

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 vim editor

The go compiler

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

Golang

Packages and modules

Packages are go's way of organizing

Programs are written in as one or more packages

Packages are imported from the go package registry

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 import everything 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

Dependencies, 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 Beautiful 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))
```

Concatating string

```
package main

import (
    "fmt"
    "strings"
)

func main() {

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

```

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

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

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

}

```

HelloWorld

["Hello" "World"]

[48656c6c6f 576f726c64]

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 go lang so we can not use this keyword throughout the program it can only use while declaring the constant.

iota in go lang 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

```

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

```

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

```

package main
import "fmt"
// Start iota from the particular 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 allow you to define several data items of same kind

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*/
```

Defining the slice

Declare the array without specifying the size will be slice

Alternatively can create the make function too

```
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)

    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]

```
package main

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

```
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]

Function in go

```

package main

import (

    "fmt"

)

```

```

func main() {

    var s string

    fmt.Println("Hello Beautifull world")

    s = passthestring()

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

}

```

```

func passthestring() string {

    return "Goodbye"
}

```

```
}
```

```
Hello Beautifull world
```

```
String from function is Goodbye
```

```
package main
```

```
import (
```

```
    "fmt"
```

```
)
```

```
func main() {
```

```
    //var s string
```

```
    //var t string
```

```
    fmt.Println("Hello Beautifull world")
```

```
    s, t := passthestring()
```

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

```
}
```

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

```
    return "Goodbye", "World"
```

```
}
```

Hello Beautifull world

String from function is Goodbye World

Function as a pointer

```
package main
```

```
import (
```

```
    "fmt"
```

```
)
```

```
func main() {
```

```
    var a string
```

```
    passString(&a)
```

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

```
}
```

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

```
    *b = "name"
```

```
}
```

Value come from function is name

```
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

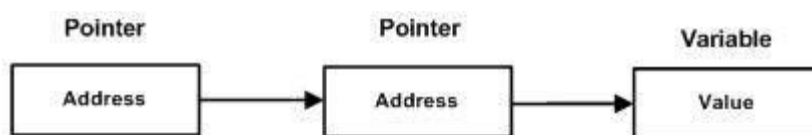
1 5

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.



```

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

```
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

```

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 –

```
package main

import "fmt"

type book struct {

    author    string

    publisher string

    revision  int

}

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;
```

```
package main

import "fmt"

type book struct {

    author    string

    publisher string

    revision  int
}

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 interference 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 be used in Go to describe blanks, tabs, new line characters and comments etc.

Line containing only white spaces possibly with a comment is known as a blank line

```
var age int;
```

```
fruit = apples + oranges; //Get the total fruits
```

No white spaces are 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, struct, 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)

```
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 structure 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

```
package main

import "fmt"

func main() {
    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

```

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

```

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

```

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

```

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
address 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.

```

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    // illegal 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

```
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;

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

```
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

```
//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\forasdownhile> go run forasdownhile.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

```

package main

import "fmt"

func main() {
    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

```
/*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

```

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 karnataka is banglore

capital city of maharashtra is mumbai

capital city of gujarat is ahmedabad

capital city of karnataka is banglore

capital city of maharashtra 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

```
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

```

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 syntax 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

```
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 regexprsimple.go
true <nil>
false <nil>
*/
```

Replacing

```
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)

```

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 regexppfindindex.go
[0 5]
[[0 5] [10 15]]
*/

```

Find string first and last character

```

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 regexppfindstring.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.

```
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.

```
package main

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. Methods will not work in the different package then where defined. If you try to do that then compiler will give error.

For example –

```

package main

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 provides pretty simple error handling framework.

With inbuilt error handling type of following declaration

In the 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)

```

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

```

func recursion(){
    recursion()
}

```

```

func main(){
    recursion()
}

```

Use of factorial in go with recursion

```

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

```

package main

import "fmt"

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

```

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.

