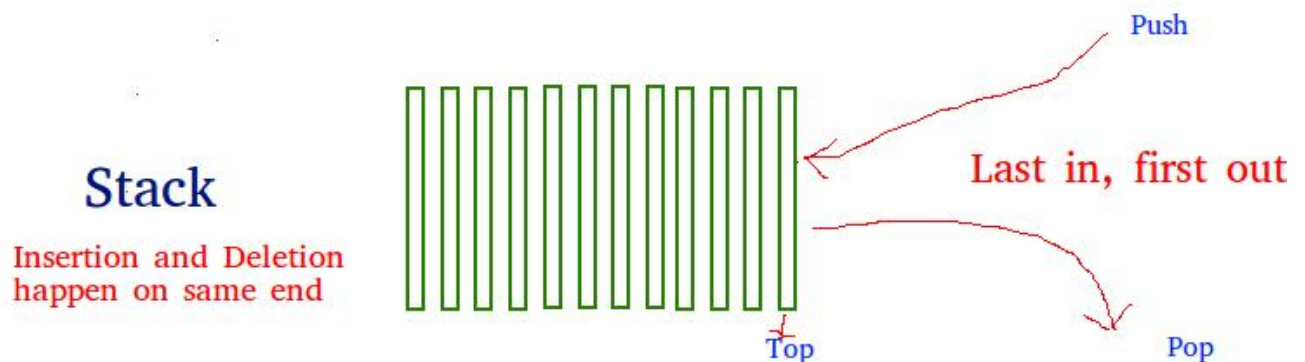
There are many real life examples of the stack. Consider the plate stacked on the canteen.
The plate which is at top is the first one to removed.
The plate at which bottom position will stay longest time.
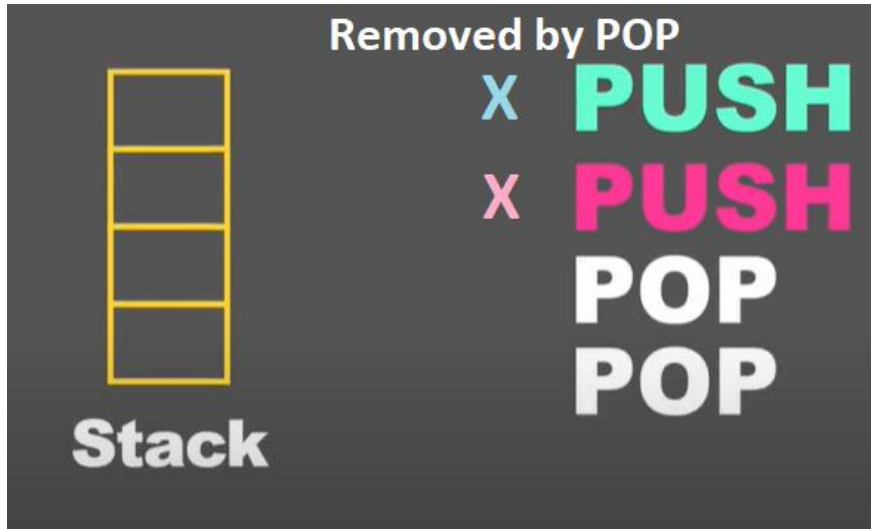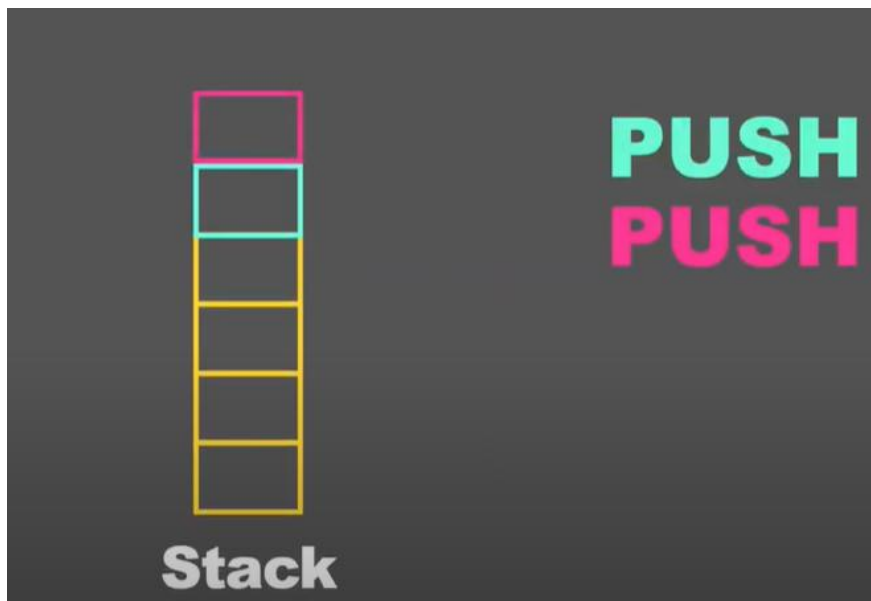
Write a program to add 3 element, print it and remove it in stack(push, pop)?

```go
package main

import (
    "fmt"
)
//Simple example for this in terms of push/pop
var stack []int
func Stack() {
    stack = append(stack, 10)
    stack = append(stack, 20)
    stack = append(stack, 30)
    for len(stack) > 0 {
        n := len(stack) - 1
        stack = stack[:n]
        fmt.Println(stack)
    }
}
func main() {
```

```
    Stack()
}
/*
go run stackpushpop.go
[10 20]
[10]
[]
*/
```

Write a program that add 3 interger and remove 2 integer in stack data structor using struct.

```go
package main

import "fmt"
// Stack can be accessed or in data or added data by using struct with slice
type Stack struct {
    items []int
}
// Create the 2 methods push and pop
// Push will add a value at the end
func (s *Stack) push(i int) {
    s.items = append(s.items, i)
}
//POP will remove value at the end
// and return the removed
func (s *Stack) pop() {
    s.items = s.items[:len(s.items)-1]
}
func main() {
    // Creating the stack
    myStack := Stack{}
    fmt.Println(myStack)
    // Push the items add the items in stack
    myStack.push(100)
    myStack.push(200)
    myStack.push(300)
    fmt.Println(myStack)
    // Pop the items remove the items from the stack
    myStack.pop()
    myStack.pop()
    fmt.Println(myStack)
}

/*
go run stackpushpopstruct.go
{[]}
{[100 200 300]}
{[100]}
*/
```

Queue

Queue is the last in first out data type.

Consider the cashier line on the mall. Who ever comes first serve first.

When we add a data then it will called the enqueue
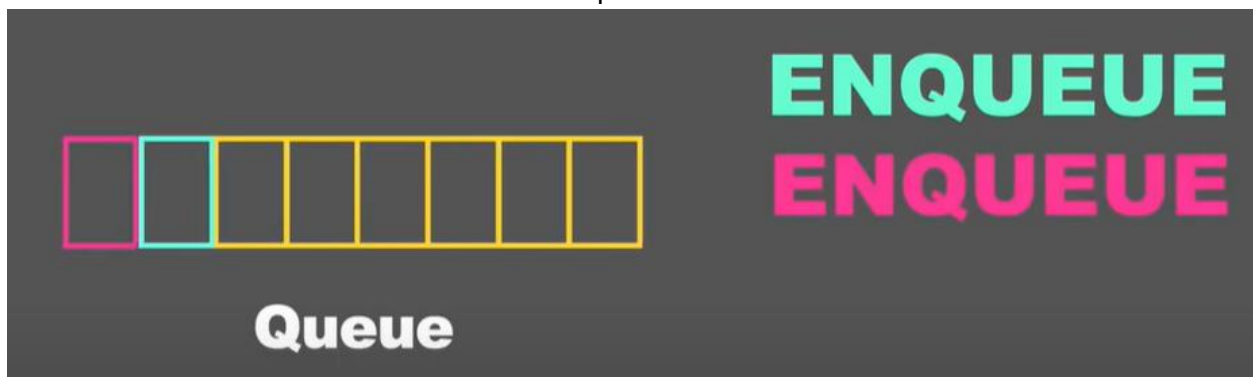
When we out th data then it is called as dequeue the data.
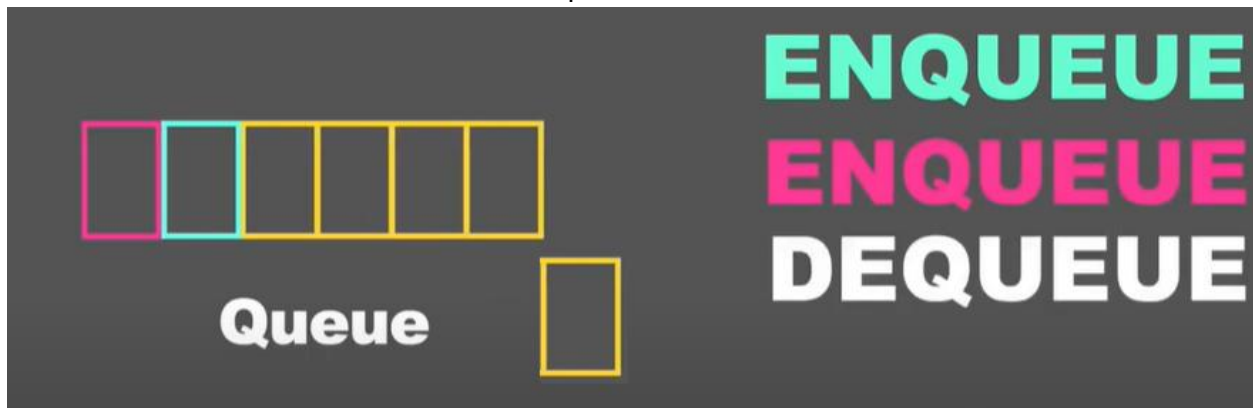

```go
package main

import "fmt"
// Queue can be accessed or in data or added data by using struct with slice
type Queue struct {
    items []int
}
// Create the 2 methods Enqueue and Dequeue
// Enqueue will add a value at the end
func (q *Queue) Enqueue(i int) {
    q.items = append(q.items, i)
}
//Dequeue will remove value at the front
```

```
// and return the removed
func (q *Queue) Dequeue() {
    q.items = q.items[1:]
}
func main() {
    // Creating the queue
    myQueue := Queue{}
    fmt.Println(myQueue)
    // Enqueue add the items in queue
    myQueue.Enqueue(100)
    myQueue.Enqueue(200)
    myQueue.Enqueue(300)
    fmt.Println(myQueue)
    // Dequeue rremove the items from the queue
    myQueue.Dequeue()
    fmt.Println(myQueue)
}

/*
go run queue.go
{[]}
{[100 200 300]}
{[200 300]}
*/
```

Deque
A deque is a double ended queue.
This is a structure in which element can be inserted or removed from both end.

