Module 1: Functions and Organization
Topic 1.3: Call by Value, Reference

Call by Value

- Passed arguments are copied to parameters
- Modifying parameters has no effect outside the function

```
func foo(y int) {
    y = y + 1
}
func main() {
    x := 2
    foo(x)
    fmt.Print(x)
}
```

Tradeoffs of Call by Value

- **Advantage: Data Encapsulation**
- Function variables only changed inside the function
- **Disadvantage: Copying Time**
- Large objects may take a long time to copy

Call by Reference

- Programmer can **pass a pointer** as an argument
- Called function has direct access to caller variable in memory

```
func foo(y *int) {
    *y = *y + 1
}
func main() {
    x := 2
    foo(&x)
    fmt.Print(x)
}
```

Tradeoffs of Call by Reference

- **Advantage: Copying Time**
- Don't need to copy arguments
- **Disadvantage: Data Encapsulation**
- Function variables may be changed in called functions
- May be what you want
  - Sort an array