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  
}
```

```
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

Go unit testing

Command to test the unit tests for function

```
go test
```

If needed all tests result then

```
go test -v
```

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 necessary is for TDD test driven development

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

Go unit testing with multiple functions

```
//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

```
// 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
    y := 5
    get := Sub(x, y)
    want := 60
    if get != want {
```

```

        t.Errorf("get %d, want %d", get, want)
    }
}

/*
To display whether 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

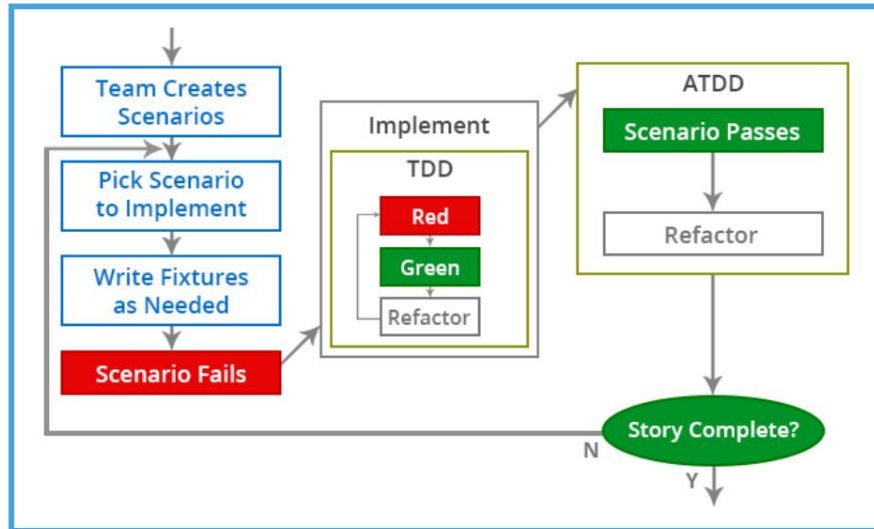
In TDD we will write a developer will write the test cases first for enhancement of new features before writing code

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 http test
- Go test example functions
- Go table driven tests

Go testing reference

<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 analysis 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 Address
- 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
```

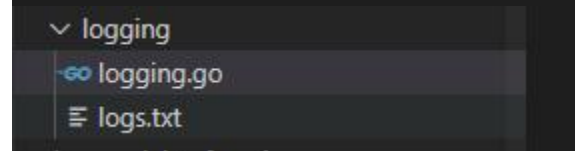
Logging to a file

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

```
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")
}
```



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
<https://download.microsoft.com/download/c/7/c/c7ca93fc-3770-4e4a-8a13-1868cb309166/SSMS-Setup-ENU.exe>

```
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()
}
```

Connecon 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

```
package main

import (
    "fmt"

    "mymodule/mypackage"
)

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

    mypackage.PrintHello()
}
```

```
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 apply to all
right hand 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 add // 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 transitively 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 errors in 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

```
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
```

```
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 convenient 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
43 // Environ provides an environment when a binary, such as the go tool, is
44 // invoked.
45 var Environ func() []string = os.Environ
```

undclared name: websocket

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
71     // JSON request id can be any JSON value;
```

field ServiceMethod 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
71     // JSON request id can be any JSON value;
```

func (*encoding/json.Decoder).Decode(v interface{ }) error

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
152     ArgType reflect.Type
153     ReplyType reflect.Type
154     numCalls uint
155 }
156
157

```

```

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 necessary 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 article.

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()})
```

```
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()})
```

```
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"})
```



```
}
```

```
// 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
```

```
}
```

```
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:

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

title: 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.

```

// 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 languages
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 dynamodb with image size, name, date etc

Things need to remember while using the lambda,

Memory size -->> 128 MB to 10240 MB

Timeout -->> 15 Minutes

Hands on with lambda,

There are 2 methods by which we can add code to lambda functions.

How you add a program/function in lambda,

Build and compile the code for linux machine

Zip it and upload into source code section

Note that if used python or similar languages then it must be directly compilable on AWS portal. But go lang will not be supported to compile directly on the portal. So we need to make machine executable binaries and then need to send to the lambda.

To test the lambda hello world 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

```
package main

import (
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
)

type Request struct {
    ID    float64 `json:"id"`
    Value string  `json:"value"`
}
```

```

}
type Response struct {
    Message string `json:"message"`
    ok      bool   `json:"ok"`
}
func Handler(request Request) (Response, error) {
    return Response{
        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 response on AWS console portal
*/

```

Amazon EC2

There is virtual server in cloud

Limited by RAM and CPU

Continuously running

You have to pay even if there is no load

Go Concurrency

In a normal situation code is executed line by line, one line at a time.

Concurrency allows multiple lines to be executed.

There are 2 types of concurrent code

1) Threaded :- The code will run in parallel based on number of CPU cores.

2) Asynchronous :- Code can be paused 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 routines and channels are lightweight built-in features for managing concurrency and communication between several functions executing at the same time.

This way one can execute the code that is outside of the main program.

Go has below keywords like

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 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 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

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
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)
        }
    }
}
```

Size of the channel

Directional and unidirectional channel

Concurrency with go routine

```
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 create go routine for executing timesThree function

There for fmt.Println("Done!") will execute before go routine

But, what if we need some value returning from that function to continue with our main function.

That's where channel comes and save the day.