Referances
https://www.youtube.com/watch?v=fsbm1FOSDJ0

What is the difference between stack and queue?
The main difference between stack and queue is the way the data removed.
Stack is LIFO
Queue is FIFO
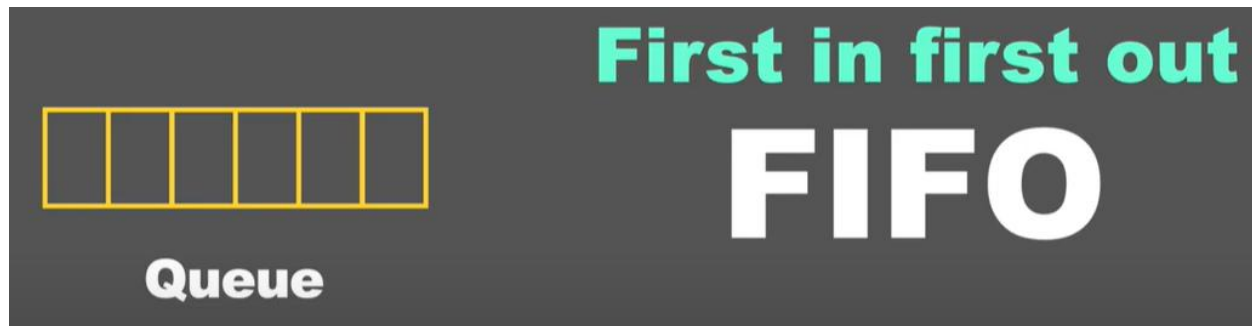
Linked list

Linked list is a linear data structure.

Elements are not stored in a contiguos manner.

The elements in the linked list are linked using pointer.

In a simple word linked list consists of nodes where each nodes contains a data field and referance link to the next node in the list.

Linked list is made up of 2 things

1) nodes (Data)

2) Referance the next field (Adress of next element)

3) Needed the detail of the first head

4) Length of linked list

Write a program that will create the 3 linked list & print the results.

or

Linked list insertion & deletion of value

or

Write a program that will create linked list, Print linked list, remove/ delete the nodes?

```go
package main

import "fmt"
// Contains the data and adress of next node
type Node struct {
    data int
```

```go
    next *Node
}
// LinkedList contains head adress and length of the LinkedList however length is not necessary every time
type LinkedList struct {
    head    *Node
    length int
}
// prepend function is used to add the data of single node in linked list
func (l *LinkedList) prepend(n *Node) {
    second := l.head
    l.head = n
    l.head.next = second
    l.length++
}
// deleteWithValue function is used to delete the data of single node in linked list
func (l *LinkedList) deleteWithValue(value int) {
    previousToDelete := l.head
    for previousToDelete.next.data != value {
        previousToDelete = previousToDelete.next
    }
    previousToDelete.next = previousToDelete.next.next
    l.length--
}
// We can not print all the linked list without the function
// We can print the single adress of the linkedlist
//prepend function will print all the linked list
func (l LinkedList) printListData() {
    toPrint := l.head
    for l.length != 0 {
        fmt.Printf("%d ", toPrint.data)
        toPrint = toPrint.next
        l.length--
    }
    fmt.Printf("\n")
}
func main() {
    // Creating the linked list
    myList := LinkedList{}
    // Add the data to linked list
    //node1 := &Node{data: 48}
    //node2 := &Node{data: 18}
    //  node3 := &Node{data: 16}
    // Print the linked list
    myList.prepend(&Node{data: 48})
    myList.prepend(&Node{data: 18})
    myList.prepend(&Node{data: 16})
    myList.printListData()
    myList.deleteWithValue(18)
    myList.printListData()
}

/*
go run linkedlist_insertion_deletion.go
16 18 48
16 48
*/
```

Referance
https://www.youtube.com/watch?v=8QoynPUY9_8

Golang Program to delete the first node from a linked list.

```go
package main

import "fmt"
type Node struct {
    value int
    next  *Node
}
func NewNode(value int, next *Node) *Node {
    var n Node
    n.value = value
    n.next = next
    return &n
}
func TraverseLinkedList(head *Node) {
    temp := head
    for temp != nil {
        fmt.Printf("%d ", temp.value)
        temp = temp.next
    }
    fmt.Println()
}
func DeleteFirstNode(head *Node) *Node {
    if head == nil {
        return head
    }
    newHead := head.next
    head.next = nil
    return newHead
}
func main() {
    head := NewNode(30, NewNode(10, NewNode(40, NewNode(40, nil))))
    fmt.Printf("Input Linked list is: ")
    TraverseLinkedList(head)
    head = DeleteFirstNode(head)
    fmt.Printf("After deleting first node of the linked list: ")
    TraverseLinkedList(head)
}
/*
go run linkedlist_deletfirstnode.go
Input Linked list is: 30 10 40 40
After deleting first node of the linked list: 10 40 40
*/
```

https://www.tutorialspoint.com/golang-program-to-delete-the-first-node-from-a-linked-list

Delete middle node of a linked list

```go
package main

import "fmt"
func main() {
    first := initList()
    first.AddFront(5)
    first.AddFront(4)
    first.AddFront(3)
    first.AddFront(2)
    first.AddFront(1)
    first.Head.Traverse()
    deleteMiddle(first.Head)
    fmt.Println("")
    first.Head.Traverse()
}
func initList() *SingleList {
    return &SingleList{}
}
type ListNode struct {
    Val  int
    Next *ListNode
}
func (l *ListNode) Traverse() {
    for l != nil {
        fmt.Println(l.Val)
        l = l.Next
    }
}
type SingleList struct {
    Len  int
    Head *ListNode
}
func (s *SingleList) AddFront(num int) {
    ele := &ListNode{
        Val: num,
    }
    if s.Head == nil {
        s.Head = ele
    } else {
        ele.Next = s.Head
        s.Head = ele
    }
    s.Len++
}
func deleteMiddle(head *ListNode) *ListNode {
    if head == nil {
        return nil
    }
    size := sizeOfList(head)
    mid := size / 2
    if mid == 0 {
        return head.Next
    }
    curr := head
```

```go
        for i := 0; i < mid-1; i++ {
            curr = curr.Next
        }
        prev := curr
        midNode := prev.Next
        if midNode == nil {
            return head
        }
        midNext := midNode.Next
        prev.Next = midNext
        return head
}
func sizeOfList(head *ListNode) int {
    l := 0
    for head != nil {
        l = l + 1
        head = head.Next
    }
    return l
}
/*
go run linkedlist_deletemiddlenode.go
1
2
3
4
5

1
2
4
5
*/
```

Sort the single linked list
or
write a program to sort the linkedlist

```go
package main

import (
    "fmt"
)
//Create prototype
// LL container which going to store list
type LL struct {
    list *linklist
}
// linklist for value and next pointer details
type linklist struct {
    val  int
    next *linklist
}
// createNode use for create node for list
func createNode(value int) *linklist {
    return &linklist{
```

```go
            val:  value,
            next: nil,
        }
}
func (lstVal *LL) insertAtBeginning(data int) {
    if nil == lstVal.list {
        lstVal.list = createNode(data)
        return
    }
    tempNode := createNode(data)
    head := lstVal.list
    tempNode.next = head
    lstVal.list = tempNode
}
func (lstVal *LL) printList() {
    if nil != lstVal && nil != lstVal.list {
        head := lstVal.list
        for nil != head {
            fmt.Printf(" %d", head.val)
            head = head.next
        }
    }
    fmt.Println()
}
func (lstVal *LL) deleteFromBeginning() {
    if nil != lstVal && nil != lstVal.list {
        head := lstVal.list
        lstVal.list = head.next
        head = nil
    }
}
func sort(ll *linklist, insertedNode *linklist) *linklist {
    head := ll
    if ll.val > insertedNode.val {
        insertedNode.next = ll
        ll = insertedNode
    } else {
        for head.next != nil && head.next.val < insertedNode.val {
            head = head.next
        }
        insertedNode.next = head.next
        head.next = insertedNode
    }
    return ll
}
func sortTheLinkedList(ll *linklist) *linklist {
    head := ll
    sortedList := new(linklist)
    var firstTime bool = true
    for head != nil {
        nextNode := head.next
        if firstTime {
            sortedList = head
            sortedList.next = nil
            firstTime = false
        } else {
```

```go
            sortedList = sort(sortedList, head)
        }
        head = nextNode
    }
    return sortedList
}
func main() {
    staticList := []int{5, 7, 1, 3, 4, 9}
    linklst := new(LL)
    for _, value := range staticList {
        linklst.insertAtBeginning(value)
    }
    fmt.Println("PrintList")
    linklst.printList()
    linklst.list = sortTheLinkedList(linklst.list)
    fmt.Println("After Sorting PrintList")
    linklst.printList()
}

/*
go run linkedlistsorted.go
PrintList
 9 4 3 1 7 5
After Sorting PrintList
 1 3 4 5 7 9

*/
```

Sort the two single linked list
or
write a **function** to sort the two linkedlist

```go
package main

import "fmt"
// Listnode represents a node in a linked list
type Listnode struct {
    Val   int
    Next *Listnode
}
// linksort sorts two linked lists and returns a new sorted linked list
func linksort(l1 *Listnode, l2 *Listnode) *Listnode {
    var dummy = new(Listnode)
    var p = dummy
    for l1 != nil && l2 != nil {
        if l1.Val < l2.Val {
            p.Next = l1
            l1 = l1.Next
        } else {
            p.Next = l2
            l2 = l2.Next
        }
        p = p.Next
    }
    if l1 != nil {
        p.Next = l1
```

```go
        } else {
            p.Next = l2
        }
    }
    return dummy.Next
}
func main() {
    // Create two lists to sort
    l1 := &Listnode{
        Val: 1,
        Next: &Listnode{
            Val: 3,
            Next: &Listnode{
                Val: 5,
            },
        },
    }
    l2 := &Listnode{
        Val: 2,
        Next: &Listnode{
            Val: 4,
            Next: &Listnode{
                Val: 6,
            },
        },
    }
    // Sort the lists
    sortedList := linksort(l1, l2)
    // Print the sorted list
    for curr := sortedList; curr != nil; curr = curr.Next {
        fmt.Println(curr.Val)
    }
}
/*
go run linkedlist_2sortedfunction.go
1
2
3
4
5
6
*/
```

