Introduction

This lesson gives an introduction to interfaces in Go using an example, also explains how to define a new type that would implement the same interface.

WE'LL COVER THE FOLLOWING

- Definition
- Example
- Defining a New Type

Definition

An **interface** type is defined by a set of methods. A value of interface type can hold any value that implements those methods. Interfaces increase the flexibility as well as the scalability of the code. Hence, it can be used to achieve polymorphism in Golang. Interface does not require a particular type, specifying that only some behavior is needed, which is defined by a set of methods.

Example

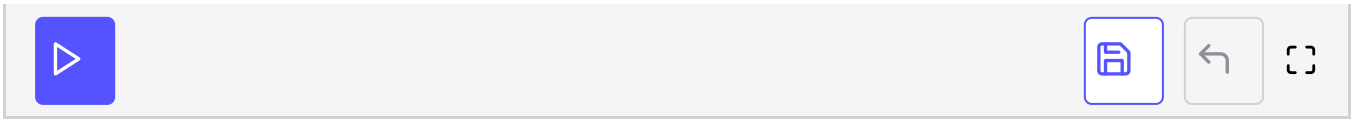
Here is a refactored version of our earlier example. This time we made the greeting feature more generic by defining a function called `Greet` which takes a param of interface type `Namer`. `Namer` is a new interface we defined which only defines one method: `Name()`. So `Greet()` will accept as param any value which has a `Name()` method defined.

To make our `User` struct implement the interface, we defined a `Name()` method. We can now call `Greet` and pass our pointer to `User` type.

Environment Variables

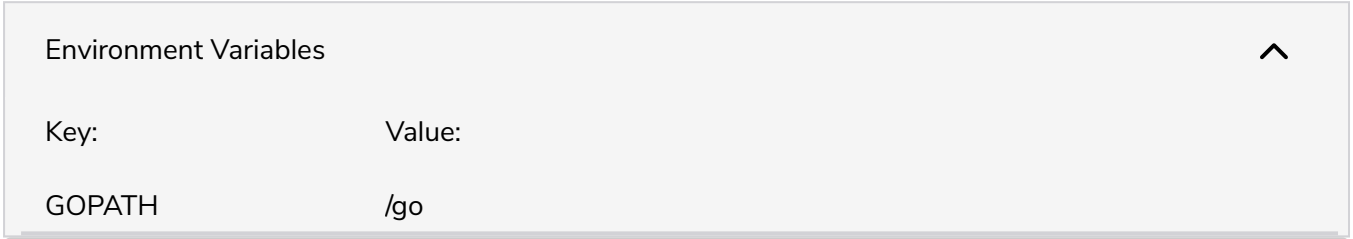
Key:	Value:
GOPATH	/go

```
package main

import (
        "fmt"
)

type User struct {
        FirstName, LastName string
}

func (u *User) Name() string {
        return fmt.Sprintf("%s %s", u.FirstName, u.LastName)
}

type Namer interface {
  Name() string //The Namer interface is defined by the Name() method
}

func Greet(n Namer) string {
        return fmt.Sprintf("Dear %s", n.Name())
}

func main() {
        u := &User{"Matt", "Aimonetti"}
        fmt.Println(Greet(u))
}
```



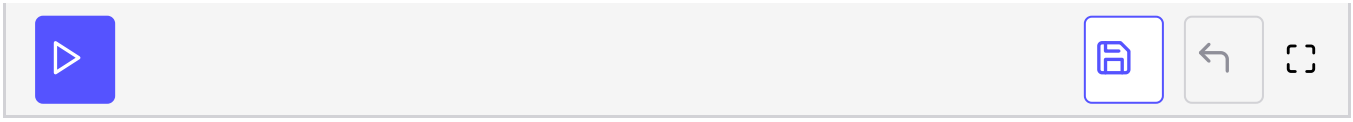
Defining a New Type

We could now define a new type that would implement the same interface and our `Greet` function would still work.




```
package main

import (
        "fmt"
)

type User struct {
        FirstName, LastName string
}

func (u *User) Name() string { //Name method used for type User
        return fmt.Sprintf("%s %s", u.FirstName, u.LastName)
}

type Customer struct {
```

```
type Customer struct {
        Id       int
        FullName string
}

func (c *Customer) Name() string { //Name method used for type Customer
        return c.FullName
}

type Namer interface {
  Name() string //Both Name() methods can be called using the Namer interface
}

func Greet(n Namer) string {
        return fmt.Sprintf("Dear %s", n.Name())
}

func main() {
        u := &User{"Matt", "Aimonetti"}
        fmt.Println(Greet(u))
        c := &Customer{42, "Francesc"}
        fmt.Println(Greet(c))
}
```



Now that we have a basic understanding of interfaces in Go, we will continue to look into how they are satisfied in the next section.