```
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

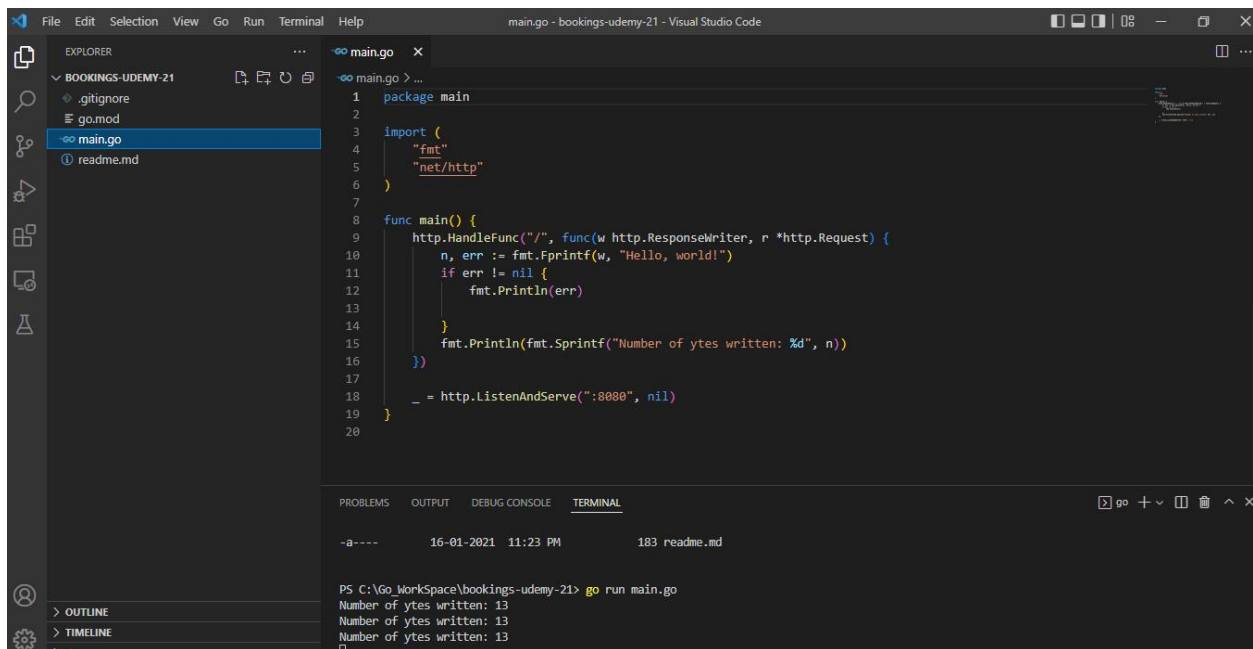
9

The result is: 9

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.

Developing a webservice

- 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



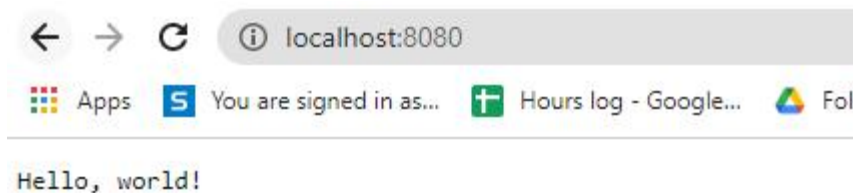
The screenshot shows the Visual Studio Code editor with a Go file named `main.go` open. The code is a simple HTTP server that listens on port 8080 and responds with "Hello, world!". The terminal at the bottom shows the command `go run main.go` being executed, and the output displays "Hello, world!" and the number of bytes written (13).

```
1 package main
2
3 import (
4     "fmt"
5     "net/http"
6 )
7
8 func main() {
9     http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
10         n, err := fmt.Fprintf(w, "Hello, world!")
11         if err != nil {
12             fmt.Println(err)
13         }
14         fmt.Println(fmt.Sprintf("Number of bytes written: %d", n))
15     })
16     _ = http.ListenAndServe(":8080", nil)
17 }
18
19
20
```

Terminal Output:

```
PS C:\Go\workspace\bookings-udemy-21> go run main.go
Hello, world!
Number of bytes written: 13
Number of bytes written: 13
Number of bytes written: 13
```