Sort the two single linked list program
or
write a **program** to sort the two linkedlist

```go
package main

import (
    "container/list"
    "fmt"
)
func main() {
    nodeA := list.New()
```

```go
        nodeA.PushBack(10)
        nodeA.PushBack(4)
        nodeA.PushBack(12)
        nodeA.PushBack(8)
        nodeB := list.New()
        nodeB.PushBack(5)
        nodeB.PushBack(10)
        nodeB.PushBack(15)
        nodeB.PushFront(1)
        nodeB.PushBack(13)
        //iterate over nodeB
        for i := nodeB.Front(); i != nil; i = i.Next() {
            fmt.Println(i.Value)
        }
        sortLinkedList(nodeA)
        sortLinkedList(nodeB)
        fmt.Println("after sort, nodeA: ")
        for i := nodeA.Front(); i != nil; i = i.Next() {
            fmt.Println(i.Value)
        }
        fmt.Println("after sort, nodeB: ")
        for i := nodeB.Front(); i != nil; i = i.Next() {
            fmt.Println(i.Value)
        }
        // passing two sorted list to merge
        mergedList := mergeSortedLinkedList(nodeA, nodeB)
        fmt.Println("after merging two sorted list, mergedList: ")
        for i := mergedList.Front(); i != nil; i = i.Next() {
            fmt.Println(i.Value)
        }
}
func sortLinkedList(node *list.List) *list.List {
    current := node.Front() //pointing to first node in the list
    if node.Front() == nil {
        return nil
    } else {
        for current != nil {
            index := current.Next() //pointing to second node in the list
            for index != nil {
                if current.Value.(int) > index.Value.(int) {
                    //comparing and swaping the nodes
                    temp := current.Value
                    current.Value = index.Value
                    index.Value = temp
                }
                index = index.Next() //increasing the pointer to next node
            }
            current = current.Next() //increasing the pointer to next node
        }
    }
    return node
}
func mergeSortedLinkedList(nodeA, nodeB *list.List) *list.List {
    //here i expect two sorted list of length > 0 passed
    //you can add more validation, leaving it for you..
    node1 := nodeA.Front() //pointing to first node of nodeA, HEAD
```

```go
        node2 := nodeB.Front() //pointing to first node of nodeB, HEAD
        resNode := list.New()  //we will store and return our sorted merged list here
        for node1 != nil && node2 != nil {
            if node1.Value.(int) < node2.Value.(int) {
                resNode.PushBack(node1.Value)
                node1 = node1.Next()
            } else {
                resNode.PushBack(node2.Value)
                node2 = node2.Next()
            }
        }
        //what if node1.length > node2.length ? add remaining element of node1 to the result list
        for node1 != nil {
            resNode.PushBack(node1.Value)
            node1 = node1.Next()
        }
        // similarly what if node2.length > node1.length ? add remaining element of node2 to the result list
        for node2 != nil {
            resNode.PushBack(node2.Value)
            node2 = node2.Next()
        }
        return resNode
}
/*
go run linkedlist_merge2sorted.go
1
5
10
15
13
after sort, nodeA:
4
8
10
12
after sort, nodeB:
1
5
10
13
15
after merging two sorted list, mergedList:
1
4
5
8
10
10
12
13
15

*/
```

reverse linkedlist

or

Write a program to reverse the linked list?

or

Write a program to init, insert, print and reverse the linked list?

```go
package main

import "fmt"
```

```go
func main() {
    first := initList()
    first.AddFront(1)
    first.AddFront(2)
    first.AddFront(3)
    first.AddFront(4)
    first.Head.Traverse()
    first.Reverse()
    fmt.Println("")
    first.Head.Traverse()
}
func initList() *SingleList {
    return &SingleList{}
}
type ListNode struct {
    Val  int
    Next *ListNode
}
func (l *ListNode) Traverse() {
    for l != nil {
        fmt.Println(l.Val)
        l = l.Next
    }
}
type SingleList struct {
    Len  int
    Head *ListNode
}
func (s *SingleList) Reverse() {
    curr := s.Head
    var prev *ListNode
    var next *ListNode
    for curr != nil {
        next = curr.Next
        curr.Next = prev
        prev = curr
        curr = next
    }
    s.Head = prev
}
func (s *SingleList) AddFront(num int) {
    ele := &ListNode{
        Val: num,
    }
    if s.Head == nil {
        s.Head = ele
    } else {
```

```
        ele.Next = s.Head
        s.Head = ele
    }
    s.Len++
}
/*
go run linkedlistreverse.go
4
3
2
1

1
2
3
4
*/
```

Comparison between linkelist and array
or How linked list are different from array?
Let us take an example of the linked list along with array as below.
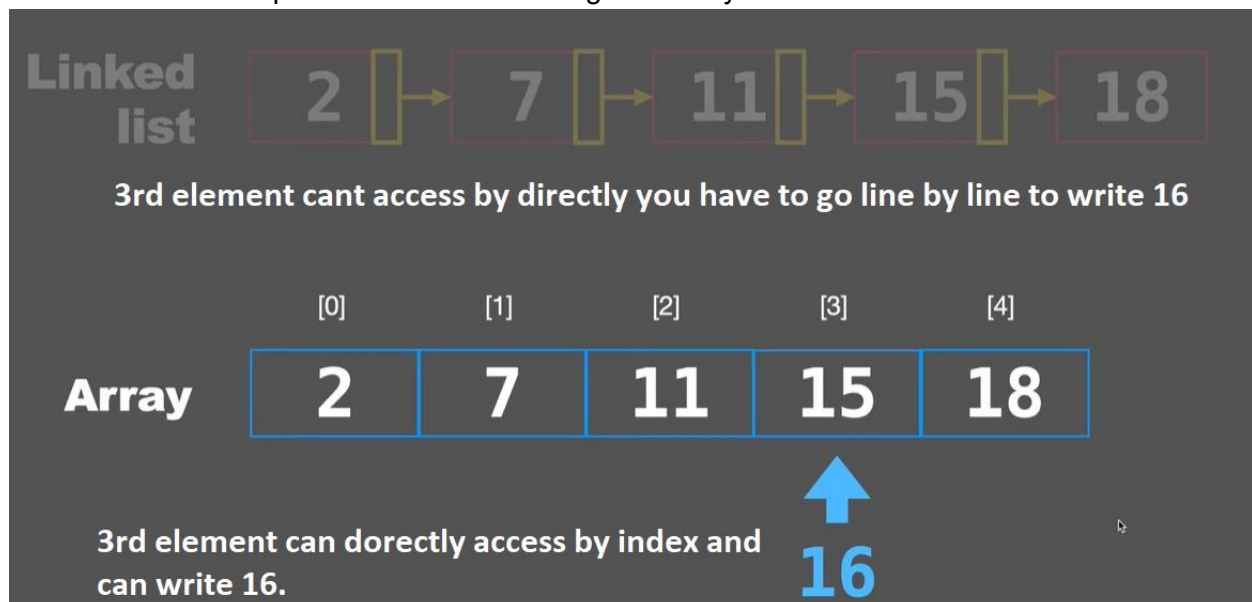
Now let us change the 3rd element in array, to do that we can directly check index 3 and then change the value without iterating whole array.
But for the linked list to change the value at 3rd location we have to reach there from 1st element, 2nd element and then 3rd element and can change the value so every time we have to go line by line instead of the direct index like array.

Why we use linked list instead of array?

Adding the element on first location on linked list takes **smaller time**. then array. Let us say if you want to add 99 value on beginning 0th element then it takes constant time.

But if want to add 99 on the 0th location of array then it will takes **longer time**. Because in array you have to shift the all the element by one and then you can add 99 on the beginning.

Doubly linked list
Doubly linked list will have node + previous address + next address.

The main use of doubly linked list is adding the element at back side will easier and reduce time.

Doubly linked list

2 ⇄ 7 ⇄ 11 ⇄ 16 ⇄ 18 ⇄ 47
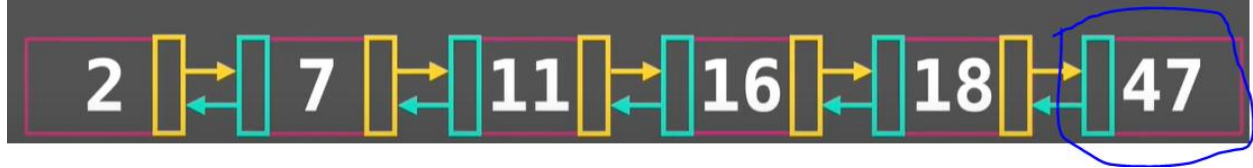
Tree

Tree,binary tree ans bnary search tree

```
        100      Root
        /  \
     52    76    Parent
     /  \
   19    76      Children
  /  \  /  \
 1  6 8  9    Leaves
```

Tree	Binary tree	Binary search tree

```
       8                    8                  8
      / \                  / \                / \
     7   5                7   5              7   8
    /|\   \              / \  / \           / \  / \
   4 5 3   3            4   3 6           3    4 5   8
      \        \
       6        1
```

Tree

Tree is having as many as children and leaf

Binary tree

Binary tree each node n more then 2 children.They are called as left or right children.
Representation of binary tree.

Binary search tree

In a binary search tree left child is less then right child

```
        8
       / \
      7   9
   Small  Bigger
```

How to insert nodes?
Nodes are always inserted at leaves.
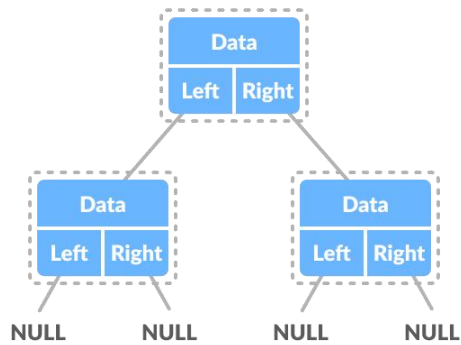
Types of binary tree
1) Full binary tree
2) Perfect binary tree
3) Complete binary tree
4) Balanced binary tree
5) Degenerate or pathological tree
6) Skewed binary tree

What is called as perfect binary tree
or
Perfect binary tree have 2 conditions to be match
1) All of its internal node has 2 exact children
2) All the leaf nodes are on the same level.

