Module 4: Interfaces for Abstraction
Topic 2.1: Using Interfaces

Ways to Use an Interface

- Need a function which takes multiple types as a parameter
- Function `foo()` parameter
  - Type X or type Y
- Define interface Z
- `foo()` parameter is interface Z
- Types X and Y satisfy Z
- Interface methods must be those needed by `foo()`

Pool in a Yard

- I need to put a pool in my yard
- Pool needs to fit in my yard
  - Total area must be limited
- Pool needs to be fenced
  - Total perimeters must be limited
- Need to determine if a pool shape satisfies criteria
- `FitInYard()`
  - Takes a shape as a argument
  - Returns true if the shape satisfies criteria

FitInYard()

- Many possible shape types
  - Rectangle, triangle, circle, etc.
- `FitInYard()` should take many shape types
- Valid shape types must have:
  - `Area()`
  - `Perimeter()`
- Any shape with these methods is OK

Interface for Shapes

```
type Shape2D interface {
    Area() float64
    Perimeter() float64
}
type Triangle {…}
func (t Triangle) Area() float64 {…}
func (t Triangle) Perimeter() float64 {…}
type Rectangle {…}
func (t Rectangle) Area() float64 {…}
func (t Rectangle) Perimeter() float64 {…}
```

- Rectangle and Triangle satisfy Shape2D interface

FitInYard() Implementation

```
func FitInYard(s Shape2D) bool {
     if (s.Area() < 100 &&
         s.Perimeter() < 100) {
          return true
     }
     return false
}
```

- Parameter is any type that satisfies the interface

Empty Interface

- Empty interface specifies no methods
- All types satisfy the empty interface
- Use it to have a function accept any type as a parameter

```
func PrintMe(val interface{}) {
      fmt.Println(val)
}
```