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~naWyZpuvFibWMU84GutwSxxlxpoDOg~EWY8lp~1l5ng6fJTHGKhu0kmKGdn8DJJJBDnNAIC5lfoGvVtsbg9VLnoHMPL24~56EEPjkqnNFteKOtm-GvSVaQz7lQZPwJQVnmCDRzv1oe0VaAozR4iKD1JQWgXYRER20ERT~50-5PycL73A~bj2rVh9qMVAOBqaFU1vLwaLTGguDm4LVV-jXSDI0blymO7mtRy4nW5QLOFC8gblmHhoOIL~UObnxe8o20uHtLhg__&Key-Pair-Id=APKAITJV77WS5ZT7262A

```
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)
}

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`
    _, 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

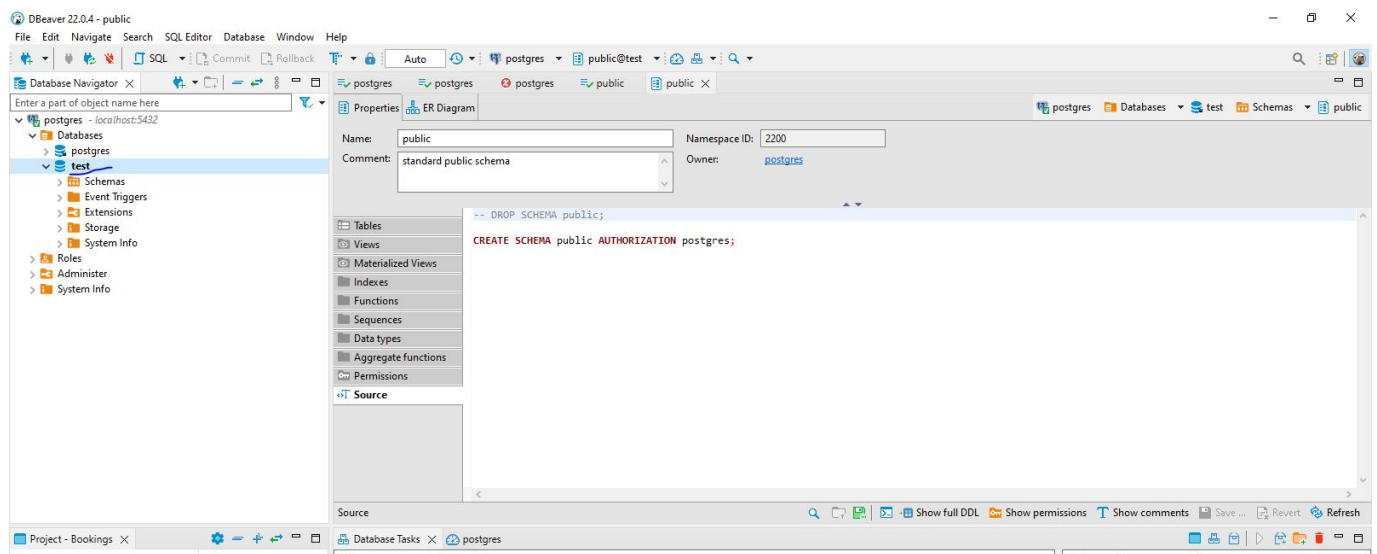
for rows.Next() {
    err := rows.Scan(&id, &firstName, &lastName)
    if err != nil {
        log.Println(err)
        return err
    }
    fmt.Println("Record is", id, firstName, lastName)
}

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

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

return nil
}

```



```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
Record is 4 Jackie Brown
Record is 6 Jack Brown
Record is 7 Jack Brown
Record is 8 Jack Brown
Record is 5 Jackie Brown
Record is 9 Jack Brown

2020/11/17 14:18:27 Updated one or more rows
Record is 1 John Smith
Record is 2 Mary Jones
Record is 3 Jackie Brown
Record is 4 Jackie Brown
Record is 6 Jack Brown
Record is 7 Jack Brown
Record is 8 Jack Brown
Record is 9 Jack Brown
Record is 5 Jackie Brown

2020/11/17 14:18:27 QueryRow returns 1 John Smith
2020/11/17 14:18:27 Deleted a row!
Record is 1 John Smith
Record is 2 Mary Jones
Record is 3 Jackie Brown
Record is 4 Jackie Brown
Record is 7 Jack Brown
Record is 8 Jack Brown
Record is 9 Jack Brown
Record is 5 Jackie Brown

(base) tcs@grendel test_connect %
```

