CSC 471 / 371
Mobile Application Development for iOS

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

0

Outline

- A little history:
  from Objective-C to Swift
- Xcode Playground
- A primer on Swift programming language, the basics
  - Data types
  - Control statements

5

Objective-C Programming Language

6

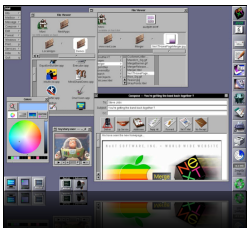
Objective-C

- Invention of C
  - Dennis Ritchie at AT&T (Bell Labs) invented C early 1970
- Extension of Objective-C
  - Brad Cox designed Objective C in 1983
    - Inspired by SmallTalk
    - Layered on top of C language
    - Added objects
- Contemporary to C++
  - Invented in 1979, by Bjarne Stroustrup at Bell Labs
  - Originally called C with classes

9

NeXTSTEP and NeXT Cube

- First major licensee
  - 1985 – Steve Jobs, NeXT Computer
  - Used to develop UI for NeXTSTEP OS (and later, OpenStep)
- Objective-C became the basis for the operating system for the NeXT cube



10

From NeXTSTEP to Mac OS X

- Apple acquired NeXT Inc.
- Mac OS X 10.0
  - March, 2001, Initial release of OS X
- Based on NeXTSTEP
  - Mac Toolbox APIs
  - 'Mac-like' UI tweaks
  - HFS+ file system
  - Mac OS 9 compatibility environment
  - Quartz rendering engine
  - Objective-C UI layer rebranded 'Cocoa'

11

iPhone & iOS

- iPhone
  - Released June 29, 2007
- iPhone OS
  - Port of Mac OS X
  - Shares same developer toolset
  - Developer frameworks adapted and scaled down for mobile device
- iPhone OS 2.0b2
  - March, 2008. Initial release of iPhone OS SDK

DePaul University 7

12

Objective-C Language

- Strict superset of C
  - Mix C with Objective-C
  - Or even C++ with Objective C (usually referred to as Objective C++)
- A very simple language, but some new syntax
- Single inheritance, classes inherit from one and only one superclass
- Protocols define behavior that cross classes
- Dynamic runtime
- Loosely typed

DePaul University 8

13

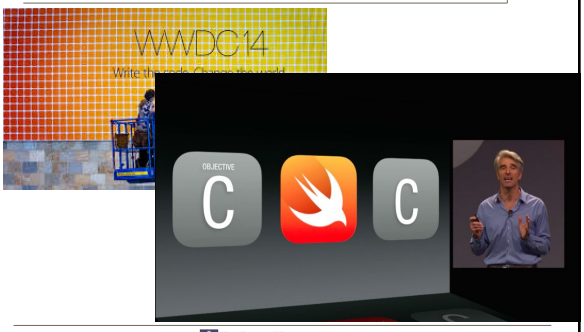
Objective-C 2.0

- Released in October 2007
- New features
  - garbage collection (OS X only)
  - declared and synthesized properties
  - dot syntax
  - fast enumeration
  - etc.
- Clang/LLVM compiler
  - Automated Reference Counting (ARC)
  - Performance improvements

DePaul University 9

14

The Birth of Swift
WWDC 2014, June 2, 2014



DePaul University 10

15

Swift Version History

- Swift beta: June 2, 2014 (Xcode 6.0 beta)
  - Announced at WWDC 2014
- Swift 1.0: September 9, 2014 (Xcode 6.0) — Official stable release
- Swift 2.0: September 21, 2015 (Xcode 7.0) — Development branch
  - Open source (OSX, Linux): Swift.org
- Swift 3: September 13, 2016 (Xcode 8.0) — Source code stability
- Swift 4: September 19, 2017 (Xcode 9.0)
- Swift 4.2: September 17, 2018 (Xcode 10.0)
- Swift 5: March 25, 2019 (Xcode 10.2) — Binary (ABI) compatibility
- Swift 5.2: September 24, 2019 (Xcode 11) — DSL, SwiftUI
- Swift 5.3: September16, 2020 (Xcode 12)

DePaul University 11

21

Hello, Swift!

22

Swift Programming Language

- Modern design and rich features
  - Fast, safe, and powerful
- Syntactic style similar to Java, C++, C#
  - Natural, concise, and readable
- Evolved from Objective-C
  - Not backward compatible, but *interoperable*
  - All Objective-C frameworks are available in Swift

DePaul University 13

24

Hello, World!

- Let's start in Java

```
public class Hello {

    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }

}
```

DePaul University 14

25

Hello, World!

- … and simplify a little more. We have Swift

```
print("Hello, world!");
```

DePaul University 15

28

Hello, World!

- Oh, *one more thing*, the semi-colon is optional

```
print("Hello, world!")
```

DePaul University 16

29

Hello, World!

