CSC 491 / 391
Mobile Application Development for iOS II

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

1

Strings in Swift

2

Strings and Characters

- Goal: full and correct representation of Unicode characters
  - 143,859 characters, 2020, Unicode 13.0
  - 16-bits not enough (needs 21 bits to represent all characters in Unicode)
- Fixed width representation of characters is wasteful
  - Most commonly used characters (ASCII) can fit in 8-bits
- Variable width representation of characters is more economical
  - More complicated operations on strings
  - Worth the trouble to conserve memory

We ❤️Emoji's! 👍

3

Strings in Swift

- Goals
  - Simplicity. Efficiency. Unicode correctness.
- Unicode correctness by default
  - Full support of Unicode
  - A character is a *grapheme,* not a *code point*
  - Graphemes can be of variable length
- String conforms to Collection of characters
  - All collection methods can be applied to strings
  - Random access by indices is expensive

4

What is a Character?

- A user-perceived character

  a B é И π क्ष 草 🇺🇸 🇫🇷 👍 👩 👩‍👧

- A Swift character – a single extended *grapheme cluster*
  - *Grapheme* – the smallest unit of a writing system of any given language
  - *Grapheme cluster* – a cluster of code points, i.e., integer values, that form a grapheme.
- A character can be represented by one or more *code points* or *grapheme clusters*.
  - **The representations are not unique**

5

Unicode Encoding Schemes

- Code point – a value represent a position in the Unicode code space.
  - 17 planes of 65.536 (2^{16}) code points each
- Unicode scalar value – a unique 21-bit number for a character: 0x0 to 0x10FFFF
- Variable length encoding schemes
  - UTF-8: each character is encoded by 1 to 4 8-bit *bytes*
  - UTF-16: each character is encoded by 1 or 2 16-bit *code units*
  - UTF-32: each character is encoded by a single 32-bit *code points*

6

© Xiaoping Jia, 2015-2021

Evolution of Characters

- ASCII, 1963
  - English alphabet, digits, punctuations, symbols
  - 128 code points (characters)
  - 7-bit integer
- Extended ACSII, 1987 (ISO-8859, Latin-1)
  - Characters in Latin script
  - Western European Languages
  - 256 code points (characters).
  - 8-bit integer
- Unicode 2.0, 1992
  - Encode most of world's writing systems
  - 65.536 code points
  - 16-bit integer
- Unicode 13.0, 2020
  - 154 modern and historic writing systems
  - 143,859 characters
  - 1,114,112 code points
  - 21 bits to encode the entire code space

7

Strings and Characters

- A simple string

```
var str = "Hello!"
print(str.count)
for c in str {
    print(c, type(of: c))
}
```

```
6
H Character
e Character
l Character
l Character
o Character
! Character
```

- And a surprise

```
str[0]
```

  - **Error**: 'subscript' is unavailable

8

Grapheme Clusters

- A Swift character is an *extended grapheme cluster*
  - A sequence of one or more Unicode scalars to produce a single human-readable character
- The character é can be produced in two ways
  - U+00E9 é
  - U+0065 U+0301 e ◌́

Unicode correct

Most other languages

```
let cafe1 = "caf\u{E9}"
let cafe2 = "cafe\u{301}"
cafe1.count
cafe2.count
cafe1 == cafe2
```

```
"café"
"café"
4
4
true
```

```
"café"
"café"
4
5
false
```


9

Zero-Width Joint

- Zero-Width Joint (ZWJ) sequences
  - Formed with U+200D
- Flags, skin tone, professions
- 👩 (woman), 👨 (man), 👧 (girl), 👦 (boy)

```
var family = "👨"
family += "\u{200D}👩"
family += "\u{200D}👧"
family += "\u{200D}👦"
family.count
```

```
"👨"
"👨👩"
"👨👩👧"
"👨👩👧👦"
1
```

Adult
Adult + Adult
Family with 2 adults and 1 child
Family with 2 adults and 2 children

10

Flag Emoji

- Flags consist of two regional indicator letters representing country codes

```
let usa1 = "🇺🇸"
let usa2 = "🇺" + "🇸"

usa1 == usa2

usa1.count
usa1.unicodeScalars.count
usa1.utf8.count
usa2.count
usa2.unicodeScalars.count
usa2.utf8.count
```

```
"🇺🇸"
"🇺🇸"

true

1
2
8
1
2
8
```


11

Modifiers

- Skin tones
- Professions

```
let girl1 = "👧"
let girl2 = "👧" + "🏽"

let man = "👨"
let aesculapius = "⚕"
let maleDoctor1 = "👨" + "\u{200D}⚕"
let maleDoctor2 = "👨" + "🏽" + "\u{200D}⚕"
```

```
"👧"
"👧🏽"

"👨"
"⚕"
"👨‍⚕"
"👨🏽‍⚕"
```


12

© Xiaoping Jia, 2015-2021

Strings

- Strings are not arrays of Characters
  - Unicode correct by default
  - Compact and space efficient representations
- Several different *views* of string
  - A collection of *characters* (Most useful for app developers)
  - A collection of *Unicode scalars, UTF8's, UTF16's*
- The length, indexes, and slices of a string are view dependent

DePaul University 6

13

The Character View of Strings

- Views are read-only
- Support
  - Character count
  - Iterate

```
var str = "Héllo, Emoji!🇺🇸👍"
print("character count: ", str.count)
for v in str {
    print(v, type(of: v))
}
```

