Module 2: Function Types
Topic 1.2: Returning Functions

Functions as Return Values

- Functions can return functions
- Might create a function with controllable parameters
- Example: **Distance to Origin function**
  - Takes a point (x, y, coordinates)
  - Returns distance to origin
- What if I want to change the origin?
  - Option 1: Pass origin as argument
  - Option 2: Define function with new origin

Function Defines a Function

```
1. func MakeDistOrigin(o_x, o_y float64)
2.                 func (float64, float64) float64 {
3.     fn := func (x, y float64) float64 {
4.             return math.Sqrt(math.Pow(x - o_x, 2) +
5.                              math.Pow(y - o_y, 2))
6.         }
7.     return fn
8. }
```

- Origin location is passed as an argument
- Origin is built into the returned function

Special-Purpose Functions

```
func main() {
    Dist1 := MakeDistOrigin(0,0)
    Dist2 := MakeDistOrigin(2,2)
    fmt.Println(Dist1(2,2))
    fmt.Println(Dist2(2,2))
}
```

- `Dist1()` and `Dist2()` have different origins

Environment of a Function

- Set of all names that are valid inside a function
- Names defined locally, in the function
- **Lexical Scoping**
- Environment includes names defined in block where the function is defined

```
var x int
funct foo(y int) {
    z := 1
    …
}
```

Closure

- Function + its environment
- When functions are passed/returned, their environment comes with them!

```
func MakeDistOrigin(o_x, o_y float64)
                func (float64, float64) float64 {
   fn := func (x, y float64) float64 {
          return math.Sqrt(math.Pow(x - o_x, 2) +
                           math.Pow(y - o_y, 2))
         }
```

- `o_x` and `o_y` are in the closure of `fn()`