Bytes and Slices

This lesson provides necessary information about the bytes package provided by Golang.

WE'LL COVER THE FOLLOWING

The `bytes` package

Slices of *bytes* are so common that Go has a `bytes` package with manipulation functions for that kind of type. It is very analogous to the strings package. Moreover, it contains a very handy type `Buffer`:

```
import "bytes"
type Buffer struct {
  ...
}
```

It is a *variable-sized* buffer of bytes with `Read` and `Write` methods because reading and writing an unknown number of bytes is best done buffered.

A `Buffer` can be created as a value, like:

```
var buffer bytes.Buffer
```

or as a pointer with `new` as:

```
var r *bytes.Buffer = new(bytes.Buffer)
```

or created with the function:

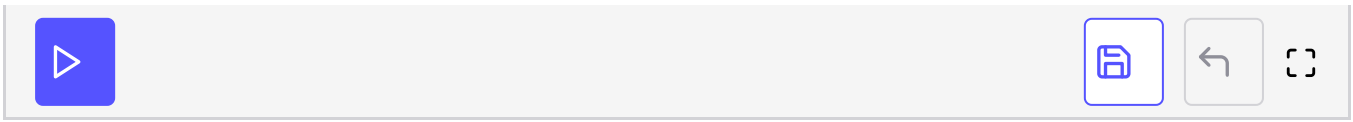
```
func NewBuffer(buf []byte) *Buffer
```

This creates and initializes a new Buffer using `buf` as its initial contents. It is best to use `NewBuffer` only to read from `buf`. Here, is a simple example of code writing to a buffer:

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

func main() {
  // New Buffer.
  var b bytes.Buffer
  // Write strings to the Buffer.

  b.WriteString("ABC")
  b.WriteString("DEF")
  // Use Fprintf with Buffer.
  fmt.Fprintf(&b, " A number: %d, a string: %v\n", 10, "bird")
  b.WriteString("[DONE]")
  // Convert to a string and print it.
  fmt.Println(b.String())
}
```



Writing to a Buffer

In the code above, in `main` at **line 9**, we make a buffer as: `var b bytes.Buffer`. Now, we start writing to the buffer `b`. Look at **line 12** and **line 13**, where we write two strings **ABC** and **DEF** to the buffer `b`. At **line 15**, we are printing the value placed at address **&b**, which will print **ABCDEF** although written to `b` initially. Then at **line 16**, we write **[DONE]** to the buffer `b` and print it by `b.String()`, which returns string written to the buffer at **line 18**.

Concatenating strings by using a buffer

This works analogous to Java's `StringBuilder` class. Make a buffer, append each string `s` to it with the `buffer.WriteString(s)` method, and convert back to a string with `buffer.String()` at the end, as in the following code snippet:

```
var buffer bytes.Buffer
for {
  if s, ok := getNextString(); ok { //method getNextString() not shown her
e
    buffer.WriteString(s)
```

```
    } else {
        break
    }
}
fmt.Print(buffer.String(), "\n")
```

This method is much more memory and CPU-efficient than +=, especially if the number of strings to concatenate is large.

That's it about the `bytes` package and some of its important functions. In the next lesson, you'll learn how to use the for construct on the slices.