Build webaapplication

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

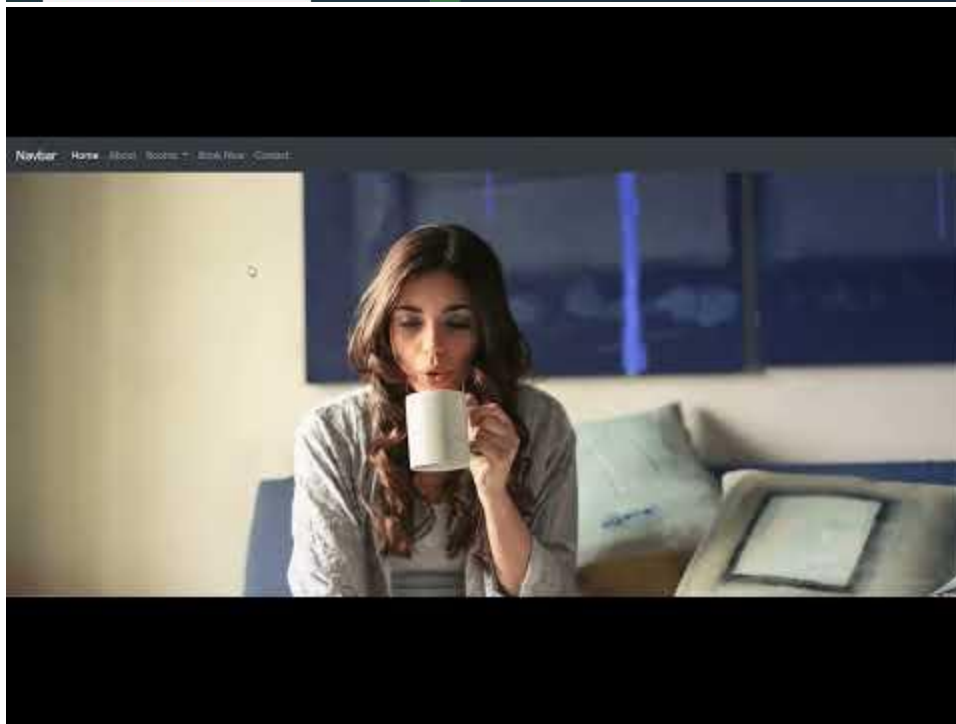
<http://localhost:8080/>

The screenshot shows the Visual Studio Code interface with a Go project named 'bookings-app-0.0.29'. The Explorer sidebar on the left lists files including 'cmd', 'web', and various test files. The main editor displays the 'main.go' file with the following code:

```
cmd > web > main.go > main
17 const portNumber = ":8080"
18
19 var app config.AppConfig
20 var session *scs.SessionManager
21
22 // main is the main function
23 func main() {
24     err := run()
25     if err != nil {
26         log.Fatal(err)
27     }
28
29     fmt.Println(fmt.Sprintf("Starting application on port %s", portNumber))
30
31     srv := &http.Server{
32         Addr:    portNumber,
33         Handler: routes(app),
34     }
35
36     err = srv.ListenAndServe()
37     if err != nil {
38         log.Fatal(err)
39     }
40 }
41
42 func run() error {
43     // what am I going to put in the session
44     gob.Register(models.Reservation{})
45 }
```

The terminal at the bottom shows the following output:

```
Starting: C:\Users\yagnik.pokal.VOLANSYS1\go\bin\dlv.exe dap --check-go-version=false --listen-127.0.0.1:53550 from d:\Go\Web\bookings-app-0.0.29\cmd\web
DAP server listening at: 127.0.0.1:53550
Type 'dlv help' for list of commands.
Starting application on port :8080
```



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

Question and answers
Mention the advantages of go lang?

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 language?
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.

`t := I.(T)`

I is the interface value

T is concret 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 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?
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

```
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

```

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?

What is use of the defer keyword?

What is rest api? Advantages of rest api?

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?

```
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?

```
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 salary of the employees?

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 if 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

While language 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 needs to be communicate with each other and,

Mutex when only one go routine should be access the critical section of the code.

```
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?

```
package main

import (
    "fmt"
    "sync"
)

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
go run mutex.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?

```
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 channel.go
1000

// Check the race detection
go run -race channel.go
1000
*/
```

Referances

<https://golangbot.com/mutex/>

What is reflection go?

How can we swap variables in 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?

```

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)
}

/* Output
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]
*/

```

How many ways are there to print the the repeating element in the string?