- We can say hello in Chinese. Just as easy.

```
print("你好，世界!")
```

DePaul University 17

30

Hello, World!

- Or in Emoji!

```
print("👋🌎!")
```

DePaul University 18

31

Running Swift Programs

- Playground
  - Interactive execution environment
  - Learning, experimenting, prototyping
  - Support GUI for iOS and OS X
- REPL (Read-Eval-Print-Loop)
  - Command-line, no GUI support
  - Available in Xcode console or OS X Terminals
- Xcode project
  - Full-blown projects for iOS or OS X

DePaul University 19

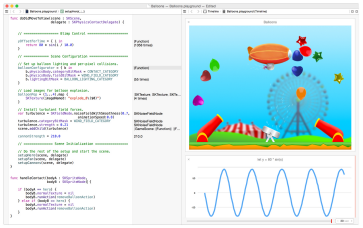
32

The Swift Playground

33

The Xcode Playground

- Introduced in Xcode 6.0
- An interactive execution environment for Swift
  - Learning
  - Exploring
  - Visualizing

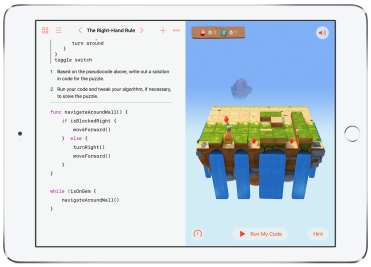


DePaul University 21

34

Playground on iPad

- Introduced in 2016, iOS 10



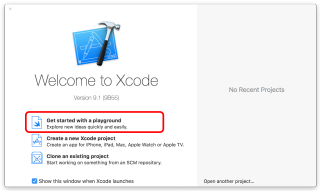
DePaul University 22

35

Launch Playground, Xcode

- Launch Xcode,
  - Select "*Get started with a playground*"
- If Xcode is already running
  - From the Xcode menu bar

File | New | Playground …



DePaul University 23

36

Demo

- Launch Playground in Xcode

DePaul University 24

37

Let's Play

- The starting template

```
// Playground - noun: a place where people can play

import UIKit

var str = "Hello, playground"
```

A comment.

Import the **UIKit** framework (for iOS). Includes the **Foundation** framework

A variable declaration.

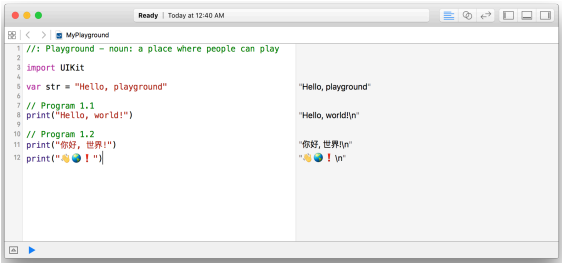
41

The Hello World Program

```
import UIKit

var str = "Hello, playground"

// Program 1.1
print("Hello, world!")

// Program 1.2
print("你好, 世界!")
print("👋🌎!")
```


42

"Hello, World!" In Playground



43

Swift Code in the Playground

- You can edit and run Swift code in a Playground
- Swift code in a Playground may contain
  - Constant and variable declarations
  - Statements
  - Values and expressions
  - Function declarations, and
  - Class declarations, etc.

44

Introduction to Swift

45

Whitespace and Comments

- (Most) Whitespace and comments are ignored
- Single-line comments

```
// This is a single-line comment
```

  - Same as C++, Java
- Multi-line comments

```
/* This is a long multi-line comment …
   It spans across several lines. */
```

  - Same as C, C++, Java
- In Swift, multi-line comments *can be nested.*

```
/* Outer comment begins …
   /* Nested comment … */
   … Outer comment ends */
```


46

Basic Data Types

- Swift provides a rich set of basic data types, include
  `Int` `Double` `String` `Character` `Bool`
- Basic data types are *structs*
  - First class citizens (compared to objects) without overhead
  - Similar syntax and capabilities as classes
  - Values of basic types can be used anywhere objects are expected
    - No conversion to objects necessary
    - No boxing and unboxing necessary
- Basic data types are *value types*
  - Simpler semantics and memory management. Not *ARC managed*

Stay tuned for *structs* and *classes*

Stay tuned for full discussions on *value types* and *ARC*

DePaul University 31

51

The Integer Types

- `Int` `Uint`
  - *Signed* and *unsigned* integers
  - Either 32 or 64 bit integer, based on the platform
- Other integer types (with specific sizes)
  `Int8` `Int16` `Int32` `Int64`
  `UInt8` `UInt16` `UInt32` `UInt64`
- Literals (_ are for readability only, ignored)
  `123` `2014` `-32` `1_000_000`
  `0b0111_1011` `0o173` `0x7B`
- Operators
  `+` `-` `*` `/` `%` `<` `>` `<=` `>=` `==` `!=`