Perfect binary tree	Not perfect	Not perfect
8	8	8
/ \	/ \	/ \
7 5	7 5	7 6
/ \ / \	/ \ \	/ \ / \
4 3 9 3	4 3 6	4 3 5 9
		/ \
		5 3

Complete binary tree
A binary tree is called as complete binary tree if the element from top to bottom and left to right meets.

8	8	8
/ \	/ \	/ \
7 5	7 5	7 6

```
    / \    /           /                / \
   4   3  9           4                4   3
```
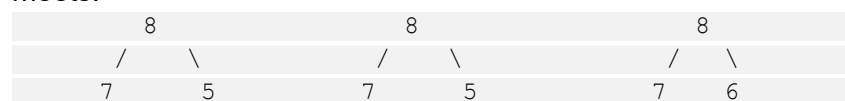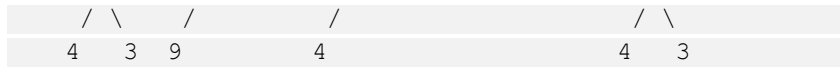
No complete binary tree

```
        8                  8                    8
      /    \             /    \               /    \
     7      5           7      5             7      6
    / \      \         / \                    \
   4   3      9       6   4                    3
```

Full binary tree

A binary tree is called as full binary tree if parent node / interna node has either two or no children.

Full binary tree

```
        8                  8                    8
      /    \             /    \               /    \
     7      5           7      5             7      6
    / \                / \                 / \
   4   3              6   4               2   3
                                            / \
                                           5   3
```

Not full binary tree

```
        8                  8                    8
      /    \             /    \               /    \
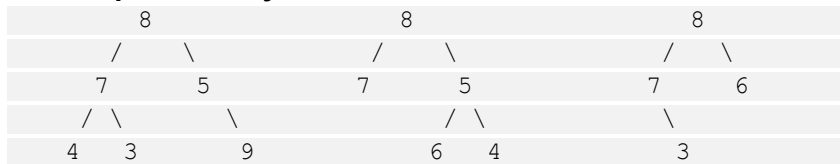     7      5           7      5             7      6
      \      \           /                    \
       3      9         6                      3
```
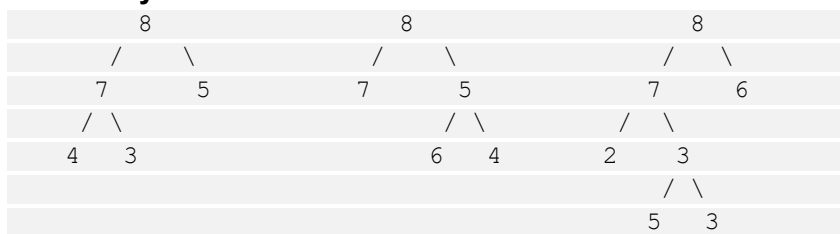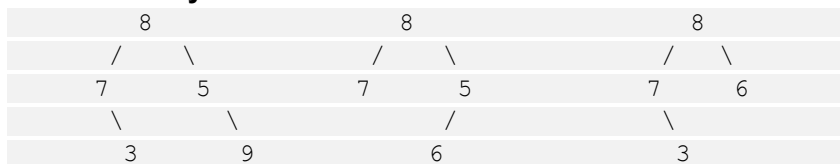
Referance
https://www.programiz.com/dsa/binary-tree

Operations on the binary tree
Basic operations
1) Insertion an element
2) Removing element
3) searching for element
4) Traversing the tree

Advanced operations
1) Finding the height of the tree
2) Find the level of node of tree
3) Find the size of entire tree

Binary search tree insert, search and print, delete, max no, min number
or

write a program to insert the element in binary tree and search the element and print the elements using methods.
or
Binary search tree
1) Insert
2) Delete
3) Traverse
4) max value
5) Min value

```go
package main

import (
    "errors"
    "fmt"
)
// TreeNode data structure represents a typical binary tree
type TreeNode struct {
    val   int
    left  *TreeNode
    right *TreeNode
}
func main() {
    t := &TreeNode{val: 8}
    t.Insert(1)
    t.Insert(2)
    t.Insert(3)
    t.Insert(4)
    t.Insert(5)
    t.Insert(6)
    t.Insert(7)
    t.Find(11)
    t.Delete(5)
    t.Delete(7)
    t.PrintInorder()
    fmt.Println("")
    fmt.Println("min is %d", t.FindMin())
    fmt.Println("max is %d", t.FindMax())
}
// PrintInorder prints the elements in order
func (t *TreeNode) PrintInorder() {
    if t == nil {
        return

    }
    t.left.PrintInorder()
    fmt.Print(t.val)
    t.right.PrintInorder()
}
// Insert inserts a new node into the binary tree while adhering to the rules of a perfect BST.
func (t *TreeNode) Insert(value int) error {
    if t == nil {
        return errors.New("Tree is nil")

    }
    if t.val == value {
        return errors.New("This node value already exists")
```

```go
    }
    if t.val > value {
        if t.left == nil {
            t.left = &TreeNode{val: value}
            return nil
        }
        return t.left.Insert(value)
    }
    if t.val < value {
        if t.right == nil {
            t.right = &TreeNode{val: value}
            return nil
        }
        return t.right.Insert(value)
    }
    return nil
}
// Find finds the treenode for the given node val
func (t *TreeNode) Find(value int) (TreeNode, bool) {
    if t == nil {
        return TreeNode{}, false
    }
    switch {
    case value == t.val:
        return *t, true
    case value < t.val:
        return t.left.Find(value)
    default:
        return t.right.Find(value)
    }
}
// Delete removes the Item with value from the tree
func (t *TreeNode) Delete(value int) {
    t.remove(value)
}
func (t *TreeNode) remove(value int) *TreeNode {
    if t == nil {
        return nil
    }
    if value < t.val {
        t.left = t.left.remove(value)
        return t
    }
    if value > t.val {
        t.right = t.right.remove(value)
        return t
    }
    if t.left == nil && t.right == nil {
        t = nil
        return nil
    }
    if t.left == nil {
        t = t.right
        return t
    }
    if t.right == nil {
```

```go
            t = t.left
            return t
        }
        smallestValOnRight := t.right
        for {
            //find smallest value on the right side
            if smallestValOnRight != nil && smallestValOnRight.left != nil {
                smallestValOnRight = smallestValOnRight.left
            } else {
                break
            }
        }
        t.val = smallestValOnRight.val
        t.right = t.right.remove(t.val)
        return t
}
// FindMax finds the max element in the given BST
func (t *TreeNode) FindMax() int {
        if t.right == nil {
            return t.val
        }
        return t.right.FindMax()
}
// FindMin finds the min element in the given BST
func (t *TreeNode) FindMin() int {
        if t.left == nil {
            return t.val
        }
        return t.left.FindMin()
}
/*
go run binarysearchtreeinsertdeletesearchmacmin.go
123468
min is %d 1
max is %d 8
*/
```

Binary tree insert and print operations

```go
package main

/* Below tree might be wrong
                        100
                       /  \
                    -20
                     /
                  -50
                  / \
               -60   -15
                       \
                        50
                       /
                     15
                     /
                    5
                   / \
                -10   60
```

```go
                                     /  \
                                    55   85
*/
import (
    "fmt"
    "io"
    "os"
)
type BinaryNode struct {
    left  *BinaryNode
    right *BinaryNode
    data  int64
}
type BinaryTree struct {
    root *BinaryNode
}
func (t *BinaryTree) insert(data int64) *BinaryTree {
    if t.root == nil {
        t.root = &BinaryNode{data: data, left: nil, right: nil}
    } else {
        t.root.insert(data)
    }
    return t
}
func (n *BinaryNode) insert(data int64) {
    if n == nil {
        return
    } else if data <= n.data {
        if n.left == nil {
            n.left = &BinaryNode{data: data, left: nil, right: nil}
        } else {
            n.left.insert(data)
        }
    } else {
        if n.right == nil {
            n.right = &BinaryNode{data: data, left: nil, right: nil}
        } else {
            n.right.insert(data)
        }
    }
}
func print(w io.Writer, node *BinaryNode, ns int, ch rune) {
    if node == nil {
        return
    }
    for i := 0; i < ns; i++ {
        fmt.Fprint(w, " ")
    }
    fmt.Fprintf(w, "%c:%v\n", ch, node.data)
    print(w, node.left, ns+2, 'L')
    print(w, node.right, ns+2, 'R')
}
func main() {
    tree := &BinaryTree{}
    tree.insert(100).
        insert(-20).
```

```
            insert(-50).
            insert(-15).
            insert(-60).
            insert(50).
            insert(60).
            insert(55).
            insert(85).
            insert(15).
            insert(5).
            insert(-10)
        print(os.Stdout, tree.root, 0, 'M')
}

/*
go run binarytreeinsertprint.go
M:100
  L:-20
    L:-50
      L:-60
    R:-15
      R:50
        L:15
          L:5
            L:-10
        R:60
          L:55
          R:85


*/
```

Heaps
There are two types of heap
1) Max heap
2) Min heap
Time complexity -->> **o(logn)**   // This is due to the height of tree
Space complexity -->> **o(logn)**
Heap is a datastructure it is a special form of binary tree.
At the root of tree it has a min and max values
Heap can be represents by the node object or within the array.
The heapify algorithm takes randomly ordered array and gives structure of the heap.
Heapify algorithm is also called as percolate down

What is heapify algorithem?
The property of the rearranging the heap by comparing each parent with its children recursively is known as the heapify.
1) The 0th element must be leaf
2) The 0th element must be greater then left or right child & it applied to all subtree.

