Go Interface

- ➔ Go does not have classes and inheritance.
- ➔ Go fulfil these requirement with it's powerful interface.
- ➔ Interfaces provide behaviour to an object: *if something can do this it can be used here.*
- ➔ An interface defines a set of abstract methods and does not contain any variable.

Syntax :-

```
type Namer interface{
    Method1(param_list) return_type
    Method2(param_list) return_type

    ...
}
```

- ■ Here Namer is an interface type.

Generally, the name of an interface is formed by the method name plus the [er] suffix, such as Printer, Reader, Writer, Converter, etc.

- A type doesn't have to state explicitly that it implements an interface: interfaces are satisfied implicitly. Multiple types can implement the same interface.
- A type that implements an interface can also have other functions.
- A type can implement many interfaces.
- An interface type can contain a reference to an instance of any of the types that implement the interface.

Go Interface Example:-

```
package main

import (
    "fmt"
)

type vehicle interface {
    accelerate()
}

func foo(v vehicle) {
    fmt.Println(v)
}

type car struct {
    model string
    color string
}

func (c car) accelerate() {
    fmt.Println("Accelrating....")
}

type toyota struct {
    model string
    color string
    speed int
}

func (t toyota) accelerate() {
    fmt.Println("I am toyota, I accelerate
fast...")
}

func main() {
```

```
    c1 := car{"suzuki", "blue"}
    t1 := toyota{"toyota", "Red", 100}
    c1.accelerate()
    t1.accelerate()
    foo(c1)
    foo(t1)
}
```

Output:-

```
Accelrating....
I am toyota, I accelerate fast...
{suzuki blue}
{toyota Red 100}
```