DePaul University 7

14

The Character View of Strings

- Iterate revered

```
for v in str.reversed() {
    print(v, type(of: v))
}
```

Output:

```
👍 Character
🇺🇸 Character
! Character
i Character
j Character
o Character
m Character
E Character
  Character
, Character
o Character
l Character
l Character
é Character
H Character
```

DePaul University 8

16

Using Index

- `String.Index` is not `Int`

```
str.startIndex
str.endIndex
str[str.startIndex]
str[str.index(after: str.startIndex)]
str[str.index(before: str.endIndex)]

let idx3 = str.index(str.startIndex, offsetBy: 3)
let idx10 = str.index(str.startIndex, offsetBy: 10)
str[idx3]
str[idx10]
let idx_3 = str.index(str.endIndex, offsetBy: -3)
str[idx_3]
```

DePaul University 9

17

Advance Index

- Forward and backward. Must stay in bound.
- Comparable

```
var i = idx3
while i < idx10 {
    print(str[i], type(of: str[i]))
    i = str.index(after: i)
}

i = idx10
while i >= idx3 {
    print(str[i], type(of: str[i]))
    i = str.index(before: i)
}
```

DePaul University 10

18

Runtime Performance of Indexing

- Not all indexing methods are equal in performance

```
for i in 3 ..< 10 {
    print(str[str.index(str.startIndex, offsetBy: i)])
}
```

```
var idx = str.index(str.startIndex, offsetBy: 3)
for _ in 3 ..< 10 {
    print(str[idx])
    idx = str.index(after: idx)
}
```

DePaul University 11

19

Prefix, Suffix, and Substring

```
str.hasPrefix("Hello")
str.hasPrefix("Héllo")

str.hasSuffix("playground")
str.hasSuffix("!🇺🇸👍")

str.contains("play")
str.contains("Emoji")

str.range(of: "Hello")
if let range = str.range(of: "Héllo") {
    str[range]
}
```

DePaul University 12

21

Substrings

```
let range = idx3 ..< idx10
str[range]
type(of: str[range])

str.substring(with: range)
str.substring(to: str.index(str.startIndex, offsetBy: 5))
str.substring(from: str.index(str.startIndex, offsetBy: 5))
```

DePaul University 13

22

Modify Strings

```
var greeting = "-- Hello, Swift!"
if let range = greeting.range(of: "Hello") {
    greeting[range]
    greeting.replaceSubrange(range, with: "Ciao")
}
if let range = greeting.range(of: ",") {
    greeting.removeSubrange(range)
}

greeting.insert(",",
    at: greeting.index(greeting.startIndex, offsetBy: 7))
greeting.append(" 👋 ")
greeting += " 👋 "
```

DePaul University 14

23

Format Strings

25

Format Strings

- Using format strings similar to `printf`

  `String(format: format_string, values, … )`
- The basic formats
  - `%@` string
  - `%d %D` integer
  - `%f %F` floating-point
  - `%%` literal %

```
String(format: "%@ %d %f %.2f%%",
       "abc", 300, 2.75, 82.2754)
```

Output:
```
abc 300 2.750000 82.28%
```

- Width and precision, both optional
  - `%w.nf`

DePaul University 16

26

Format String

- Padding and left adjust

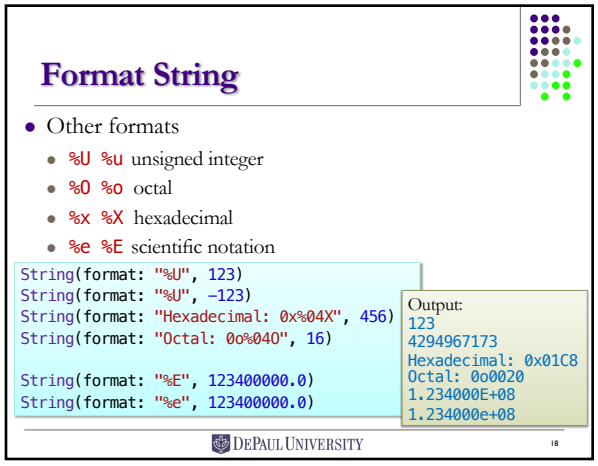
```
String(format: "|%6d|%6d|", 12, 345)
String(format: "|%-6d|%-6d|", 12, 345)
String(format: "|%6.2f|%6.2f|", 1.2, 3.4567)
String(format: "|%-6.2f|%-6.2f|", 1.2, 3.4567)

String(format: "The current time is %02d:%02d", 10, 4)
```

Output:
```
|    12|   345|
|12    |345   |
|  1.20|  3.46|
|1.20  |3.46  |
The current time is 10:04
```

DePaul University 17

27


Format String
Other formats
%U %u unsigned integer
%O %o octal
%x %X hexadecimal
%e %E scientific notation
String(format: "%U", 123)
String(format: "%U", -123)
String(format: "Hexadecimal: 0x%04X", 456)
String(format: "Octal: 0o%04O", 16)
String(format: "%E", 123400000.0)
String(format: "%e", 123400000.0)
Output:
123
4294967173
Hexadecimal: 0x01C8
Octal: 0o0020
1.234000E+08
1.234000e+08
DePaul University
18

28