Decimal by default

Binary, octal, and hex-decimal values (equal to decimal 123)

DePaul University 32

58

The Floating-Point Types

- `Double` `Float`
  - 64 and 32 bit floating-point numbers
- Literals
  `3.1415927` `-100.0`
  `1.2345e2` `1E-10`
- Operators
  `+` `-` `*` `/` `<` `>` `<=` `>=` `==` `!=`
  - No remainder operator (%)

DePaul University 33

59

The String and Character Type

- `Character`
  - A single Unicode character. Full Unicode support
- `String`
  - A sequence of Unicode character
- Literals
  `"Hello, World!"` `"A"` `"t"`
- Escape sequences
  `\n` `\r` `\t` `\"` `\'` `\\` `\0`
- Operators
  `+` String concatenation
  `<` `>` `<=` `>=` `==` `!=` Compare string *values*

Single character string and character literals are indistinguishable.

DePaul University 34

64

The Boolean Type

- `Bool`
  - Boolean values
  - Different and incompatible with `Int`
- Literals
  `true` `false`
- Operators
  `&&` `||` `!` `==` `!=`

DePaul University 35

65

Declarations

- Constant declaration
  `let` *Identifier* `:` *Type* (optional) `=` *Initial Value*
- Variable declaration
  `var` *Identifier* `:` *Type* (optional) `=` *Initial Value* (optional)
- Constants are immutable. Variables are mutable.

DePaul University 36

67

Example: Declarations

```
let distanceToMoon = 384_400        // km
let earthGravityAcceleration = 9.8  // m/s/s
let languageName = "Swift"
let swiftIsAwesome = true

var total = 100
var velocity = 30.5
var statusMessage = "Success"
var isComplete = false
```

- Types are omitted, since they can be inferred from the initial values

68

Types of Constants & Variables

- Constants and variables
  - Declaration before use
  - Initialization before access
- Types are optional in declarations
  - Types can be inferred from initial values
- Static typing
  - Types are determined at the time of declaration
    - Explicitly declared or *inferred*
  - Types *cannot* be changed after declaration

69

Explicit Type Declaration

- Explicit type declarations are necessary sometimes
  - when the inferred type is different from the intended type

```
let currencySymbol : Character = "$"

let ten : Int32 = 10
var itemCount : UInt = 0

let length : Float = 100.0
var totalAmount : Double = 0
```


70

Variable Declarations

```
var total : Int
var velocity : Double
var statusMessage : String
var isComplete : Bool

total = 100
velocity = 30.5
statusMessage = "Success"
isComplete = false
```


71

Mixed-Type Expressions

- All common arithmetic and comparison operators are supported
- Mixed-type operations are ***not*** allowed
  - Binary operations on values of the same type only
- No implicit type conversion, a.k.a. coercion
- Type conversion must be explicit

Type Name (*Expression*)

- Numeric literals are convertible to compatible types

72

Mixed-Type Expressions

```
let unitPrice = 1.25          // Double
var quantity = 3              // Int

unitPrice * quantity          // Error

unitPrice * Double(quantity)  // Double

Int(unitPrice) * quantity     // Int
```


76

String Interpolation

- A string literal may contain interpolated expressions of any type
  `\(` *Expression* `)`
- Each interpolated expression is evaluated at runtime and replace by its value (converted to string)

```
let a = 3, b = 5
let result = "\(a) times \(b) is \(a * b)"
```

The output:
```
3 time 5 is 15
```

DePaul University 43

79

Control Statements

80

Control Statements

- Most common control statements are supported
  - The *if* statement, and the *if-else* statement
  - The indefinite loops
    - The *while* loop
    - The *repeat-while* loop
  - The *for-in* loop, a.k.a. *for-each* loop — C-style for-loop is not supported
  - The *switch* statement, enhanced — Stay tuned

DePaul University 45

83

Conditional Statements

Parentheses around the condition are *optional*

Braces for the branches are *required*.

```
var x = 7.5, y = 3.0
if y != 0 {
    print("\(x)/\(y) is \(x/y)")
} else {
    print("\(x)/\(y) is undefined")
}
```

- Similar rules on parentheses and braces for *while*, *repeat-while*, and *for* loops

DePaul University 46

87

For Loop

```
for Variable in Sequence {
    Statements
}
```

- The *Sequence* can be
  - a collection, e.g., an *array*, or
  - a *range*, or
  - a *stride*, or
  - any object that conforms to the *Sequence* protocol.

