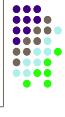


CSC 471 / 371
Mobile Application Development for iOS




Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
[@DePaulSWEEng](https://twitter.com/DePaulSWEEng)

Outline

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



DEPAUL UNIVERSITY

Objective-C
Programming Language



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





DEPAUL UNIVERSITY

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



DEPAUL UNIVERSITY

From NeXTSTEP to Mac OS X

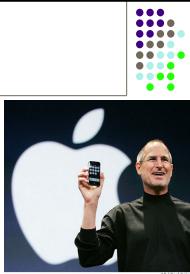
- Apple acquired NeXT Inc.
- Mac OS X 10.0
 - March, 2001, Initial release of OS X
- Based on NeXTSTEP
 - Mac Toolbox APIs
 - ‘Mac-like’ UI tweaks
 - HFS+ file system
 - Mac OS 9 compatibility environment
 - Quartz rendering engine
 - Objective-C UI layer rebranded ‘Cocoa’



DEPAUL UNIVERSITY

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

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

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

The Birth of Swift WWDC 2014, June 2, 2014



DEPAUL UNIVERSITY

10

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)
- Swift 2.0: September 21, 2015 (Xcode 7.0)
 - Announced at WWDC 2015, June 8, 2015
 - Open source (OSX, Linux): Swift.org
- Swift 3: September 13, 2016 (Xcode 8.0)
- Swift 4: September 19, 2017 (Xcode 9.0)

DEPAUL UNIVERSITY

11

Hello, Swift!



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

Hello, World!



- ... and simplify a little more. We have Swift

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

DEPAUL UNIVERSITY

15

Hello, World!



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

```
print("你好, 世界!")
```

DEPAUL UNIVERSITY

17

Hello, World!



- Let's start in Java

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

DEPAUL UNIVERSITY

14

Hello, World!



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

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

DEPAUL UNIVERSITY

16

Hello, World!



- Or in Emoji!

```
print("👋🌍! ")
```

DEPAUL UNIVERSITY

18

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

The Swift Playground



The Xcode Playground

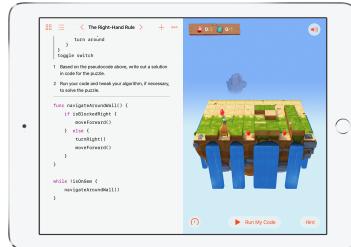
- Introduced in Xcode 6.0
- An interactive execution environment for Swift
 - Learning
 - Exploring
 - Visualizing

DEPAUL UNIVERSITY

21

Playground on iPad

- Introduced in 2016, iOS 10

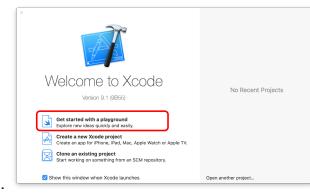


DEPAUL UNIVERSITY

22

Launch Playground, Xcode

- Launch Xcode,
 - Select “Get started with a playground”
 - If Xcode is already running
 - From the Xcode menu bar
- File | New | Playground ...

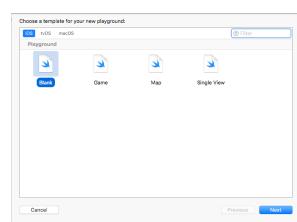


DEPAUL UNIVERSITY

23

Launch Playground, Xcode

- Choose a name
- Choose a platform:
 - “iOS”, “macOS”, or “tvOS”
 - Choose “iOS”
 - You may choose any of the three, for simple programs without UI
- Click “Next”

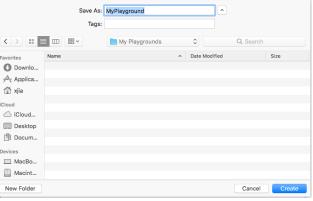


DEPAUL UNIVERSITY

24

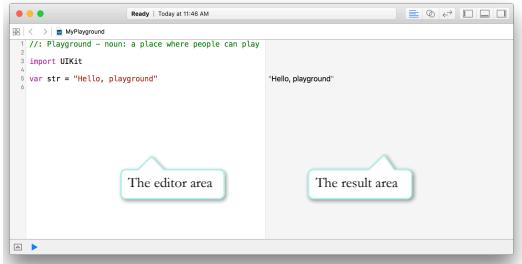
Launch Playground, Xcode

- Choose a folder to save the playground file
- Playground files are saved with the .playground suffix
- Click “Create”



DEPAUL UNIVERSITY 25

New Playground



The editor area The result area

DEPAUL UNIVERSITY 26

Let's Play

- The starting template
 - A comment.
 - Import the **UIKit** framework (for iOS). Includes the **Foundation** framework
 - A variable declaration.



