Handling Text from Around the World in Go

Marcel van Lohuizen
@mpvl_
github.com/mpvl
Go Team @ Google Zurich

or…

The golang.org/x/text subrepository

or, for those unfamiliar with it…

Internationalization and Localization

- Searching and Sorting
- Upper, lower, title case
- Bi-directional text
- Injecting translated text
- Formatting of numbers, currency, date, and time
- Unit conversion
- etc. etc. etc.

Status golang.org/x/text

Language tags
- language
  - display

String equivalence
- collate
- search
- secure
  - precis

Other
- many internal packages

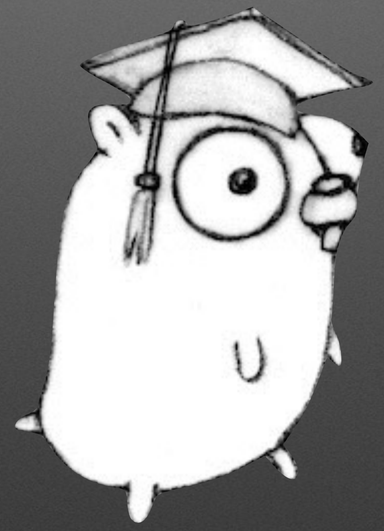
Text processing
- cases
- encoding
  - ...
- runes
- *segment*
- secure
  - bidirule
- transform
- unicode
  - *bidi*
  - cldr
  - norm
  - rangetable
- width

Formatting
- currency
- *date*
- message
- *number*
- *measure*
  - *area*
  - *length*
  - *...*
- *feature*
  - *gender*
  - *plural*

All Native Go

Why not just wrap ICU?

Unicode in Go Refresher



Gopher by Renée French

Go and UTF-8

Go natively handles UTF-8:

```
const nihongo = "日本語"
for i, runeValue := range nihongo {
    fmt.Printf("%#U starts at byte position %d\n", runeValue, i)
}
```

The output shows how each code point (rune) occupies multiple bytes:

```
U+65E5 '日' starts at byte position 0
U+672C '本' starts at byte position 3
U+8A9E '語' starts at byte position 6
```

String Model

- UTF-8
- Same format for source code as for text handling
- No meta data (except for byte length) or string “object”
- Strings not in canonical form
- No random access

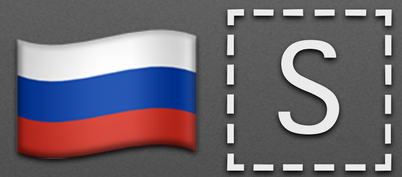
Normal Forms

	é	ặ
NFC	U+00e9	U+1eb7
NFD	e U+0301	a U+0323 U+0306
not normalized		a U+0306 U+0323

- Hard to maintain normalized form
- Often cheap to do on the fly for operations that need it

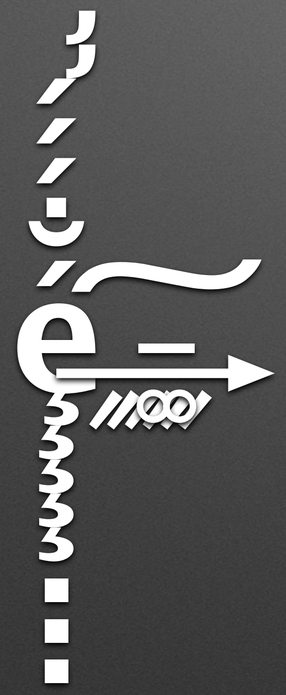
No Random Access

```
const flags = "🇰🇷🇺🇸"  // country code "kr" and "us"

fmt.Println(flags[4:])
```



Text processing is inherently sequential, even for UTF-32

Sequential nature of text



Title(“ΟΣ……”) == “Ος……”

Title(“ΟΣ……a”) == “Οσ……a”

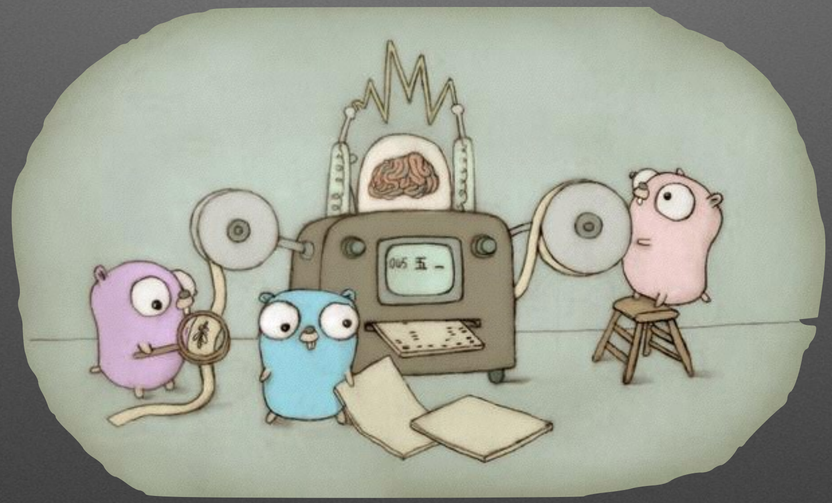
Iterate over Characters

```
import (
   "fmt"
   "golang.org/x/text/unicode/norm"
)

func main() {
   s := norm.NFD.String("Mêlée")

   for i := 0; i < len(s); {
      d := norm.NFC.NextBoundaryInString(s[i:], true)
      fmt.Printf("%[1]s: %+[1]q\n", s[i:i+d])
      i += d
   }
}
```