DePaul University 47

88

Range Expression

- Define a range of consecutive integers
  - *Start index* and *end index* must be integers
  - *Start index* ≤ *end index*
- *Closed* range

  Expression `...` *Expression* — From the *start index* up to and include the *end index*.
- *Half-open* range

  Expression `..<` *Expression* — From the *start index* up to the *end index* - 1.
- Can be one-sided — Stay tuned

DePaul University 48

94

Examples of For Loop

```
let N = 100
var sum = 0
for i in 1 ... N {
    sum += i
}
print("sum = \(sum)")
```

Calculate the sum of 1 + 2 + … + 100

```
var powerOfTwo = 1
for _ in 1 ... 10 {
    powerOfTwo *= 2
}
print(powerOfTwo)
```

Anonymous loop variable _

DePaul University 49

98

Examples of For Loop

```
for i in 0..<10 {
    print(i)
}
```

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

```
for i in (1...10).reversed() {
    print(i)
}
```

10, 9, 8, 7, 6, 5, 4, 3, 2, 1

DePaul University 50

99

Array Literals

- An *array* is an ordered collection of values of a uniform type
  - Not the same as arrays in C++ and Java
  - Can be variable size
    - Can grow and shrink as needed
  - Behave like a *List* in Java
- Array literals

[$Expression_1$, $Expression_2$, …]

DePaul University 51

100

Example: Linear Search

```
let cities = [
  "London", "Berlin", "Madrid", "Rome", "Paris", … ,
  "Marseille", "Amsterdam", "Valencia", "Zagreb" ]
let searches = [ "Rome", "Venice", "Barcelona", "Sevilla" ]
for city in searches {
    var found = false
    for c in cities {
        if c == city {
            found = true
            break
        }
    }
    print(city, "is", found ? "found" : "not found")
}
```

Terminate the *immediate* enclosing loop

DePaul University 52

102

Advanced Topics

103

Using Stride Function

- Generate a sequence with: *start, end, stride*
  - *start, start + stride, start + 2 * stride, … , start + n * stride*
  - To: up to but not include *end*. Through: may include *end*.

```
for i in stride(from: 1, to: 10, by: 3) {
    print(i)
}
```

1, 4, 7

```
for i in stride(from: 1, through: 10, by: 3) {
    print(i)
}
```

1, 4, 7, 10

```
for i in stride(from: 3, to: 0, by: -1) {
    print(i)
}
```

3, 2, 1

```
for i in stride(from: 0.0, through: Double.pi, by: Double.pi/4) {
    print(i)
}
```

$0.0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi$

DePaul University 54

112

Switch Statement

- Empty case not permitted
- Default required unless cases are exhaustive
- Break implied after each case

```
for month in 1 ... 12 {
    switch month {
    case 1:  print("January")
    case 2:  print("February")
    case 3:  print("March")
    case 4:  print("April")
    case 5:  print("May")
    case 6:  print("June")
    case 7:  print("July")
    case 8:  print("August")
    case 9:  print("September")
    case 10: print("October")
    case 11: print("November")
    case 12: print("December")
    default: print("Invalid month")
    }
}
```

DePaul University 55

113

Switch Statement

```
let months = [ "January", "February", … , "December" ]
for month in months {
    let num: Int
    switch month {
    case "January":   num = 1
    case "February":  num = 2
    …
    case "December":  num = 12
    default:          num = 0
    }
    let suffix: String
    switch num {
    case 1:  suffix = "st"
    case 2:  suffix = "nd"
    case 3:  suffix = "rd"
    default: suffix = "th"
    }
    print("\(month) is the \(num)\(suffix) month")
}
```

Output:

```
January is the 1st month
February is the 2nd month
March is the 3rd month
April is the 4th month
May is the 5th month
June is the 6th month
July is the 7th month
August is the 8th month
September is the 9th month
October is the 10th month
November is the 11th month
December is the 12th month
```

DePaul University 56

114

Switch Statement

- Case for multiple values: comma separated list
- Support other types, conditions, pattern matching, etc.

```
for month in months {
    let season: String
    switch month {
    case "March", "April", "May":             season = "spring"
    case "June", "July", "August":            season = "summer"
    case "September", "October", "November": season = "autumn"
    case "December", "January", "February":  season = "winter"
    default: season = "";
    }
    print("The month \(month) is in the meteorological \(season)")
}
```

DePaul University 57

115

Switch Statement

- Break is implied after each case

```
for i in 0 ... 4 {
    switch i {
    case 1: print("\t  *")
    case 2: print("\t ***")
    case 3: print("\t*****")
    default: print()
    }
}
```

DePaul University 58

116

Switch Statement

- Fall-through cases must be explicit

```
for i in 0 ... 4 {
    print("i = ", i)
    switch i {
    case 1: print("\t  *")
            fallthrough
    case 2: print("\t ***")
            fallthrough
    case 3: print("\t*****")
    default: print()
    }
}
```

DePaul University 59

118

Next …

- More Swift
- Functions
- Classes and objects

✧ Xcode, iOS, WatchOS are trademarks of Apple Inc.

DePaul University 60

120