Course 1 :Getting started with GO

-Advantages of GO:

1) code runs fast

2) **garbage collection**

3) simpler objects

4) concurrency is efficient

-compiled vs interpreted

compilation: translated to machine code **once** before running the code

interpretation: code is translated while executed line be line which makes you slow

java is compiled to generate byte code (not machine code) which is interpreted at runtime

- compiled code is fast,interpreter manages memory automatically and infer variables type

- go is a compiled language with garbage collection

- **go does not use the term class, go uses structs with associated methods ,no inheritance no constructor no generics which makes it easy to code and efficient to run**

- morees law that number of transistors doubles every 18 months and more transistor leads to higher clock freqs but because of power/temperature constrains that limits the clock freqs we went to parallelism (increasing number of cores) so you can perform multiple tasks at the same time

-difficult with concurrency:

→ when do tasks start/stop?

→ what if task need data from another task?

→ do tasks conflict in memory?

-concurrency is the management of multiple task at the same time(alive at the same time)

- concurrent programming enables parallelism
    → management of task execution
    → communicating between tasks
    → synchronization between tasks
- **Goroutines** represents concurrent tasks
  Channels are used to communicate between tasks
  Select enables task synchronization

- workspace directory
    -src : contains source code files
    -pkg : contains packages(libraries)
    -bin : contains executables
- GOPATH is defined during installation

-Package is group of related source code and each package can be imported by other packages(package reuse)

- package “name” to define a package at the start of its file
- there must be one package called main
- building the main package generates an executable program
- main package needs a main function
- import keyword to import other package
- GOROOT and GOPATH environment variables are where the packages should be searched for and if want to import package not there you have to change the environment variables
- **go build** compiles the program
- go doc : prints documentation for a package
- go fmt : formats source code files

- go get : downloads packages and installs them

- go list : lists all installed packages

- go run : compiles .go file and runs the executable

- go test : runs tests using file ending in “_test.go”

- variable declaration:

    var x int (keyword name type) the most basic declaration

    var x,y int

- alias of type

    type Celsius float64

- initializing variable

    var x int = 100

    var x = 100 // infers the type

    var x int // uninitialized variables have a zero value 0,””,

    x = 100

    x:= 100 //performs declaration and initialization

- **new()** function creates a variable zero initialized and returns a pointer to this variable

- stack vs heap

    → stack is dedicated to function calls , local variables are stored here and deallocated after function complete

    → data on heap must be deallocated when its done being used,in most compiled languages allocation and deallocation is done manually which is fast but error prone

- garbage collection is done by java virtual machine in java and python interpreter in python, this is easy for the programmer

- go is a compiled language with gc

- compiler determines heap or stack and gc works in the background which slows the code a little bit but makes coding easier

- iota generates a set of related but distinct constants

- in switch statements you don't have to use break after each case

Week 3

Array Literal

- var x [5] int = [5] {1,2,3,4,5}

  x:= […] int{1,2,3,4,5}

for I,v **range** x {

}

- slice is a window on **underlying array,** variable size up to the whole array

- each slice has pointer which indicates the start of the slice,length number of elements in the slice, capacity is the max number of elements from start of the slice and the end of the array

arr:= […] string{“a”,”b”,”c”,”d”,”e”,”f”,”g”}

s1:=arr[1:3] // len = 2 , cap = arr.size() -1

s2:=arr[2:5]

- writing to slice changes the underlying array

sli :=[…] int{1,2,3}

- make

sli := make([]int,10) // here length = capacity

sli := make([]int,10,15) // here length = 10 , capacity = 15

- append() adds to the end of array if the capacity isnt reached, if capacity is reached will allocate new memory

-maps

var idMap map[string]int

idmap := make(map[string]int)

idMap[“joe”] = 5 // adding to map

delete(idMap,”joe”) //deletes from the map

id,p := idMap[“joe”] // p is boolean exist or not

for key,val:= range idMap {}

- struct //read from code

p1:= new(person)

- packages for standards
    - → net/http : web communication protocol (http.get(“”))
    - → net : TCP/IP and socket programming (net.Dial(“tcp”,”uci.edu:80”)
- JSON : java script object notation

    json marchal .unmarchal

-package ioutil reads the whole files and closes the handle it is good for small files but for large files you cant do that you have to read file byte by byte

- os file system