캡스톤디자인 I (Swift Programming Language)

구명완교수

Office: K관 239호

Email: mwkoo9@gmail.com



Swift Programming Language

- Swift is now open source (Dec. 3, 2015)
 - Public source code repositories at github.com/apple
 - Apple has a new home on GitHub located at github.com/apple where you can find all the source code for the Swift project
 - https://swift.org
 - https://developer.apple.com/swift/resources/

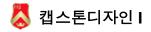
Xcode 7 + Swift 2

Download Xcode 7 and get started with Swift 2.0. Use the migrator in Xcode 7 to convert your existing Swift code to use the new Swift 2.0 features and syntax.





- A surprise announcement at Apple's WWDC (Worldwide Developer Conference) in June 2014
- Apple's Language of the Future
 - Language of the future for app and systems programming
- Popular Language Features
 - Simpler syntax than Objective-C.
 - Type inference, tuples, closures (lambdas), generics, operator overloading, functions with multiple return values, optionals, String interpolation, switch statement enhancements
- Performance
 - about 1.5 times faster than Objective-C code on today's multi-core systems



Swift Programming Language

- Error Prevention: eliminates many common programming errors
 - automatic memory management, no pointers, required braces around every control statement's body
 - assignment operators that do not return values,
 - requiring initialization of all variables and constants
 - array bounds checking,
 - automatic checking for overflow of integer calculations
- Interoperability with Objective-C
 - Combine Swift and Objective-C in the same app
 - To enhance existing Objective-C apps without having to rewrite all the code
- Playgrounds: an Xcode window in which you can enter Swift code that compiles and executes as you type it



Explosive Growth of the iPhone and iPad

- First-generation iPhone, released in June 2007
 - sold 6.1 million units in its initial five quarters of availability
- The iPhone 5s and the iPhone 5c, in September 2013
 - nine million combined in the first three days of availability
- iPhone 6 and iPhone 6 Plus, announced in September 2014
 - 10 million units combined in their first weekend of availability
- The first generation iPad, launched in April 2010
 - 3 million units in its first 80 days of availability6 and over 40 million worldwide by September 2011
- The iPad mini, the iPad Air in November 2013.
 - In just the first quarter of 2014, Apple sold a record 26 million iPads
- 1.3 million apps in the App Store and over 75 billion iOS apps

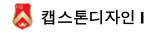




- The Swift Programming Language—available in the iBooks store and at:
 - https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/
- Using Swift with Cocoa and Objective-C—available in the iBooks store and at:
 - https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps
- The Swift Standard Library Reference:
 - https://developer.apple.com/library/ios/documentation/General/Reference/SwiftStan dardLibraryReference
- The Swift Blog:
 - https://developer.apple.com/swift/blog/
- World Wide Developers Conference (WWDC) 2015 videos:
 - https://developer.apple.com/videos/wwdc/2015/

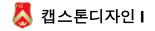
Contents I

- Basics
- Basic Operators
- Strings and Characters
- Collection Types
- Control Flow
- Functions
- Closures
- Enumerations
- Classes and Structures
- Properties
- Methods



Contents II

- Subscripts
- Inheritance
- Initialization
- Deinitialization
- Automatic Reference Counting
- Optional Chaining
- Type Casting
- Nested Types
- Extensions
- Protocols
- Generics





- Swift
 - a new programming language for iOS and OS X, watchOS app development
 - familiar from your experience of developing in C and Objective-C
- Characteristics
 - provides its own versions of all fundamental C and Objective-C types
 - Int, Double, Float, Bool, String
 - provides powerful versions of the two primary collection types
 - Array, Set, Dictionary

- Variables
 - To store and refer to values by an identifying name
- Constants
 - Extensive use of variables whose values cannot be changed
- Tuples, not found in Objective-C
 - Tuples enable you to create and pass around groupings of values
 - use a tuple to return multiple values from a function as a single compound value
- Optional types
 - handle the absence of a value
 - say either "there is a value, and it equals x"
 - or "there isn't a value at all