0th element must be higher then max of left and right el

What is heap push?
Push is inserting element in algorithem. Push is sometimes also follow maximum size to reduce memory.
Root is at index 0 of the array.
Left child of index i is at (2*i + 1).
Right child of index i is at (2*i + 2).
Parent of index i is at (i-1)/2.
Example:
```
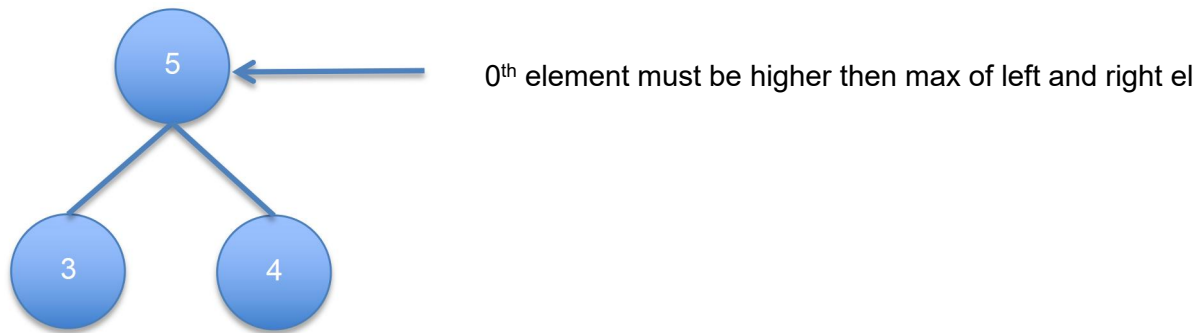Input: []int{8, 7, 5, 4, 3}
        8
      /   \
     7     5
    / \
   4   3
```

```
New = Input + insert{6} = []int{8, 7, 5, 4, 3,6}
        8                       8                       8
      /   \                   /   \                   /   \
     7     5 -->>            7     5   -->>          7      6
    / \                    / \   / \               / \    / \
   4   3                  4   3 6                  4   3  5
```

Write a program to min heapify the given array?

What is heapify algorithem?
The property of the rearranging the heap by comparing each parent with its children recursively is known as the heapify.
1) The 0th element must be leaf
2) The 0th element must be greater then left or right child & it applied to all subtree.

0th element must be higher then max of left and right el

What is heap push?
Push is inserting element in algorithem. Push is sometimes also follow maximum size to reduce memory.
Root is at index 0 of the array.
Left child of index i is at (2*i + 1).
Right child of index i is at (2*i + 2).
Parent of index i is at (i-1)/2.
Example:
```
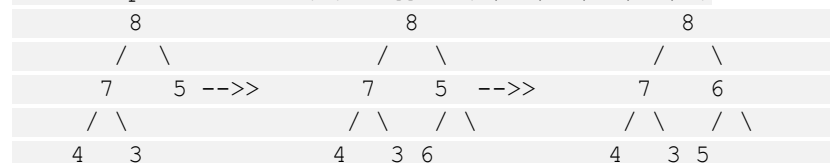Input: []int{8, 7, 5, 4, 3}
        8
      /   \
     7     5
    / \
   4   3
```

```
New = Input + insert{6} = []int{8, 7, 5, 4, 3,6}
        8                       8                       8
      /   \                   /   \                   /   \
     7     5 -->>            7     5   -->>          7      6
    / \                    / \   / \               / \    / \
   4   3                  4   3 6                  4   3  5
```

Write a program to min heapify the given array?

Root is at index 0 of the array.
Left child of index i is at (2*i + 1).
Right child of index i is at (2*i + 2).
Parent of index i is at (i-1)/2.
Example:
Input: []int{4, 12, 3, 6, 5}

```
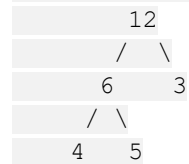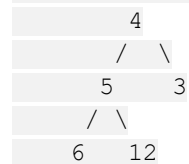        4
      /   \
     12    3
    / \
   6   5
```

Max-Heap: []int{12,6,3,4,5}

```
       12
      /   \
     6     3
    / \
   4   5
```

Min-Heap: []int{3,12,4,6,5}   //0th element

```
        3
      /   \
     12    4
    / \
   6   5
```

Min-Heap: []int{4,5,3,6,12}   //1th element

```
        4
      /   \
     5     3
    / \
   6   12
```

Min-Heap: []int{4,12,3,6,5}   //2th element

```
        4
      /   \
     12    3
    / \
   6   5
```

Write the program to implement the mean heap,

```go
package main

import "fmt"
func heapify(heap *[]int, i int) {
    smallest := i
    lChild := 2*i + 1
    rChild := 2*i + 2
    if lChild < len(*heap) && (*heap)[lChild] < (*heap)[smallest] {
        smallest = lChild
    }
    if rChild < len(*heap) && (*heap)[rChild] < (*heap)[smallest] {
        smallest = rChild
    }
```

```
    if smallest != i {
        (*heap)[i], (*heap)[smallest] = (*heap)[smallest], (*heap)[i]
        heapify(heap, smallest)
    }
}
func main() {
    input := []int{4, 12, 3, 6, 5}
    fmt.Println(input)
    heapify(&input, 0)
    fmt.Println(input)
}
/*
go run minheap.go    //0th time
[4 12 3 6 5]
[3 12 4 6 5]

go run minheap.go    //1th time
[4 12 3 6 5]
[4 5 3 6 12]

go run minheap.go    //2th time
[4 12 3 6 5]
[4 12 3 6 5]
go run minheap.go    //3th time
[4 12 3 6 5]
[4 12 3 6 5]
*/
```

The applications of the heap
1) Heap sorting uses to sort the array in O(nLogn) time
2) Priority queue can be effectively implemented by the heap due to it supports inserts(),
delete(), extractmax() operations in O(logn) times
3) Problems like Kth largest element in array, Sort an almost sorted array, Merge k sorted array
etc
The time complexity of heap,

Function	Time Complexity
get_min()	O(1)
insert_minheap()	O(logN)
delete_minimum()	Same as insert - O(logN)
heapify()	O(logN)

delete_element()	**O(logN)**

Referances,
https://www.geeksforgeeks.org/heap-data-structure/
https://levelup.gitconnected.com/how-to-build-a-min-max-heap-in-go-5090617a3142
https://www.youtube.com/watch?v=3DYIgTC4T1o

How to check/see the environment variable in VS?
go env   // Command to see the environment variables in go

How to set environment variable?
set GOOS=linux
set GOARCH=arm64
The applications of the heap
Write a program to add the last element of array, if it is 9 then add to next to last
Or
plus one arrray last element
or
plus one leetcode

```go
package main

import "fmt"
func plusOne(digits []int) []int {
    length := len(digits)
    if length == 0 {
        return []int{1}
    }
    digits[length-1]++
    for i := length - 1; i > 0; i-- {
        if digits[i] < 10 {
            break
        }
        digits[i] -= 10
        digits[i-1]++
    }
    if digits[0] > 9 {
        digits[0] -= 10
        digits = append([]int{1}, digits...)
    }
    return digits
}
func main() {
    arr := []int{9, 9, 9}
    fmt.Println(arr)
    plusOne(arr)
    fmt.Println(arr)
}
```

```
/*
go run plusone.go
[9 9 9]
[0 0 0]

*/
```

How to set the go module?
go env -w GO111MODULE=on
go mod init concurrency

How to install packages for postgre sql database?
go get github.com/jackc/pgconn
go get github.com/jackc/pgx/v4
go get github.com/jackc/pgx/v4/stdlib

Which package is good for the session log?
go get github.com/alexedwards/scs/v2

Which package is good for the session log storage?
Reddis databse is good
go get github.com/alexedwards/scs/redisstore

Which router is good for the golang?
chi router is really good
go get github.com/go-chi/chi/v5

Which package os good for the middleware?
go get github.com/go-chi/chi/v5

What is Docker?
Docker is the platform for developeing, shipping and running the applications.

How docker is born?
When we move our application from our computing environment to another or stage to
production then it will not work due to environment change or OS change. Docker or container
will solve that issue.

How docker solve the problems?
Put simply, a container consists of an entire runtime environment: an application, plus all its
dependencies, libraries and other binaries, and configuration files needed to run it, bundled into
one package.

What things needed to run docker on windows 10 home?
1) Virtualisation must be enable
This can be enabled by pressing CTRL + ALT + DLT -->> Task manager -->> Performance
check that virtulisation must be enable
If it is disabled then go to bios setting --->> System setting --->> enable virtulisation --->> F10 and
restart
2) WSL 2 driver is needed.
Can be downloaded from
https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi
and install the same.

How does docker compose file looks like? & command to run the file?
Command is docker-compose up -d

```
# file name docker-compose.yml
# command to run this docker file docker-compose up -d
# -d will run docker in background
version: '3'

services:
  #  start Postgres, and ensure that data is stored to a mounted volume
  postgres:
    image: 'postgres:14.2'
    ports:
      - "5432:5432"
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: concurrency
    volumes:
      - ./db-data/postgres/:/var/lib/postgresql/data/
  #  start Redis, and ensure that data is stored to a mounted volume
  redis:
    image: 'redis:alpine'
    ports:
      - "6379:6379"
    restart: always
    volumes:
      - ./db-data/redis/:/data
  #  start mailhog
  mailhog:
    image: 'mailhog/mailhog:latest'
    ports:
      - "1025:1025"
      - "8025:8025"
    restart: always
```

How to up and down the docker?

docker-compose up -d
docker-compose down

How to connect databses from docker? as per above docket yml file?
Open beekeeper studio.
open new connections-->>select postgres in type -->> as per the above yml file fill details of port no, user, password default database as concurrency.

What is make file and use of make file?
make file comes in all 3 OS supports windows, mac, linux.
Make file will reduce developers effort by converting bunch of command to single command.
Like let us say if I used the data base then I it will gove username, PW, build, run code etc with single command.

