# Type Assertion

This lesson explains converting the type of a given value to another specific value using examples

#### WE'LL COVER THE FOLLOWING ^

- Introduction
- Examples
- Quiz

### Introduction #

In the previous lesson, we covered type conversions. Now, we will move on to discuss how, if you have a value that you want to convert to another or a specific type (in case of interface{}), you can use **type assertion**. A type assertion takes a value and tries to create another version in the specified explicit type.

## Examples #

In the example below, the timeMap function takes a value and if it can be asserted as a map of interfaces{} keyed by strings, then it injects a new entry called "updated\_at" with the current time value.

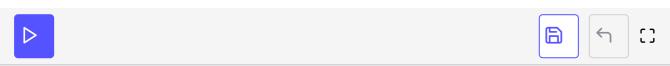
```
Environment Variables

Key: Value:

GOPATH /go

package main
import (
    "fmt"
    "time"
)

func timeMap(y interface{}) {
    z, ok := y.(map[string]interface{}) //asserting y as a map of interfaces
```



The type assertion doesn't have to be done on an empty interface. It's often used when you have a function taking a param of a specific interface but the function inner code behaves differently based on the actual object type. Here is an example:

```
Environment Variables
 Key:
                          Value:
 GOPATH
                         /go
package main
                                                                                         import "fmt"
type Stringer interface {
        String() string
}
type fakeString struct {
        content string
}
// function used to implement the Stringe interface
func (s *fakeString) String() string {
        return s.content
}
func printString(value interface{}) {
        switch str := value.(type) {
        case string:
                fmt.Println(str)
        case Stringer:
                fmt.Println(str.String())
        }
}
func main() {
        s := &fakeString{"Ceci n'est pas un string"}
```

```
printString("Hello, Gophers")
}

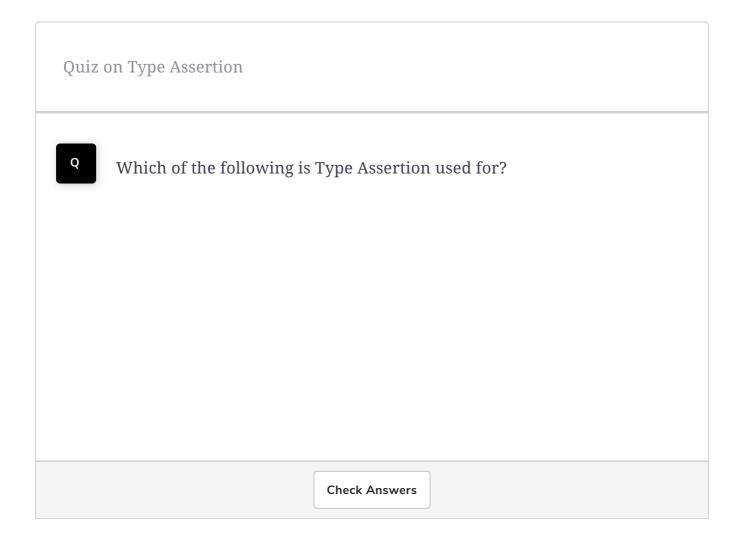
| Compared to the printString of the
```

Another example is when checking if an error is of a certain type:

```
if err != nil {
  if msqlerr, ok := err.(*mysql.MySQLError); ok && msqlerr.Number == 1062 {
    log.Println("We got a MySQL duplicate :(")
  } else {
    return err
  }
}
```

• Read more in the Effective Go guide

# Quiz #



Now that we know how inbuilt types can be manipulated, we will move on to discuss how we can create our own types in Go.