What is protobuf?

protobuf is the googles language neutrals, platform neutral, 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++. However proto3 version also supports kotlin, 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 takes input as a struct and data can be transferable in a serialized manner forward and backward compatibility.
2. Protocol buffers can be extended with the new information without invalidating existing data or requiring 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 comes to transfer speed of more than few mbps then it will not work.
5. What are the steps to use the protobuf with go?
 - Define the message formats 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 support inheritance?

Go doesn't support 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 recommended?

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 syntax 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?

```
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 factorial of the given number?

```

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.

```

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 whether character is present or not in given string.

```
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 useful?

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

```
package main

import "fmt"

func main() {
    fmt.Println("Begning")
    defer fmt.Println("One")
    defer fmt.Println("Two")
    defer fmt.Println("Three")
    fmt.Println("End")
}

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

Write a program whether the parenthesis is valid or not?

Write a program whether the string is palindrome or not?

```
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")
    }
}

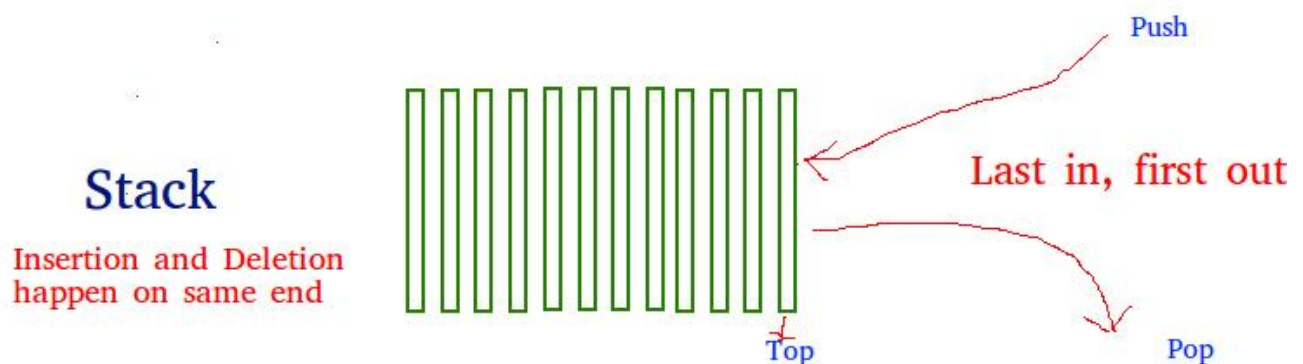
/*
go run palindrome.go
Not Palindrome
*/
```

Data structure and algorithms

Stack

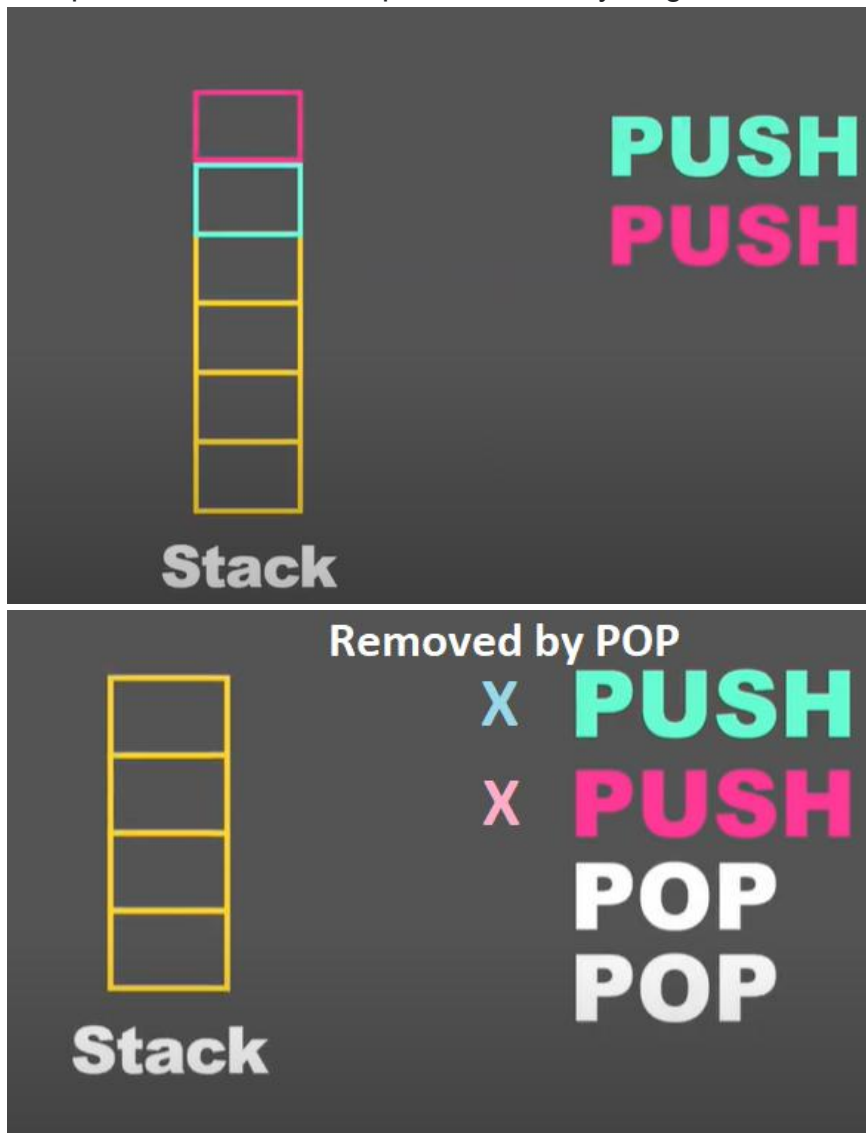
Stack is a linear data structure which follows the particular order in which operations are performed.

Stacks are variable size you can add 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 that add 3 interger and remove 2 integer in stack data structor

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