How does make file looks like?
For example when I run
make start

```
#name of make file is makefile

DSN=host=localhost port=5432 user=postgres password=password dbname=concurrency
sslmode=disable timezone=UTC connect_timeout=5
BINARY_NAME=myapp.exe

## build: builds all binaries
build:
        @go build -o ${BINARY_NAME} ./cmd/web
        @echo back end built!

run: build
        @echo Starting...
        set "DSN=${DSN}"
        start /min cmd /c ${BINARY_NAME} &
        @echo back end started!

clean:
        @echo Cleaning...
        @DEL ${BINARY_NAME}
        @go clean
        @echo Cleaned!

start: run

stop:
        @echo "Stopping..."
        @taskkill /IM ${BINARY_NAME} /F
        @echo Stopped back end

restart: stop start
```

```
test:
        @echo "Testing..."
        go test -v ./...
```

What to do if make command not run?
Install make in wondows
Open command prompt by adminstrativ
type
@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -
InputFormat None -ExecutionPolicy Bypass -Command
"[System.Net.ServicePointManager]::SecurityProtocol = 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))" &&
SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"

go to your browser
choco install make

referance
https://answers.microsoft.com/en-us/windows/forum/all/powershell-terminal-in-vs-code-make-
the-term-make/74d69621-c91e-4929-83c2-2252f0371397

Slices are reference type or value type?
Map, pointer and slices are reference types in go
Why we can not append into the slice in the function?

```go
package main

import "fmt"
func ModifyData(a []int) {
    a[0] = 5
}
func AddData(a []int) {
    a = append(a, 4)
}
func main() {
    a := []int{1, 2, 3} // Slice
    AddData(a) //{1,2,3,4}
    fmt.Println(a) // 1,2,3,4
    ModifyData(a) //5,2,3,4
    fmt.Println(a) // 5,2,3,4
}
/*
go run sliceappendnotworkfunction.go
[1 2 3]
[5 2 3]
*/
```

To install docker in ubuntu
sudo snap install docker
to compile the docker file(.yml)
docker-compose up -d

If permission deni then fire below command
sudo chmod 666 /var/run/docker.sock

What is CI/CD pipeline?
or
what is ci/cd
or
continuous integration/continuous delivery.
or
continuos integration/continuos deployement

Element of the CI/CD

```
Source   ->> Build  -->> Test   -->>    Deploy
git push       compile      smoke          staging
               docker       unit           QA
                            integration     Production
```
Smaller organisation will not d the ci/cd. Instea they can create make file for first 3 stages above
and can use jira for staging and QA and can add production manually.
While CI/CD can be do below things.
It will send email to team or developer if anything in the above step is going wrong.
Benefits of CI?
Improving collabration and quality
Better code quality
Team efficiency

What is continuos delivery?
After iteration or code changes code must be eaily testable
Benefits of cd?
Frequent deployement
Provides real time feedbacks
Faster software builds
Better time to market

CI/CD can be achieved by below tools.
1) Jenkins (One of the oldest tool)

2) Semaphore (Fastest and cloud based)
3) git also have ci/cd tool

Jenkins
1) It is on premises
Supports language specific files for
Needed a machine for setup and dedicated person for the same where semaphore solve that issue
It comes with reach library of the plugins
Need to add extra RAM/CPU as needed
No ssh access
Can be run on EC2

Semaphore
1) It is cloud base
Steps -->> Authentication to git -->> Import rep as a project
It offers block diagram kind of facility.
Supports YAML file
Has built in docker
No need to add machines/ram/CPU it is auto scaling
It has ssh access for anything gone wrong

Other CI/CD tools
1) buddy
2) circle ci
3) Azure devops
4) Bamboo
5) Build bot

What are linea datastructor?
1) Array
2) Linkedlist
3) Stack
4) Queue

What are hirarchical datastructor?
1) Tree
2) Graph

Graph datastructor
What is graph?
It is a collection of the nodes connected through edges.

Graph is a non linear data structor

```
     10 - 65
    / | \  /
  52  6  76
   |   \   <<--    Edges/Lines/Arcs
   5 -- 75 <<--    Nodes/Vertex
```

Edge :- Uniquely defined by the its 2 endpoints
Vertex :- Fundamental unit/entity of the ehich graph are formed.

V = {10,65,52,6,76,5,75}   // Vertex
E = {{10,65}, {10,52}, {10,6}, {10,76},{65,76},{52,5},{6,75},{5,75}}   // Edges total 8 edges
Tree is also a graph but it has its own rules.
Graph is a composed of vertices and edges. The graph is denoted by the G(E, V)
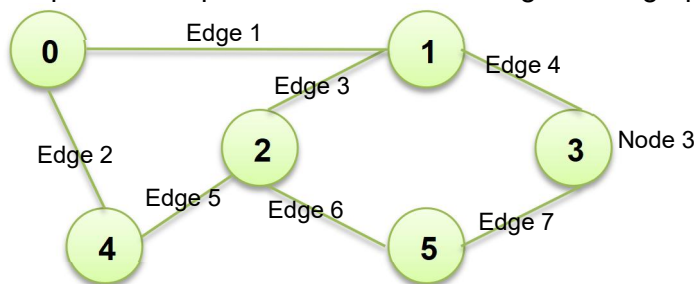
Applictions of the graphs:
1) Maps
2) Social network like FB, Linkedin
3) State transisition diagram
4) Employee internal company relations ship and position
5) Website backlinks

Types of the graphs
1) Null graph
2) Trivial graph
3) Undirected graph
4) Directed graph
5) Connected graph
6) Disconnected graph
7) Regular graph
8) Complete graph
9) Cycle graph
10) Cyclick graph
11) Directed acyclick graph
12) Bipartitte graph
13) Weighted graph

Reprsentation of the graph
There are 2 way to store the graph

1) Adjacency matrix
2) Adjacency list -->> Can be done by slice or linkedlist

Adjacency matrix   Adjacency list

When we can use the adjacency matric?
When a graph contains a large number of edges then it is good to store it as a matrix. Because only some of the entries in the matrix will be empty.
Algorithms like Prim's and Dijkstra adjacency matrix is used to have less complexity.

Action	Adjacency Matrix	Adjacency list
Adding Edge	O(1)	O(1)
Removing edges	O(1)	O(N)
Initialising	O(N*N)	O(N)
2 Nodes conn or not	O(1)	O(N)

Basic operations on the graph
1) Insertion of the edges and nodes in the graph
2) Deletion
3) Searching
4) Traversal

1) NULL Graph :- A graph is said to be a null if there is no edges.

2) Trivial Graph :- A graph is only having a single vertex then it is called as trivial. It is also a smallest possible graph.

3) Un-directed graph :- A graph in which edges do not have any direction is called as undirected graph.
All unidirected edges
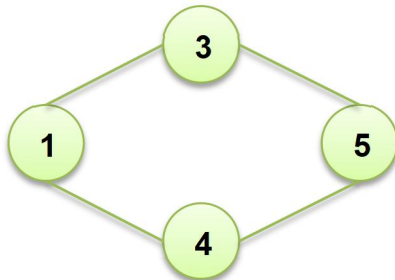
4) Directed graph :- A graph in which edge has direction.That is the nodes are ordered pairs in the definition of every edge.
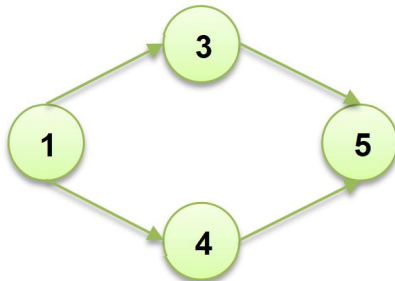
5) Connected Graph :- A graph in which from one node we can vist any other node in the graph is known as connected graph.

6) Disconnected graph :- The graph in which atleast one node is not reachable from and node is known as disconnected graph.

7) Regular Graph :- A graph in which the deegree of every vertex is equal to K then it is called as K regular graph.
2 Regular graph

Referances
https://www.geeksforgeeks.org/introduction-to-graphs-data-structure-and-algorithm-tutorials/

Write a program to create the graph.
or
Add a vertex and edges in graph using third party lib tool.
or
Write a program to insert nodes and edges using adjacenc list
or
graph using the array.
or
example of directed graph

```go
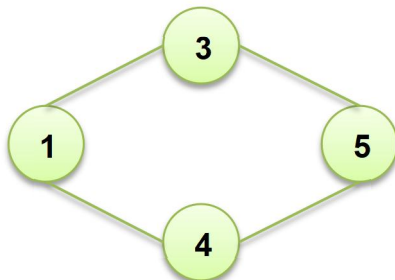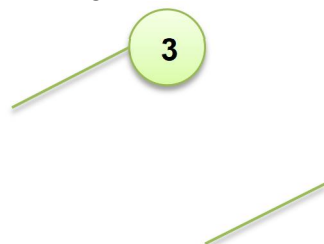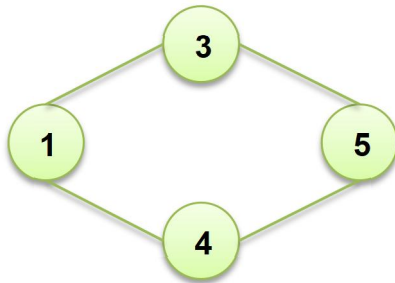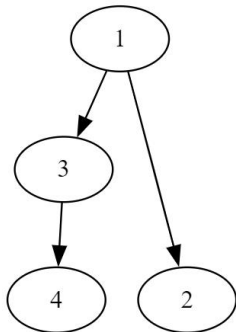package main

import (
    "os"
    "github.com/dominikbraun/graph"
    "github.com/dominikbraun/graph/draw"
)
func main() {
    g := graph.New(graph.IntHash, graph.Directed())
    _ = g.AddVertex(1)
    _ = g.AddVertex(2)
    _ = g.AddVertex(3)
    _ = g.AddVertex(4)
    _ = g.AddEdge(1, 2)
    _ = g.AddEdge(1, 3)
    _ = g.AddEdge(3, 4)
    //  _ = g.AddEdge(1, 2, graph.EdgeAttribute("color", "red"))
    file, _ := os.Create("my-graph.gv")
    _ = draw.DOT(g, file)
}
```

```
/*
go run grapdirectedhthirdpartypackageadjacencylist.go
*/
Install the Graphviz Interactive Preview to see the graph in visual studion image will looks like below.
```

