Go Booby Traps 7

Michel Casabianca
casa@sweetohm.net

Go programming language is easy to learn, but there are some tricky traps. This article series is trying to show these booby traps so that you avoid them.

We would like to print the length of a string. So we write following code:

```
package main

func main() {
        s := "échec"
        println("length of", s, "is", len(s))
}
```

On the Playground

But when we run it, we get:

```
$ go run broken.go
length of échec is 6
```

This is probably not what you expected! Why? How could you fix this code?

Explanation

The name character string is inaccurate for Go string type. We should name it `bytes` array. To be precise, a Go string is an array of bytes resulting from the encoding of a string in *UTF-8*. As our string contains an accentuated character, it won't result in a single `byte` encoded in *UTF-8* but in two. Thus the size of `bytes` array is not the same as the `string` length.

It remains to be seen how we can get the number of characters (or *Runes* in Go) in this string...

First solution is to convert the string into a *Runes* array and get its size:

```
package main

func main() {
        s := "échec"
        println("length of", s, "is", len([]rune(s)))
}
```

On the Playground

Another solution is to use dedicated function `RuneCountInString()` from *utf8* package:

```
package main

import "unicode/utf8"

func main() {
        s := "échec"
        println("length of", s, "is", utf8.RuneCountInString(s))
}
```

On the Playground

In both cases, we get the correct size:

```
$ go run fixed.go
length of échec is 5
```

What you must remember here is that `len(string)` returns the length of a string if it contains only ASCII characters. Thus **you should not use function `len(string)` in the general case**.

Enjoy!