```

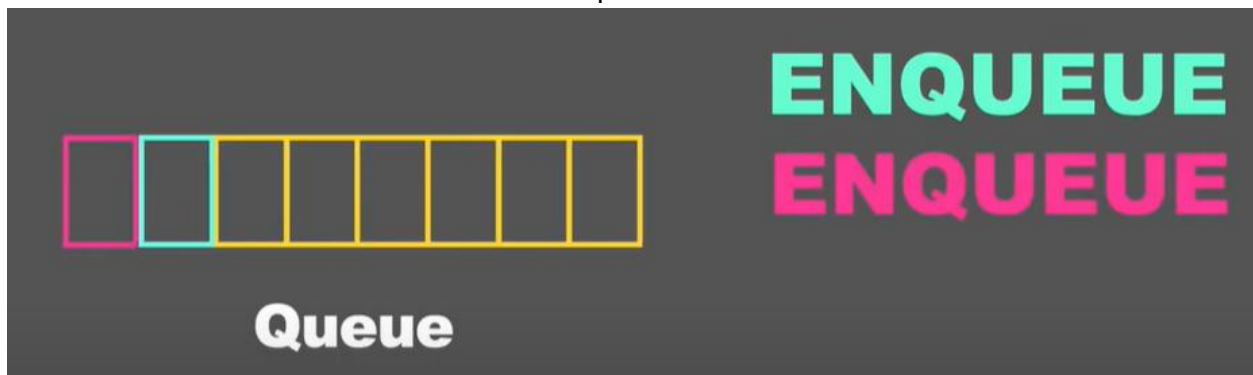
Queue

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.



```
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]}
*/
```

Referances

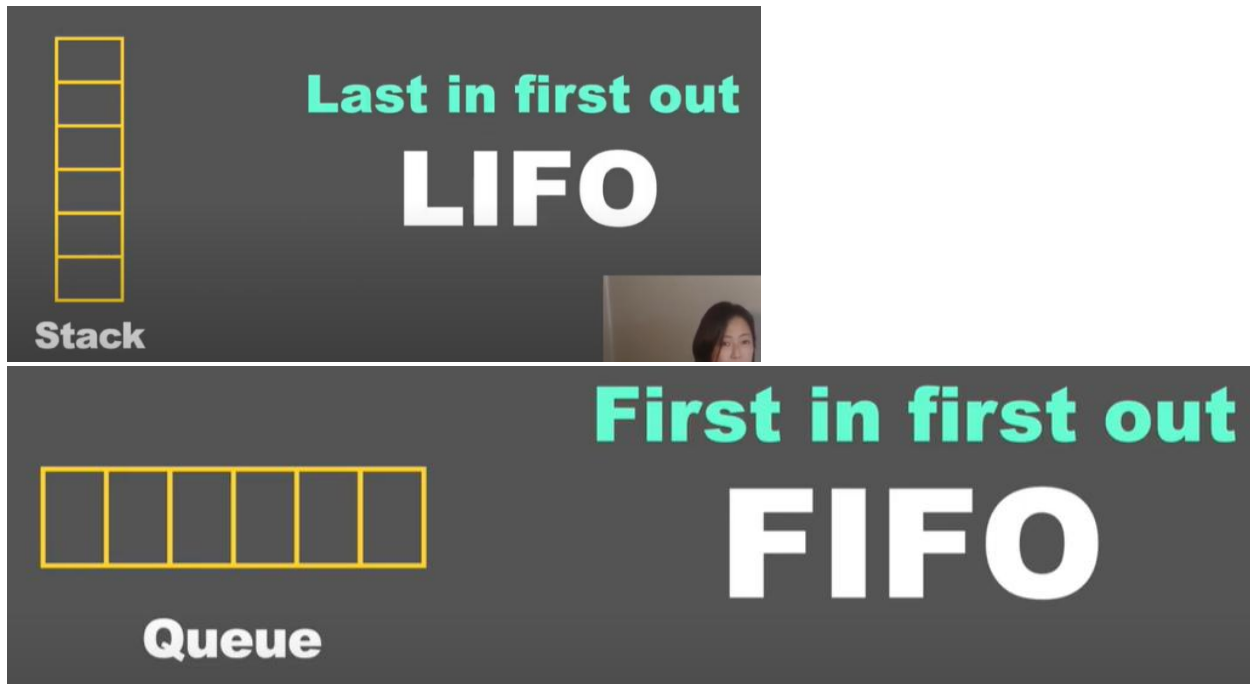
<https://www.youtube.com/watch?v=fsbm1FOSDJ0>

