Library import

```
import …
import(
  …,
  ....
)
```

Function

```
func func_name(arg dt,arg dt) (return_dt,return_dt) {}
```

It can also support naked return if specified in return itself
Functions can be passed, returned or assigned as values

Note : If a function or variable need to exported or called from another package its name must start with Uppercase

Variable

```
i:=0
var i,j dt=value,value
var(
   I = value,
   J = value
)
var i,j = value,value
```

Pointers same as c; * for differing and & for addr

Struct

```
type StructName struct{
   var1 dt
   var2 dt
}
v1:=StructName{var1=value,var2=value}
```

Array

```
var a [size]datatype
a := [size]datatype{value(s)..}
```

slice array[low:high]; they are like reference so changes will be reflected.

Length and Capacity, len(slice) refers the actual length of of slice whereas cap(s) refers capacity underlying array's capacity from the first element of slice.

make([]dataype,len,capacity) to create slices

```
arr:=[][]datatype{
        []datatype{values},...
```

}
append(slice,values...)

Conditionals and loop statement
for decl&init;condition;update{}
for condition{} //acts as while loop
for{} //infinite loop
for i,v:=range array{//i will have iteration no and v values of array}

if assignment;condition{
}else{}

switch assignment;variable{}
Naked switch is also supported, case could contain condition; execution is top to bottom; on condition satisfied on a case it exits

Maps
Similar to dictionary in python
var variable map[dt_key]dt_value / variable:=make(map[dt_key]dt_value) /
variable:=map[dt_key]dt_value{
  key1:value1,...
}

Methods
Similar to class can also be made as arguments
func (variable datatype) func_name() return_type{} / func func_name(variable dt) return_type{}
Calling : var a datatype = value; a.func_name() / func_name(a)

Called as reference can also be done by assigning pointers in arguments;
func func_name(var *dt) rdt{}; func_name(&variable)
func (var *dt) func_name() rdt{}; variable.func_name()

Interface
Similar to class
type interface_name interface{
 method_name() return_type,..
}
func (variable data_type) func_name() retur_type{}
Calling : var variable datatype = value;var i interface=variable;i.func_name();
A variable decl to an interface can happen only if the decl method in that interface has been defined for that datatype;

Interface is defined in (value,datatype)
To access value of the interface interface_variable.(datatype)
interface_variable.(type) to access the datatype of interface variable

Golang additional features
defer : A statement to exec once before the function returns; Multiple defer will be exec in LIFO order

go routine : the function referred in go routine executes in separate thread; go function()

channels : to send msgs within a program | ch:=make(chan datatype); ch <- value //send; variable<-ch //recv; prev in unbuffered send or recv is blocked till the opposite occur
Buffered; ch:=make(chan datatype,size_of_buffer); even for loop could be used to iterate over recvd channel values; for a:=range ch{} but need to be closed to exit loop.

select : Similar to switch, but listens on channels;

```
select{
 case a<-ch:
          …
 case ch<-a:
          …
 default:
  //executes if non other are true
  ...
}//repetes till exit.
```