Strings

This lesson discusses the datatype string in detail.

WE'LL COVER THE FOLLOWING

- Introduction
- Strings in Go
  - Advantages of strings
  - Types of string literals
    - Interpreted strings
    - Raw strings
  - Length of a string
  - Concatenation of strings

Introduction

Strings are a sequence of **UTF-8 characters** (the 1-byte ASCII-code is used when possible, and a 2-4 byte UTF-8 code when necessary). UTF-8 is the most widely used encoding. It is the standard encoding for text files, XML files, and JSON strings. With `string` data type, you can reserve 4 bytes for characters, but Go is intelligent enough that it will reserve *one-byte* if the string is only an ASCII character.

Strings in Go

Contrary to strings in other languages as C++, Java or Python that are fixed-width (Java always uses 2 bytes), a Go string is a sequence of variable-width characters (each 1 to 4 bytes).

Advantages of strings

The advantages of strings are:

- Go strings and text files occupy less memory/disk space (because of variable-width characters).
- Since UTF-8 is the standard, Go doesn't need to encode and decode strings like other languages have to do.

Strings are value types and immutable, which means that once created, you cannot modify the contents of the string. In other words, strings are immutable arrays of bytes.

Types of string literals

Two kinds of string literals exist:

- Interpreted strings
- Raw strings

Interpreted strings

They are surrounded by " " (double quotes). For example, escape sequences are interpreted:

```
"\n" // represents a newline
"\r" // represents a carriage return
"\t" // represents a tab
"\u" // represents Unicode characters
"\U" // also represents Unicode characters
```

Raw strings

They are surrounded by back quotes ` ` and are not interpreted. For example in the following string:

```
`This is a raw string \n`
```

`\n` is not interpreted but taken literally. They can span multiple lines.

Length of a string

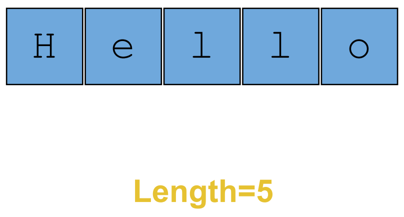
Strings in Go do not terminate by a special character as in C/C++. The initial (default) value of a string is the empty string “”. The usual comparison operators (==, !=, <, <=, >=, and >) work on strings by comparing byte by byte in memory.

The length of a string `str` (the number of bytes) is given by the `len()`

function:

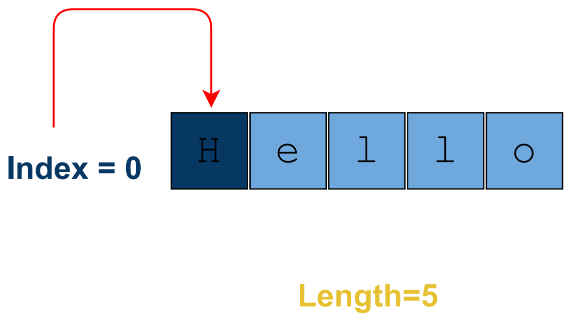
```
len(str)
```

The contents of a string (the *raw* bytes) are accessible via standard indexing methods, the index is written between [], with the index starting from **0**. The following illustration shows indexing of a string of length **5**:



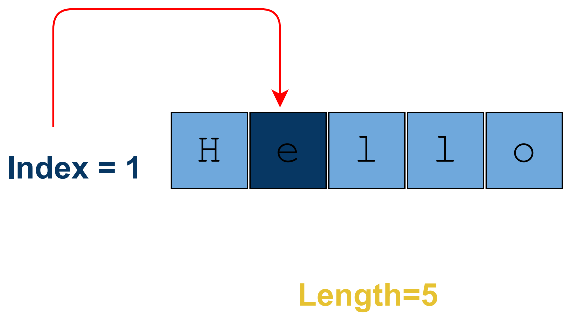
Indexing the String

1 of 6



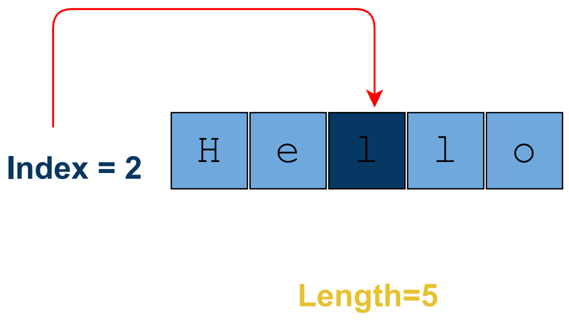
Indexing the String

2 of 6



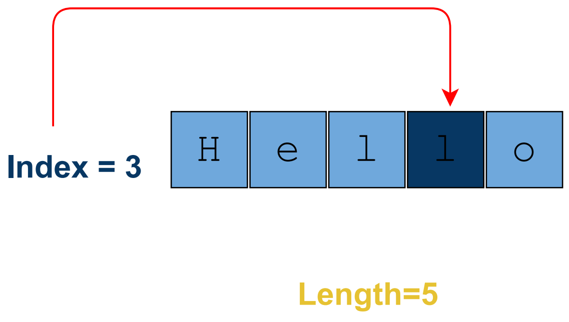
Indexing the String

3 of 6



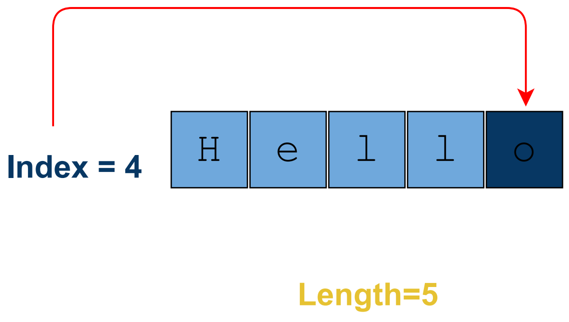
Indexing the String

4 of 6



Indexing the String

5 of 6



Indexing the String

6 of 6

So you can see that:

- The first byte of a string `str` is given by: `str[0]`
- The i-th byte by: `str[i]`
- The last byte by: `str[len(str)-1]`

> **Note:** Taking the address of a character in a string is illegal.

Concatenation of strings

Two strings `s1` and `s2` can be made into one string `s` with:

```
s := s1 + s2
```

`s2` is appended to `s1` to form a new string `s`. Multi-line strings can be constructed as follows:

```
str := "Beginning of the string " +
"second part of the string"
```

The `+` has to be on the *first line* due to the insertion of ; by the compiler. The append shorthand += can also be used for strings. Run the following program to see how concatenation works:

```
package main
import "fmt"

func main(){
  s := "Hel" + "lo "
  s += "World!"
  fmt.Println(s) // prints out “Hello World!”
}
```

Concatenation of Strings

As you can see, the two strings `Hel` and `lo ` are concatenated at **line 5**, and the result of this concatenation is further concatenated with the string `World!` at **line 6**. The final string **Hello World!** is printed at **line 7**.

Now that you are familiar with the basics of strings in Go, you will write a function to solve a problem with strings in the next lesson.