11.24.2022 Go Lesson Instruction

111. Unfurling a slice  
  
func main() { func sum(x ...int) int {  
xi := []int{1, 2, 3, 4, 5, 6, 7, 8, 9} //slice fmt.Println(x)  
x := sum(xi...) fmt.Printf("%T\n", x)  
fmt.Println(x) fmt.Printf("%T\n", x)  
} sum := 0  
for v := range x {  
return sum  
}  
}  
  
We have to pay attention `func sum(x ...int)` for getting data from another calling function. And there is `sum(xi...)` for sending data to another function.  
  
112. Defer  
defer is used for deferring function for dynamically. It is like lazy loading. Performance is like that `defer function\_name()` in this situation function name will be deferred.  
  
Deferring a call to a function such as Close has two advantages. First, it guarantees that you will never forget to close the file, a mistake that's easy to make if you later edit the function to add a new return path. Second, it means that the close sits near the open, which is much clearer than placing it at the end of the function.  
  
113. Method  
As we saw with ByteSize, methods can be defined for any named type (except a pointer or an interface); the receiver does not have to be a struct.  
In the discussion of slices above, we wrote an Append function. We can define it as a method on slices instead. To do this, we first declare a named type to which we can bind the method, and then make the receiver for the method a value of that type.  
  
func (s secretAgent) speak() { //we have to pay attention to `(s secretAgent)`  
fmt.Println("I am", s.first, s.second)  
}  
  
`sa.speak()` //in here sa is struct which is going to send.  
  
114. Interfaces & Polymorphism  
  
  
  
115 Anonymous Function  
`func() {  
fmt.Println("I am an Anonymous Function")  
}()`  
This is anonymous function interface. If we want to use methods in anonymous function next code is shown.  
`func(x int) {  
fmt.Println(x)  
} (23)`  
  
116. Func expression  
We can set function to one expression. For example `f := func() {} f()`  
f is the expression for defining function  
  
117. Func return  
func main() {  
a := foo()  
fmt.Println(a)  
  
b := bar()  
fmt.Println(b())  
}  
  
func foo() string {  
s := "hello world"  
return s  
}  
  
func bar() func() int {  
return func() int {  
return 473  
}  
}  
  
119. Callback  
func main() {  
a := incrementor()  
b := incrementor()  
  
fmt.Println(a())  
fmt.Println(a())  
fmt.Println(a())  
fmt.Println(a())  
  
fmt.Println(b())  
fmt.Println(b())  
}  
  
func incrementor() func() int {  
var x int  
return func() int {  
x++  
return x  
}  
}  
  
120. Recursion  
Recursion is just like factorial. For get factorial some number. we use extra function instead of main. There are two type of way to get factorial number. First function, second loop.