

# Interfaces

**01**

About

**02**

Implementation

**03**

Usage

# Interfaces

- | The type of data expected by a function must be specified in the function parameters
- | Don't always know the type ahead of time
  - | **Interfaces** allow specifying behaviors of a type instead of the type itself
  - | This allows functions to operate on more than one type of data

# Creation & Implementation

```
type MyInterface interface {  
    Function1()  
    Function2(x int) int  
}  
  
type MyType int  
func (m MyType) Function1() {}  
func (m MyType) Function2(x int) int {  
    return x + x  
}  
  
func execute(i MyInterface) {  
    i.Function1()  
}
```



# Notes

- | Interfaces are **implicitly** implemented
  - | When a type has all receiver functions required by the interface, then it is considered implemented
- | Functions operating on interfaces should never accept a pointer to an interface
  - | Caller determines whether pointer or value (copy) is used
- | Prefer multiple interfaces with a few functions over one large interface

# Pass By Value vs Pointer

```
type MyType int // Implements MyInterface

func execute(i MyInterface) {
    i.Function1()
}

m := MyType(1)
execute(m)
execute(&m)
```



# Pointer Receiver Implementation

- | When implementing a pointer receiver function, all functions accepting the interface will only accept pointers
- | If self-modification is needed, implement all interface functions as receiver functions for consistency

```
type MyType int

func (m *MyType) Function1() {}
func (m MyType) Function2(x int) int {
    return x + x
}
```

```
func execute(i MyInterface) {
    i.Function1()
}


m := MyType(1)
execute(m) // Can no longer use value types
execute(&m)
```

# Pointer Receiver Implementation



```
type MyType int
```

```
func (m *MyType) Function1() {}  
func (m MyType) Function2(x int) int {  
    return x + x  
}
```



```
func (m *MyType) Function1() {}  
func (m *MyType) Function2(x int) int {  
    return x + x  
}
```



# Example

```
type Resetter interface {  
    Reset()  
}  
  
type Player struct {  
    health int  
    position Coordinate  
}  
  
func (p *Player) Reset() {  
    p.health = 100  
    p.position = Coordinate{0,0}  
}  
  
func Reset(r Resetter) {  
    r.Reset()  
}
```

```
player := Player{50, Coordinate{5, 5}}  
fmt.Println(player) // {50 {5 5}}  
Reset(&player)  
fmt.Println(player) // {100 {0 0}}
```



# Access Implementing Type

- | It is sometimes needed to access the underlying type that implements an interface
- | Call functions, make modifications, etc

```
func ResetWithPenalty(r Resetter) {  
    if player, ok := r.(Player); ok {  
        player.health = 50  
    } else {  
        r.Reset()  
    }  
}
```

# Recap

- | Interfaces allow functions to operate on more than one data type
- | Interfaces are implicitly implemented
  - | Create receiver functions matching interface function signatures
- | No need to use pointers to interfaces in function parameters
  - | Use a pointer at the call site
- | If a **pointer** receiver function is implemented, then the type can only be used as a pointer in function calls