String() Method and Format Specifiers

This lesson describes how to use format specifiers and change the String() function to bring variations to the default print functions in Go.

When you define a type with a lot of methods, chances are you will want to make a customized string-output for it with the `String( )` method, in other words: a human-readable and printable output. This is because if `String( )` is defined for a certain type, then this method will be used in `fmt.Printf()` to produce the default output, which is the output produced with the format specifier **%v**. Furthermore, `fmt.Print()` and `fmt.Println()` will automatically use the `String( )` method.

We will test this out with the help of a program:

```
package main
import (
"fmt"
"strconv"
)

type TwoInts struct {
  a int
  b int
}

func main() {
  two1 := new(TwoInts)
  two1.a = 12
  two1.b = 10
  fmt.Printf("two1 is: %v\n", two1) // output: two1 is: (12 / 10)
  fmt.Println("two1 is:", two1) // output: two1 is: (12 / 10)
  fmt.Printf("two1 is: %T\n", two1) // output: two1 is: *main.TwoInts
  fmt.Printf("two1 is: %#v\n", two1) // output: &main.TwoInts{a:12, b:10}
  }

  func (tn *TwoInts) String() string {
    return "(" + strconv.Itoa(tn.a) + " / " + strconv.Itoa(tn.b) + ")"
}
```

String Method

In the above code, we have a struct of type `TwoInts` with two integer fields `a` and `b`. See the header of the `String` method at **line 22** as: `func (tn *TwoInts) String() string`. It shows that the variation of this function can be called on `TwoInts` type variables only. It converts the fields of `tn` to string using the `strconv` package (imported at **line 4**) and concatenating it with `/`, which in turn is placed with *closed brackets*. Then, it returns this string.

Now, look at `main`. We make a variable `two1` of type `TwoInts` using the `new()` function at **line 13**. In the next two lines, we assign the fields of `two1`: `a` and `b` with the values: **12** and **10**.

At **line 16**, we are printing `two1` using the `Printf` function of package `fmt`. The variation of the `String()` method from **line 22** will be called, and (**12 / 10**) will be returned and printed. In the next line, we are printing `two1` using `Println` function of package `fmt`. Notice the format specifier is **%T** in this case. So, the type of `two1` will be printed, which is `*main.TwoInts`. Now, at **line 19**, we are printing `two1` using the `Printf` function of package `fmt`. Notice the format specifier is **%#v** in this case. This format specifier prints the type along with the values, so `&main.TwoInts{a:12, b:10}` is printed.

If you make the mistake of defining `String()` in terms of itself, like in the following snippet, then the program does an infinite recursion (`TT.String()` calls `fmt.Sprintf` which calls `TT.String()` ...) and quickly gives an out of memory error:

```
type TT float64

func (t TT) String() string {
  return fmt.Sprintf("%v", t)
}
t.String()
```

That's it about the `String()` function and its role in printing output. In the next lesson, we'll study how methods work on embedded structs.