Introduction to Interfaces in Go

In Go, both [structs](https://www.developer.com/languages/understanding-structs-in-go/) and interfaces are used to create custom types that can associate a collection to [methods](https://www.developer.com/languages/methods-in-go-explained/).

**Struct**, which helps in creating **concrete types**.

**Interface** creates **abstract types**.

**Concrete types** are particularly used for representing its values by exposing its methods.

They expose their behaviour through a set of methods.

**Abstract types** means that they cannot be instantiated like struct.

That, however, does not stop us from creating interface type variables.

The abstract types are used to assign any concrete types.

An interface value does not even allow its client to know what it is – the behaviour is only exposed by its methods.

## Empty Interfaces in Go

When an interface does not contain any method signature in its declaration, it is called an empty interface.

interface{}

**Variadic functions:** the functions which not only accepts different types of values as arguments but also the different number of arguments.

The Go uses an empty interface to implement these techniques.

The signature of the **Println** function in Go is as follows:

func Println(a ...interface{})(n int, err, error)

## Embedding an Interface in Go

In Go, there is no concept of extending an interface.

But we definitely can create a new interface embedding more than one interface.

type dawn interface {

morning() string

}

type dusk interface {

night() string

}

type twilight interface {

dawn

dusk

}

Now, since the twilight interface embeds both dawn and dusk interface, we can assign greet struct type to the twilight variable and invoke both the methods without any problem.