Lec 3: Go Intro
Why? C was designed in 1970 with those machines in mind. Go is C - 40 years later. Biggest change — no explicity memory management (malloc and free). Rather more java-like with new and garbage collection.

Put every different go program in a different folder.
put program files in files that end in .go

```
package main    // REQUIRED
import “fmt"    // won’t compile unless imports exactly match
uses (unlike java).  do a search for “golang package fmt”
func main() { // the function to start the program.  Should be
exactly one instance of a main function in a directory
    fmt.Println("hello geoff!”) // Do something!!!
}
```

Go has lots of packages. We will discuss this end of semester with modules and types in OO programming.

```
go list …

Variables
     lots of types :: usually you do not need to know.  Go
figures it out
     var i = 0
    var i int
     i := 0
These are all equivalent.  Go initializes all integers to 0
(second case). (All types have a “zero” value. Go figures out
that i is an int (first and third). := gives “short form”
initialization … “=“ does assignment “:=“ does initialization
and assignment

Go uses value model of variables (as does Java for primitive
types).  As does C. So like C, go has pointers and the
complexities of referencing and dereferencing pointers. Will
talk about this in ch 6. Unlike C, go has garbage collection
(more on that in ch 8.5.3)

“Tuple Assignment”
package main
import "fmt"
func main() {
    j,k := 5, 20  // initialize j and k
    fmt.Printf("j:%d  k:%d\n", j,k);
    k,j = j,k // swap j and k   uses only one line!!!
```

```
    fmt.Printf("j:%d  k:%d\n", j,k);
    l, m := mul(j,k)  // call function and initialize l and m
for return values
    fmt.Printf("l:%d   m:%f\n", l,m)
}

/**
* do something
* @param i an integer
* @param j an integer
* @return an integer and a float32
**/
func mul(i , j int) (int, float32) { // return two values
    ii := i*j;
    jj := float32(i) / float32(j);  // casting
    return ii,jj

}


if and for
no parens, must have {}
package main
import "fmt"
func main() {
    ii,f1, f2 := 0,1,1
    for { // Go does not have a while loop!   Just for with
nothing (or ;;) No Parens MUST {}
        ii++;
        f1,f2 = f2, (f1+f2)
        if f2<0 {  // no parens must {}
            break
        }
        fmt.Printf("%d  %d %d\n", ii, f1, f2)
    }
}

printf
%v    the value in a default format
      when printing structs, the plus flag (%+v) adds field names
%t    the word true or false
%d    base 10
%f    decimal point but no exponent, e.g. 123.456
%s    the uninterpreted bytes of the string or slice
\n CR-LF
```

Scope – very much like java We will discuss scope in great detail

arrays and slices
arrays – homegeneous collection with length fixed at compile time
slice – somewhat Java ArrayList
see slic.go
also with slices you can get a piece
slice[start:end]
for example see remove fun in slice_go or slisli_go

Generics and make – look a lot like Java. Generics mostly apply to libraries. In data structures you implememted a lot of libraries. In this class you will mostly use. Current Go does not have user definable generics

structs
much like java classes, with some different syntax. Structs can have methods!
speed.go
Structs do “inherit” – somewhat
- embedding (embed.go)
- static (mostly) method binding (funcbind_go/funcbind.go)
  contract with Java funcbind_go/FuncBind.java

Program across multiple files
In same directory
UNIX> mkdir AAA
UNIX cd AAA
UNIX> go mod init GGT/AAA
UNIX> go run .
VSC run button does not work.

Encapsulation and multiple directories:
Everything in a package is public to everything in the same package. In other packages, capitalization indicates public to other packages. See encap_go
Also note that fmt.Println, fmt is initial cap, hence is is public from the fmt package.