Output:

```
M: "M"
ê: "e\u0302"
l: "l"
é: "e\u0301"
e: "e"
```

Transforming Text



Gophers by Renée French

Transformers

- x/text packages with transformers:
  - cases
  - encoding/...
  - runes
  - transform
  - width
  - secure/precis
  - secure/bidirule
  - unicode/norm
  - unicode/bidi

Transformer Interface

```
type Transformer interface {
    Transform(dst, src []byte, atEOF bool) (nDst, nSrc int, err error)
    Reset()
}
```

Gopher by Renée French

Transformers

- Streaming like io.Reader/Writer, but faster
- Like ICU transforms, but Go, not DSL
- package transform provides helper functions:

`NewReader`	Create io.Reader from Transformer
`NewWriter`	Create io.Writer from Transformer
`String`	Convert strings using Transformer
`Bytes`	Convert []byte using Transformer
`Append`	Convert []byte appending to buffer

- Not thread-safe (unless noted otherwise)!

Using Transformers

- Helper function:

```
gbk := simplifiedchinese.GBK.NewEncoder()
s, _, _ := transform.String(gbk, "你好")
```

- Most packages provide convenience wrappers

```
s := gbk.String("你好")

w := norm.NFC.Writer(w)
```

Package cases

Title case:

```
toTitle := cases.Title(language.Dutch)

fmt.Println(toTitle.String("'n ijsberg"))
```

Output:

```
'n IJsberg
```

Languages may require different casing algorithms!

Chaining Transforms

- Objective: remove accents from text

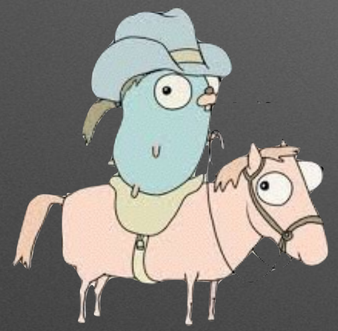
```
rm := runes.Remove(runes.In(unicode.Mn))
```

- Does not handle composed characters, like U+00E9 (é)
- Use transform.Chain with NFD and NFC normalization:

```
t := transform.Chain(norm.NFD, rm, norm.NFC)
s, _, _ := transform.String(t, “résumé") // “resume”
```

- Using transform.Append may be easier if no streaming is needed.

Language Identification



Language Tags

- BCP 47 Language Tag
- Identifies both locale and language, depending on context
- No data (data in separate packages)
- Package golang.org/x/text/language

Language Tag Examples

<lang> [-<script>] [-<region>] [-<variant>]* [-<extension>]*

`en`	English (defaults to American English)
`af-Arab`	Afrikaans in Arabic script
`en-US`	American English
`en-oxendict`	English using Oxford English dictionary spelling
`nl-u-co-phonebk`	Dutch with phone-book sort order

Matching is Non-Trivial

- Swiss German speakers usually understand German `gsw` ⇒ `de`
- The converse is not often true! `de` ⇏ `gsw`
- `cmn` is Mandarin Chinese, `zh` is more commonly used
- `hr` matches `sr-Latn`
- Angolan Portuguese (`pt-AO`) is closer to European Portuguese (`pt-PT`) than Brazilian (`pt`)

The Matcher in x/text/language solves this problem

Language Matching

- Problem:
  match user-preferred language to supported language

- General approach:

  1. User language.Matcher to find best match

  2. Use matched tag to select language-specific resources
     - translations
     - sort order
     - case operations

Language Matching in Go

```
import (
  “http”,
  ”golang.org/x/text/language”
)

var matcher = language.NewMatcher([]language.Tag{
    language.AmericanEnglish,  // en-US
    language.German,           // de
})

func handle(w http.ResponseWriter, r *http.Request) {
 prefs, _, _ := language.ParseAcceptLanguage(
                              r.Header.Get(“Accept-Language”))

 tag, _, _ := matcher.Match(prefs…)
}
```