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 contiguous manner.

The elements in the linked list are linked using pointer.

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

Linked list is made up of 2 things

- 1) nodes (Data)
- 2) Reference the next field (Address of next element)
- 3) Needed the detail of the first head
- 4) Length of linked list

Linked list.



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

or

Linked list insertion

```
package main

import "fmt"
// Contains the data and address of next node
type Node struct {
    data int
    next *Node
}
// LinkedList contains head address 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++
}
// We can not print all the linked list without the function
// We can print the single address 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--
    }
}
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(node3)
myList.printListData()
}

/*
go run linkedlist.go
16 18 48
*/

```

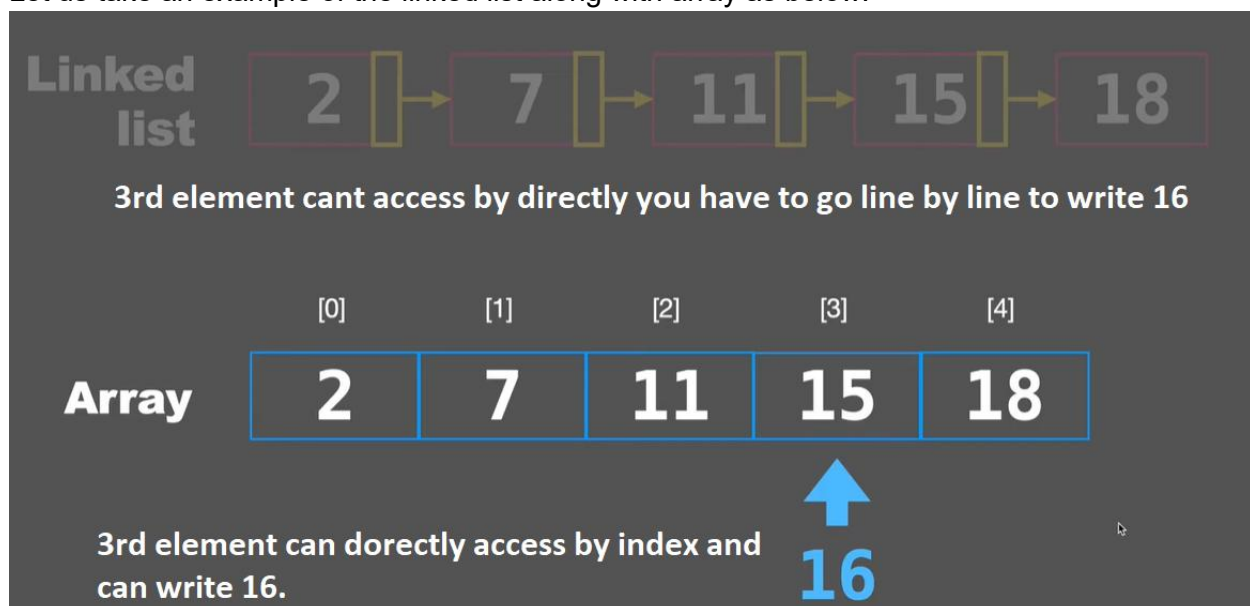
Reference

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

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.

Add and removing values at the beginning of the list



Faster $O(1)$

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.

Array



Doubly linked list

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

Doubly linked list



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

Doubly linked list



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

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