Write a program to add the edges, vertex, search the vertex in golang?
or
graph using the array/adjacency list print, add vertex, add edge, find vertex etc.
or
write a program for graph using struct and method
or
example of directed graph

```go
package main

import "fmt"
// Graph structure
type Graph struct {
    vertices []*Vertex
}
// Adjacent Vertex
type Vertex struct {
    key      int
    adjacent []*Vertex
}
// AddVertext will add a vertex to a graph
func (g *Graph) AddVertex(vertex int) error {
    if contains(g.vertices, vertex) {
        err := fmt.Errorf("Vertex %d already exists", vertex)
        return err
    } else {
        v := &Vertex{
            key: vertex,
        }
        g.vertices = append(g.vertices, v)
    }
    return nil
}
// AddEdge will add ad endge from a vertex to a vertex
func (g *Graph) AddEdge(to, from int) error {
```

```go
        toVertex := g.getVertex(to)
        fromVertex := g.getVertex(from)
        if toVertex == nil || fromVertex == nil {
            return fmt.Errorf("Not a valid edge from %d ---> %d", from, to)
        } else if contains(fromVertex.adjacent, toVertex.key) {
            return fmt.Errorf("Edge from vertex %d ---> %d already exists", fromVertex.key, toVertex.key)
        } else {
            fromVertex.adjacent = append(fromVertex.adjacent, toVertex)
            return nil
        }
}
// getVertex will return a vertex point if exists or return nil
func (g *Graph) getVertex(vertex int) *Vertex {
    for i, v := range g.vertices {
        if v.key == vertex {
            return g.vertices[i]
        }
    }
    return nil
}
func contains(v []*Vertex, key int) bool {
    for _, v := range v {
        if v.key == key {
            return true
        }
    }
    return false
}
func (g *Graph) Print() {
    for _, v := range g.vertices {
        fmt.Printf("%d : ", v.key)
        for _, v := range v.adjacent {
            fmt.Printf("%d ", v.key)
        }
        fmt.Println()
    }
}
// func PrintEgDirectedGraph() {
func main() {
    g := &Graph{}
    g.AddVertex(1)
    g.AddVertex(2)
    g.AddVertex(3)
    g.AddEdge(1, 2)
    g.AddEdge(2, 3)
    g.AddEdge(1, 3)
    g.AddEdge(3, 1)
    g.Print()
    fmt.Println(g.getVertex(4))
}
/*
o run graphdirectedprintaddvertexedgesearchvertexadjacencylist.go
1 : 3
2 : 1
3 : 2 1
<nil>
```

Which is a good package for directed and undirected graph ploting?
https://pkg.go.dev/gonum.org/v1/gonum/graph#Directed
https://github.com/gonum/graph/blob/master/simple/undirected.go

Trie data structor
Trie is a k-ary search tree used for storing and searching a specific key from a set.
Using trie search complexity can be brought to optimal limit (key length).
Root node is always NULL
Each child of nodes is sorted alphabetically
Each node having maximum of 26 childrens
Each node exacept the root node store one letter of alphabets
R = NULL

Below keywords can be creed by using the above string
and
ant
dad
do

Write a program to insert word in trie using method?
or
Trie insert delete word

```go
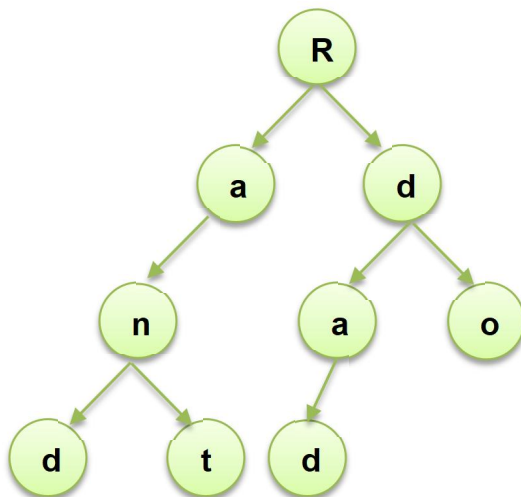package main

import "fmt"
//Declaring trie_Node  for creating node in a trie
type trie_Node struct {
    //assigning limit of 26 for child nodes
    childrens [26]*trie_Node
    //declaring a bool variable to check the word end.
    wordEnds bool
```

```go
}
//Initializing the root of the trie
type trie struct {
    root *trie_Node
}
//inititlaizing a new trie
func trieData() *trie {
    t := new(trie)
    t.root = new(trie_Node)
    return t
}
//Passing words to trie
func (t *trie) insert(word string) {
    current := t.root
    for _, wr := range word {
        index := wr - 'a'
        if current.childrens[index] == nil {
            current.childrens[index] = new(trie_Node)
        }
        current = current.childrens[index]
    }
    current.wordEnds = true
}
//Initializing the search for word in node
func (t *trie) search(word string) int {
    current := t.root
    for _, wr := range word {
        index := wr - 'a'
        if current.childrens[index] == nil {
            return 0
        }
        current = current.childrens[index]
    }
    if current.wordEnds {
        return 1
    }
    return 0
}
//initializing the main function
func main() {
    trie := trieData()
    //Passing the words in the trie
    word := []string{"and", "ant", "dad", "do"}
    for _, wr := range word {
        trie.insert(wr)
    }
    //initializing search for the words
    words_Search := []string{"and", "ant", "dad", "do", "cat", "dog", "can"}
    for _, wr := range words_Search {
        found := trie.search(wr)
        if found == 1 {
            fmt.Printf("\"%s\"Word found in trie\n", wr)
        } else {
            fmt.Printf(" \"%s\" Word not found in trie\n", wr)
        }
    }
}
```

```
}
/*
go run trieinsertesearch.go
"and"Word found in trie
"ant"Word found in trie
"dad"Word found in trie
"do"Word found in trie
 "cat" Word not found in trie
 "dog" Word not found in trie
 "can" Word not found in trie
*/
```

Write a program to insert, delete, search branch/specific word

or

trie with

1) Insert string first time
2) Insert single keyword
3) search prefix
4) search keyword
5) delete keyword
6) delete branch
7) delete after specifix branch/keyword

or

Trie with insert,delete,search

```
// Go Implementation of a Thread Safe Trie Data Structure and (some of) Trie Operations
package main

import (
    "fmt"
    "sync"
)
type Node struct {
    // list of all the children
    Children      []*Node
    m             *sync.RWMutex
    childIndexMap map[rune]int
    Val           rune
    // isEndOfWord is true if the node
    // represents end of a word
    IsEndOfWord bool
}
// CreateNode returns an initialized trie node
func CreateNode(v rune) *Node {
    return &Node{
        Children:      make([]*Node, 0),
        childIndexMap: make(map[rune]int),
        Val:           v,
        m:             new(sync.RWMutex),
    }
}
// AddChildNode add child node to current node with value v
```

```go
func (n *Node) AddChildNode(v rune) *Node {
    node, exist := n.GetChildNode(v)
    if exist {
        return node
    }
    n.m.Lock()
    n.childIndexMap[v] = len(n.Children)
    n.m.Unlock()
    node = CreateNode(v)
    n.Children = append(n.Children, node)
    return node
}
// Len returns the number of children of node
func (n *Node) Len() int {
    n.m.RLock()
    l := len(n.childIndexMap)
    n.m.RUnlock()
    return l
}
// IsLeafNode returns true if current node is a leaf node in Trie
func (n *Node) IsLeafNode() bool {
    return n.Len() == 0
}
// GetChildNode retrieve child node with value v.
func (n *Node) GetChildNode(v rune) (node *Node, exist bool) {
    n.m.RLock()
    defer n.m.RUnlock()
    if i, ok := n.childIndexMap[v]; !ok {
        return nil, false
    } else {
        return n.Children[i], true
    }
}
// DeleteChildNode Deletes the child node if it exist
func (n *Node) DeleteChildNode(v rune) {
    n.m.Lock()
    defer n.m.Unlock()
    n.Children[n.childIndexMap[v]] = nil
    delete(n.childIndexMap, v)
}
// Trie Represent a node of trie
// Not advised to create the node directly. Use helper function CreateNode() instead
type Trie struct {
    Node
}
// New Creates an initialized trie data structure
func New() *Trie {
    return &Trie{
        *CreateNode(0),
    }
}
// Insert allow one or more keyword to be inserted in trie
// keyword can be any unicode string
func (t *Trie) Insert(keywords ...string) *Trie {
    for _, v := range keywords {
        t.insert(v)
```

```go
    }
    return t
}
func (t *Trie) insert(keyword string) {
    node := &t.Node
    for _, v := range []rune(keyword) {
        node = node.AddChildNode(v)
    }
    node.IsEndOfWord = true
}
// PrefixSearch checks if keyword exist in trie as a keyword or prefix to a keyword
func (t *Trie) PrefixSearch(key string) (found bool) {
    node := &t.Node
    for _, v := range []rune(key) {
        if n, ok := node.GetChildNode(v); ok {
            node = n
            found = ok
            continue
        }
        return false
    }
    return found
}
// Search checks if keyword exist in trie as a fully qualified keyword.
func (t *Trie) Search(keyword string) (found bool) {
    node := &t.Node
    for _, v := range []rune(keyword) {
        if n, ok := node.GetChildNode(v); ok {
            node = n
            found = ok
            continue
        }
        return false
    }
    return found && node.IsEndOfWord
}
// Delete deletes a keyword from a trie if keyword exist in trie
func (t *Trie) Delete(keyword string) {
    node := &t.Node
    var breakNode *Node
    var breakRune rune
    for _, v := range []rune(keyword) {
        if n, ok := node.GetChildNode(v); ok {
            if node.IsEndOfWord {
                breakNode = node
                breakRune = v
            }
            node = n
            continue
        }
        return
    }
    if !node.IsEndOfWord {
        return
    }
    if !node.IsLeafNode() {
```

```go
            node.IsEndOfWord = false
            return
        }
        if breakNode == nil {
            breakNode = &t.Node
            breakRune = []rune(keyword)[0]
        }
        breakNode.DeleteChildNode(breakRune)
}
// DeleteBranch deletes all child after last letter of key if key exists in trie
// If key is found, key will be treated as a keyword after this operation
func (t *Trie) DeleteBranch(key string) {
    node := &t.Node
    for _, v := range []rune(key) {
        if n, ok := node.GetChildNode(v); ok {
            node = n
            continue
        }
        return
    }
    node.childIndexMap = make(map[rune]int)
    node.IsEndOfWord = true
    node.Children = make([]*Node, 0)
}
func main() {
    trie := New().Insert("foo", "bar", "baz")
    trie.insert("food")
    fmt.Println(trie.PrefixSearch("food")) // returns true since we added food
    fmt.Println(trie.PrefixSearch("fo"))   // returns true
    fmt.Println(trie.PrefixSearch("fb"))   // returns false
    fmt.Println(trie.Search("fo"))         // returns false
    fmt.Println(trie.Search("foo"))        // returns true
    trie.Delete("foo")
    fmt.Println(trie.Search("foo")) // returns false because we delete the foo
    trie.DeleteBranch("ba")         // Delete bar and baz since we added ba then ba and after all items
deleted
    fmt.Println(trie.Search("bar")) // returns false because we delete the bar
    fmt.Println(trie.Search("baz")) // returns false because we delete the baz
    fmt.Println(trie.Search("ba"))  // returns true because ba is present
}
/*
go run trieinsertsearchdeletebranchchildnodeprefixsearch.go
true
true
false
false
true
false
false
false
true
*/
```

Hashing algorithm

or

Hash algorithm

Algorithem which takes input a message of arbitory length and produces as output a "fingerprint" of the original messages.

