Date No

Maps Equality

Maps cant be compared using the == operator. The == can be only used to check the map is nil

Strings.

A String is a slice of bytes in Go. Strings can be created by enclosing a set of characters inside double quotes " ".

Strings in Go are Unicode compliant and UTF-8 encoded.

Accessing individual bytes of Strings.

Since String is a Slice of bytes, It's possible to access each byte of a String.

```
func printBytes(s string) {
    fmt.Printf("Bytes")
    for i:=0; i< len(s); i++ {
        fmt.Printf("%x", s[i])
    }
```

returns no.of bytes in the String.

%x format Specifier to print hexadecimals

```
func main() {
    name := "Hello World"
    fmt.Printf("String : %s \n", name)
    printBytes(name)
}
```

%s format Specifier to print Strings

→ String : Hello World

Byte: 48 65 6c 6c 6f 20 57 6f 72 6c 64

Date No

Accessing individual character of a String.

```
func printBytes (s string) {
    fmt.Printf("\n Bytes ")
    for i:=0; i<len(s); i++ {
        fmt.Printf("%.x ", s[i])
}

func printChars (s string) {
    fmt.Println("\n characters :")
    for i:=0; i<len(s); i++ {
        fmt.Printf("%.c ", s[i])
}
```

↑ format specifier to print characters.

```
func main () {
    name := "Hello world"
    fmt.Println("\n String", name)
    printChars (name)
    printBytes (name)
}
```

→ String : Hello world
characters :
Hello world
Bytes :
48 65 6c 6c 6f 20 57 6f 72 6c 64

```
func printChars (s string) {
    fmt.Println("characters :")
    for i:=0; i<len(s); i++ {
        fmt.Printf("%.c ", s[i])
}
```

Date No

```
func main() {
  name := "Señor"
  fmt.PrintChar(name)
}
```

→ characters :

S e ± A ± o r ← wrong output

Bytes

53 65 C3 b1 6F 72

The unicode code point of ñ is U+00F1 and its UTF-8 encoding occupies 2 bytes, C3 and b1.

In UTF-8 encoding a code point can occupy more than 1 byte.

use rune to solve this.

Rune

Rune is builtin type. Rune represents a unicode code point in Go. It dosen't matter how many bytes the code point occupies, it can be represented by a rune.

Let's modify the printChar function

Señor

```
func printChar(s string) {
  fmt.Print("characters : ")
  runes := []rune(s)    // x := float64(a)
  for i := 0; i < len(runes); i++ {
    fmt.Printf("%c", runes[i])
  }
}
```

builtin type

String is converted to a slice of rune.

→ characters

S e ñ o r

Bytes

53 65 C3 6F 72

Date No

Accessing individual runes using for range loop.

```
func charsAndBytePosition(s string) {
    for index, rune := range s {
        fmt.Printf("%c starts at byte %d\n", rune, index)
    }
}

func main() {
    name := "Señor"
    charAndBytePosition(name)
}
```

(value)

Creating a String from a Slice of bytes.

```
func main() {
    byteSlice := []byte{0x43, 0x61, 0x66, 0xc3, 0xA9}
    str := string(byteSlice) ← convert byte slice to string
    fmt.Println(str)
}
```
→ café

Decimal values also work and also print café

```
func main() {
    byteSlice := []byte{67, 97, 102, 195, 169}
}
```

Decimal equalent of {'\x43', '\x61', '\x66', '\xc3', '\x49'}

Date No

Creating a String, from a Slice of runes.

```
func main() {
    runeSlice := []rune {0x0053, 0x0065, 0x00f1, 0x006f,
                         0x0072 }
    Str := String (runeSlice)
    fmt.Println (Str)
} -> Señor
```

(S, e, ñ, o, r)

String length.

Utf8. RuneCountInString (s string) (n int) function of utf8 package used to find the length of a String. len(s) is used to find the no of bytes in the String. Some unicodes points occupy more than 1byte, will return the incorrect length.

```
func printLengthAndBytes (S string) {
    fmt.Printf (" String : %s \n Length : %d \n Size in
                Bytes %d \n ", S, utf8.RuneCountInString
                (S), len (S) )
}

func main () {
    Str1 := "Señor"
    Str2 := "Andrew"
    printLengthAndBytes (Str1)
    PrintLengthAnd Bytes (Str2)
}
```

-> String : Señor
Length : (5)
no of Bytes : (6)

String : Andrew
Length : 6
no of Bytes : 6

Date No

String comparison.

The == operator is used to compare two Srings for equality. If Both equal result is true.

```
func compareStrings(S1, S2 string){
  if S1==S2 {
    fmt.Printf("%S and %S are equal ", S1, S2)
    return
  }
  fmt.Printf("%S and %S are not equal", S1, S2)
}

func main() {
   Str1 := "Go"
   Str2 := "Go"
   ComareStrings (Str1, Str2)

   Str3 := "Hello"
   Str4 := "world"
   CompareString (Str3, Str4)
}
```

→ Go and Go are equal
Hello and world are not equal.

String Concatenation.

Most Simple way to perform String concatenation is + operator.

```
func main ()
  Str1 := "Go"
  Str2 := "is owsome"
```

→ Go is owesome

Date No

Concatenate using Sprintf function. ← fmt.

```
func main() {
    str1 := "Go"
    str2 := " is owesome"
    result := fmt.Sprintf("%s %s", str1, str2)   ← Similar to printf
    fmt.Print(result)
}
```
→ Go is awesome

Strings are immutable.

Strings are imutable in Go. Once a String is created it's not possible to chang it.

```
func mutate(s string) string {
    s[0] = 'a'   ← error! cannot assign s[0] (value of type byte)
    // any valid unicode character within single code
       is a rune.
    return s
}

func main() {
    h := "hello"
    fmt.Println(mutate(h))
}
```

To mutate → String are Converted to Slices. Of Runes.

```
func mutate(s string) string {
    r := []rune(s)
    r[0] := 'A'
    return string(r)
}

func main() {
```