Example language matching

```
var matcher = language.NewMatcher([]language.Tag{
    language.English
    language.SimplifiedChinese // zh-Hans
})

func foo() {
    pref := language.Make("cmn-u-co-stroke")

    tag, _, _ := matcher.Match(pref) // zh-Hans-u-co-stroke

    c := collate.New(tag) // Correct sort order is used!
}
```

Custom locale-specific data

```
var matcher = language.NewMatcher([]language.Tag{
    language.English
    language.SimplifiedChinese
})

var flags = []string{"🇺🇸","🇨🇳"}

func foo() {
    pref := language.Make("cmn-u-co-stroke")

    _, index, _ := matcher.Match(pref)

    selectedFlag := flags[index] // 🇨🇳
}
```

Searching and Sorting

Multilingual Search and Sort

- Accented characters: e < é < f
- Multi-letter characters: "ch" in Spanish
- Equivalences: å ⇔ aa in Danish and ß ⇔ ss in German
- Reordering: Z < Å in Danish
- Compatibility equivalence: K (U+004B) ⇔ K (U+212A)
- Reverse sorting of accents in Canadian French
- Compound modifiers in Tibetan

Comparing strings

Pick the right package for the right task

search	localized search (and replace)
collate	localized comparison
secure/precis	comparing labels (domain names, user names, passwords)
cases folding	custom case-insensitive compare, but don’t forget to normalize!
unicode/norm	hardly ever the right tool

Using normalization or case folding is often not the right approach!

Search and Replace

- Using bytes.Replace to replace “a cafe” with “many cafes” in:
  1. “We went to a cafe.”
  2. “We went to a café.”
  3. “We went to a cafe/u0301.”
- Result case 3:

  “We went to many cafes/u0301.” ＝NFC ⇒

  “We went to many cafeś.”

Simple byte-oriented search and replace will not work!

x/text/search Example

SEARCH	TEXT	MATCH
aarhus	Århus a\u0303\u031b	Århus
a		a\u0303\u031b
a\u031b\u0303		a\u0303\u031b

```
m := search.New(language.Danish,
                search.IgnoreCase, search.IgnoreDiacritics)

start, end := m.IndexString(text, s)

match := s[start:end]
```

x/text/collate Example

```
import (
    "fmt"

    "golang.org/x/text/collate"
    "golang.org/x/text/language"
)

func main() {
    a := []string{“résumé”,"Resume", "Restaurant"}
    collate.New(language.Und).SortStrings(a)
    fmt.Println(a)
}
```

Output: `[Restaurant Resume résumé]`

Secure comparison

- Compatibility mappings
  - "é" (NFC) versus "é" (NFD)
  - "K" versus "K" (Kelvin symbol)
  - “a b” versus “a b”
- Mixed-script spoofing detection (planned)
  - http://citibank.com
  - http://citibank.com // Using Cyrillic "c".

Translation Insertion

Gopher by Renée French

Translating Text

- General approach
  1. Mark text within your code “To Be Translated”
  2. Extract the text from your code
  3. Send to translators
  4. Insert translated messages back into your code

Mark Text “To Be Translated”

Before:

```
import "fmt"

// Report that person visited a city.
fmt.Printf("%[1]s went to %[2]s.", person, city)
```

After:

```
import "golang.org/x/text/message"

p := message.NewPrinter(userLang)

// Report that person visited a city.
p.Printf("%[1]s went to %[2]s.", person, city)
```

Insert Translations in Code

```
import "golang.org/x/text/message"

message.SetString(language.Dutch,
                  "%[1]s went to %[2]s.",
                  "%[1]s is in %[2]s geweest.")

message.SetString(language.SimplifiedChinese,
                  "%[1]s went to %[2]s.",
                  "%[1]s去了%[2]s。")
```

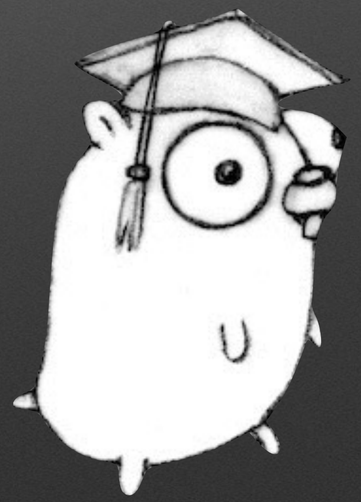
Conclusion

- Human languages are hard to deal with
- x/text can simplify it for you

Q & A

- References
  - godoc.org/golang.org/x/text
  - blog.golang.org/matchlang
  - blog.golang.org/normalization
  - blog.golang.org/strings
  - golang.org/issue/12750

Thank you
Marcel van Lohuizen
@mpvl_
github.com/mpvl



Gopher by Renée French