Common hashing algorithems

MD5   128 bits

SHA/SHA1   160 bits

SHA2 Family

        SHA224 224 bits

        SHA256 256 bits

        SHA384 384 bits

        SHA512 512 bits

hello   -->>   Hashing Algorithem -->> 52

            8 + 5 + 12 + 12 + 5

cello   -->>   Hashing Algorithem -->> 47

            3 + 5 + 12 + 12 + 5

If the original data is changed, resulting digest will be different.

Result of hashing is also called as Digest, checksum, fingerprint, has, CRC etc

Above example is just for theory puepose. Real life hashing algo has 4 requirements to meet

1) From the digest original algorithm must not be creatable

2) From the message original message must not be creatable (One way encryption)

3) Slight change in message will produce very large difference in digest

4) Resulting digest must produced same length

```go
package main

import (
    "crypto/md5"
    "encoding/hex"
    "fmt"
)
func GetMD5Hash(text string) string {
    hasher := md5.New()
    hasher.Write([]byte(text))
    return hex.EncodeToString(hasher.Sum(nil))
}
func main() {
    fmt.Println(GetMD5Hash("hello"))                                    // Fist and second requirement
imposible to extrac has function and hash message
    fmt.Println(GetMD5Hash("cello"))                                    // Third requirement Small
change in message will produce drastic differenece
    fmt.Println(GetMD5Hash("The quick brown fox jumped over the lazy dog")) // Fourth requirement
digest/ouput have same length
    //Wether you take one string, 10 string, full book string the length will be same
}
/*
go run hashingalgorithem.go
```

```
5d41402abc4b2a76b9719d911017c592
711d6d0272f84193afc991a50492f57a
08a008a01d498c404b0c30852b39d3b8
*/
```

Collision in hashing algorithems.

Collision occures when 2 different message will produce same digest.

Collision can not be avoided due to below reasons

It is a by product of fixed width diest

https://www.mscs.dal.ca/~selinger/md5collision/

```go
package main

import (
    "crypto/md5"
    "fmt"
    "io"
    "os"
)
func main() {
    file1, err := os.Open("erase.exe")
    if err != nil {
        panic(err)
    }
    defer file1.Close()
    hash1 := md5.New()
    _, err = io.Copy(hash1, file1)
    if err != nil {
        panic(err)
    }
    fmt.Printf("%s MD5 checksum is %x \n", file1.Name(), hash1.Sum(nil))
    file2, err := os.Open("hello.exe")
    if err != nil {
        panic(err)
    }
    defer file2.Close()
    hash2 := md5.New()
    _, err = io.Copy(hash2, file2)
    if err != nil {
        panic(err)
    }
    fmt.Printf("%s MD5 checksum is %x \n", file2.Name(), hash2.Sum(nil))
}
/*
go run hashinglgorithemcollisionfile.go
erase.exe MD5 checksum is cdc47d670159eef60916ca03a9d4a007
hello.exe MD5 checksum is cdc47d670159eef60916ca03a9d4a007
// Download the files from the https://www.mscs.dal.ca/~selinger/md5collision/
*/
```

What are the 4 concept of the object oriented programming?
1) abstraction
2) Encapsulation

3) Inheritance
4) Polymorphism

Encapsulation :- Wrapping up of the data under single unit.
It is the mechanism that bind the together the code and data at manipulate.
Go don't have a classes and objects. Normally in object oriented programming the variable or data of a class are hidden from any other class and can be accessed by any function of own class which they are declared.
Golang support **package** through which **encapsulation** can be acceded.
If you write **function** in **lower case** in **package** then it is **not accessed** by other package which is un-exported identifier.
If you write function in first letter capital then it can be accessed by other package which is called exported identifiers.

Benefits of Encapsulation:

- Hiding implementation details from the user.
- Increase the reusability of the code.
- It prevents users from setting the function's variables arbitrarily. It only sets by the function in the same package and the author of that package ensure that the function maintain their internal invariants.

GIN vs Gorillamux API
Api with gin is really easier then gorilla mux. Refer below where there are 2 functions getgophers and creategophers
Line of code

GIN		Gorilla	
getgophers	creategophers	Getgophers	creategophers
1	7	6	11

API with GIN

```
package main

import (
    "log"
    "net/http"
    "github.com/gin-gonic/gin"
)
type Gopher struct {
    ID        string `json:"id"`
    FirstName string `json:"firstname"`
    LastName  string `json:"lastname"`
}
var gophers = []Gopher{
    {"1", "Ken", "Thompson"},
```

```go
        {"2", "Robert", "Griesemer"},
}
func main() {
    router := gin.Default()
    router.GET("/gopher", getGophers)
    router.POST("/gopher", createGopher)
    err := router.Run("localhost:8080")
    if err != nil {
        log.Fatal(err)
    }
}
/*
c.IndentedJSON(http.StatusOK, gopher)
    -automatically sets content type to application/json
    -uses passed in status code
    -serializes given struct as pretty JSON
*/
func getGophers(c *gin.Context) {
    c.IndentedJSON(http.StatusOK, gophers)
}
/*
c.BindJSON(&newGohper)
    -binds JSON to struct pointer
    -we handle any errors
*/
func createGopher(c *gin.Context) {
    var newGohper Gopher
    err := c.BindJSON(&newGohper)
    if err != nil {
        c.String(http.StatusBadRequest, err.Error())
        return
    }
    gophers = append(gophers, newGohper)
    c.IndentedJSON(http.StatusCreated, gophers)
}
/*
    // BindJSON is a shortcut for c.MustBindWith(obj, binding.JSON).
    // MustBindWith binds the passed struct pointer using the specified binding engine.
    // It will abort the request with HTTP 400 if any error occurs. See the binding package.
    // func (c *Context) BindJSON(obj interface{}) error
*/
/*
Use thunder bolt or postman
http://localhost:8080/gopher
get meyhod will return 2 structs
post method
go to json
{
    "id": "3",
    "firstname": "Rob",
    "lastname": "Pike"
}
Above post method will return 3 JSON
*/
```

API with gorillamux

```go
package main

import (
    "encoding/json"
    "log"
    "net/http"
    "github.com/gorilla/mux"
)
type gopher struct {
    ID        string `json:"id"`
    FirstName string `json:"firstname"`
    LastName  string `json:"lastname"`
}
var gophers = []gopher{
    {"1", "Ken", "Thompson"},
    {"2", "Robert", "Griesemer"},
}
func main() {
    router := mux.NewRouter()
    router.HandleFunc("/gopher", getGopher).Methods("GET")
    router.HandleFunc("/gopher", createGopher).Methods("POST")
    http.Handle("/", router)
    err := http.ListenAndServe(":8080", router)
    if err != nil {
        log.Fatal(err)
    }
}
/*
w.Header.Set() sets the content type to application/json
if w.WriteHeader is no called explicitly, implicit http.StatusOK is used with w.Write()
jsonGohper, err := json.Marshal(gopher) converts the gopher struct to JSON
if err != nil {} sends the 400 error code if the marshaling fails and ends the func
w.Writer()  writes the data to the connection as part of an HTTP reply
*/
func getGopher(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")
    jsonGohper, err := json.Marshal(gophers)
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
    w.Write(jsonGohper)
}
/*
A Decoder reads and decodes JSON values from an input stream.
NewDecoder returns a new decoder that reads from r.
The decoder introduces its own buffering and may read data from r beyond the JSON values requested.
Decode reads the next JSON-encoded value from its input and stores it in the value pointed to by v.
handle any errors create decoder, setting HTTP status code
since err from defer r.Body.Close() is not nill, closing r.Body must be explicitly closed
json.Marshal(gophers) marshals gophers slice so w.Write() can recieve it
w.Header().Set() sets the headers to application/json
w.Write() writes our JSON to the http.ResponseWriter
*/
func createGopher(w http.ResponseWriter, r *http.Request) {
```

```go
    var newGohper gopher
    decoder := json.NewDecoder(r.Body)
    err := decoder.Decode(&newGohper)
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
    defer r.Body.Close()
    gophers = append(gophers, newGohper)
    GophersJSON, _ := json.Marshal(gophers)
    w.Header().Set("Content-Type", "application/json")
    w.Write(GophersJSON)
}
/*
/*
Use thunder bolt or postman
http://localhost:8080/gopher
get meyhod will return 2 structs
post method
go to json
{
    "id": "3",
    "firstname": "Rob",
    "lastname": "Pike"
}
Above post method will return 3 JSON
*/
*/
```

How to cancel the go routine?
or
How to interrupt go routine?
or
how to stop co routines?
In Go, there are several ways to stop a goroutine:

Using a **context** with a cancellation function: As I described in the previous answer, you can use a context with a cancellation function to send a cancellation signal to a goroutine. The goroutine should check for the cancellation signal and exit when it receives it.

Using a channel: You can use a channel to send a message to a goroutine to tell it to stop. The goroutine can then exit when it receives the message on the channel.

Using a sync.WaitGroup: The sync.WaitGroup type allows you to wait for a group of goroutines to finish. You can use this to stop a goroutine by calling the Done method when the goroutine finishes.

Using a sync.Once: The sync.Once type allows you to ensure that a function is only called once. You can use this to stop a goroutine by calling the Do method with a function that stops the goroutine.

Using a sync.Mutex: A sync.Mutex allows you to synchronize access to a variable. You can use this to stop a goroutine by setting a flag variable and using a mutex to protect it.

Using a time.Ticker: A time.Ticker sends periodic ticks on a channel. You can use it to stop a goroutine after a certain period of time.

Using a time.Timer: A time.Timer sends a single tick on a channel after a certain period of time. You can use it to stop a goroutine after a certain period of time.

It is important to note that, regardless of the method used, stopping a goroutine does not guarantee that it will stop immediately, it's only a signal for it to stop.

Also, it is important to note that it's not possible to forcefully stop a running goroutine, because the go scheduler is responsible for scheduling goroutines and it's not designed to forcefully stop a running goroutine.

How goroutines are different from normal thread?
Light weight it takes smaller size, lower overhead
Scheduling We don't have to schedule go routine manually it is done automatically by go schedulaer
Channels To communicate with other go routine without a locking mechanisam.
Non premitive Go routines can runs untill they explicitluy yield or block by I/O