**Go Functions**

**Functions are first-class citizens in Go. Go supports**

* Functions
* Higher-order Functions that use other functions as arguments and return values.
* User-defined Function Types, a type signature that describes the types of its arguments and return values
* Closures or Function literals
* Variadic functions
* Go functions can return multiple values

**Variadic functions** like AddResource take a variable number of arguments. These arguments are available as a slice inside the function.

**func (ptrToStruct \*ptr) AddResource( paths ...string) {**

**for \_, path := range paths {**

**ptrToStruct.structField().funcValue(path)**

**}**

**}**

**Functions**

// inputs output

// multiple parameters of the same type

**func add(x, y int) int {  
 return x + y  
}**

// multiple parameters return multiple values

**func add(**param1 string, **y int) (int, int) {  
 return y,** len(param1) **}**

**Closures,** **Function literals**, lexically scoped means Closures can access values that were in scope when defining the function

// x is currently in scope

**x := 5**

// definition for a variable fn that is a function; it is of type func() invoke with fn()

**fn := func() {  
 y := x + x**

**fmt.Println(y )  
}**

// just to show scope for new students

**func another\_func() int {**

**// won't compile because y not defined in this scope**

**x = 444**

**return y**

**}**

// a function literal type for a struct

**type op struct {  
 name string  
 fn func(int, int) int  
}**

**User-defined Function Types**

// define the type

**type demoType func(**param1 string **) int**

// The functions demo, demo1 matches the demoType signature.

**func demo(s** string **) int {  
 return len(s)  
}**

**func demo1(s** string **) int {  
 return len(s)  
}**

**demoTypes := []demoType {demo, demo1}**

**Higher-order Functions**

**func demoRunner(fn demoType ) {  
 fmt.Println(fn(“this is pop”) )  
}**

**func main() {**

**demoRunner(demo )**

**}**