DEPAUL UNIVERSITY 27

The Hello World Program

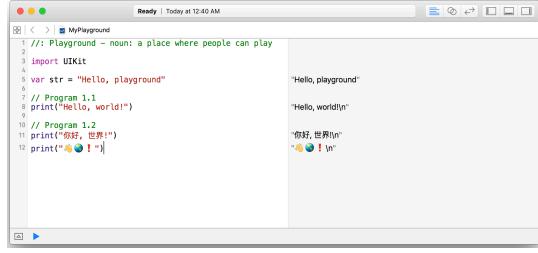
```
import UIKit
var str = "Hello, playground"

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

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

DEPAUL UNIVERSITY 28

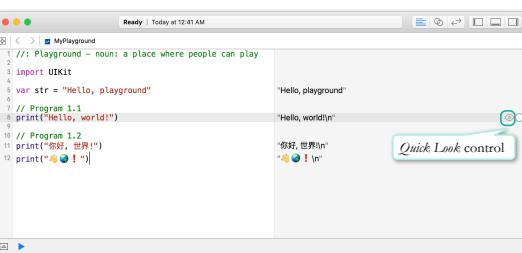
“Hello, World!” In Playground



DEPAUL UNIVERSITY 29

Quick Look

- Left: “Quick Look” control



DEPAUL UNIVERSITY 30

Quick Look Popup

- A popup displaying the results

The screenshot shows a Mac OS X desktop with an Xcode playground window titled 'MyPlayground'. A blue 'Quick Look' popup is overlaid on the right side of the screen, containing the text 'Hello, world!'. The main playground window shows the same text in its output pane.

Show Inline Results

- Right: “Show Result”

The screenshot shows an Xcode playground window with the 'Show Result' control highlighted with a red box. The control is located in the top right corner of the playground's output area. The playground displays the text 'Hello, playground' and 'Hello, world!'.

Inline Results

- Click on the “Show Result” control

The screenshot shows an Xcode playground window with three separate 'Inline result' popups displayed. Each popup contains the text 'Hello, world!', which corresponds to the code 'print("Hello, world!")' in the playground. The 'Show Result' control is also highlighted with a red box.

The Debug Area

The screenshot shows an Xcode playground window with the 'Show/hide the debug area' button highlighted with a red box. The 'The debug area' section is visible at the bottom of the playground window, showing the text 'Hello, playground' and 'Hello, world!'.

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.

The screenshot shows an Xcode playground window with Swift code being edited. The code includes comments like '// Program 1.1' and '// Program 1.2', and prints 'Hello, playground' and 'Hello, world!' respectively. The playground's output pane shows the results of the code execution.

Introduction to Swift

This slide is a blank white page with a decorative graphic in the top right corner consisting of a grid of colored dots.

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 */
```

DEPAUL UNIVERSITY

37

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
```

Decimal by default

```
0b0111_1011  0o173  0x7B
```

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


```
+  -  *  /  %  <  >  <=  >=  ==  !=
```

DEPAUL UNIVERSITY

39

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 full discussions on *value types* and *ARC*

DEPAUL UNIVERSITY

38

The String and Character Type

- Character**
 - A single Unicode character. Full Unicode support
- String**
 - A sequence of Unicode character
- Literals


```
"Hello, World!"  "A"  "t"
```

Single character string and character literals are indistinguishable.
- Escape sequences


```
\n  \r  \t  \"  \'  \\  \0
```
- Operators


```
+  String concatenation
<  >  <=  >=  ==  !=
```

Compare string values 

DEPAUL UNIVERSITY

41

The Boolean Type

- Bool**
 - Boolean values
 - Different and incompatible with `Int`
- Literals


```
true  false
```
- Operators


```
&&  ||  !  ==  !=
```

DEPAUL UNIVERSITY

42

Declarations

- Constant declaration

```
let Identifier : Type = Initial Value
    optional
```

- Variable declaration

```
var Identifier : Type = Initial Value
    optional
```

- Constants are immutable. Variables are mutable.

DEPAUL UNIVERSITY

43

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

DEPAUL UNIVERSITY

44

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

DEPAUL UNIVERSITY

45

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
```

DEPAUL UNIVERSITY

46

Variable Declarations

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

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

DEPAUL UNIVERSITY

47

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

DEPAUL UNIVERSITY

48

Mixed-Type Expressions

```
let unitPrice = 1.25    Double
var quantity = 3        Int
unitPrice * quantity   Error
unitPrice * Double(quantity) Double
Int(unitPrice) * quantity Int
```

DEPAUL UNIVERSITY

49

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

50

Control Statements

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

52

Conditional Statements

Parentheses around the condition are optional

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

Braces for the branches are required.

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

DEPAUL UNIVERSITY

53

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

54

Range Expression

- Define a range of consecutive integers
 - Start index* and *end index* must be integers
 - Start index* \leq *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

55

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

56

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

57

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

58

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

59

Advanced Topics

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)
}
for i in stride(from: 1, through: 10, by: 3) {
    print(i)
}
for i in stride(from: 3, to: 0, by: -1) {
    print(i)
}
for i in stride(from: 0.0, through: Double.pi, by: Double.pi/4) {
    print(i)
}

```

DEPAUL UNIVERSITY 61

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 62

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")
}

```

DEPAUL UNIVERSITY 63

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

```

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 64

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 65

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 66

Next ...



- More Swift
- Functions
- Classes and objects

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

 DEPAUL UNIVERSITY

67