- Constants and Variables
 - Declaring Constants and Variables
 - declare constants with the let keyword
 - let maximumNumberOfLoginAttempts = 10
 - declare variables with the var keyword
 - var currentLoginAttempt = 0
 - declare multiple constants or multiple variables on a single line, separated by commas
 - var x = 0.0, y = 0.0, z = 0.0

- Type annotation
 - the constant or variable name + a colon + space + the name of the type to use
 - var welcomeMessage: String
 - Define multiple related variables of the same type on a single line
 - var red, green, blue: Double
- Naming Constants and Variables
 - Can contain almost any character, including Unicode characters
 - Cannot contain whitespace characters, mathematical symbols, arrows, private-use (or invalid) Unicode code points, or line- and box-drawing characters, Not begin with a number

- Can change the value of an existing variable
 - var friendlyWelcome = "Hello!"
 - friendlyWelcome = "Bonjour!"
- Cannot change the value of a constant
 - let languageName = "Swift"
 - languageName = "Swift++" → Error

- Printing Constants and Variables
 - print(_:separator:terminator:)
 - print(friendlyWelcome)
 - // prints "Bonjour!"
 - print("Welcome to","Paris",separator:" ", terminator:" ")
 - print(" Swift Language")
 - // prints "Welcome to Paris Swift Language "
 - print("This is a string", terminator:" ")
 - String interpolation
 - print("The current value of friendlyWelcome is \((friendlyWelcome)\)")
 - // prints "The current value of friendlyWelcome is Bonjour!"

- Comments
 - Single-line comments begin with two forward-slashes (//):
 - // this is a comment
 - Multiline comments start with (/*), end with (*/):
 - /* this is also a comment,
 - but written over multiple lines */
 - Can be nested inside other multiline comments
 - /* this is the start of the first multiline comment
 - /* this is the second, nested multiline comment */
 - this is the end of the first multiline comment */
- Semicolons
 - not require you to write a semicolon (;) after each statement

- require semicolon to write multiple statements on a single line
- let cat = "**\'**"; print(cat)
- Integers
 - provides signed and unsigned integers in 8, 16, 32, and 64 bit forms
 - 8-bit unsigned integer: UInt8, 32-bit signed integer: Int32,
 - access the minimum and maximum values of each integer type with its min and max properties
 - let minValue = UInt8.min // minValue = 0, UInt8
 - let maxValue = UInt8.max // maxValue = 255, type UInt8
 - Int, UInt; the current platform's native word size



- Floating-point numbers
 - Double represents a 64-bit floating-point number
 - precision of at least 15 decimal digits
 - Float represents a 32-bit floating-point number
 - precision of at least 6 decimal digits
- Type Safety and Type Inference
 - Swift is a type safe language
 - performs type checks when compiling your code



- Type inference
 - deduce the type of a particular expression automatically
 - requires far fewer type declarations than languages such as C or Objective-C.
 - useful when you declare a constant or variable with an initial value
 - let meaningOfLife = 42 // Deduce Int
 - let pi = 3.14159 // inferred to be of type Double
 - let anotherPi = 3 + 0.14159 // inferred to be of type Double



- Numeric Literals
 - let decimalInteger = 17
 - let binaryInteger = 0b10001 // 17 in binary notation
 - let octalInteger = $\frac{00}{21}$ // 17 in octal notation
 - let hexadecimalInteger = 0x11 // 17 in hexadecimal notation

- Numeric Literals
 - decimal numbers with an exponent of exp, 10^{exp}
 - 1.25e2 means 1.25×10^2 , or 125.0.
 - 1.25e-2 means 1.25×10^{-2} or 0.0125
 - For hexadecimal numbers with an exponent of exp, 2^{exp}
 - 0xFp2 means 15×2^2 60.0
 - 0xFp-2 means 15×2^{-2} 3.75
 - extra formatting to make them easier to read
 - let oneMillion = 1_000_000
- Numeric Type Conversion
 - Use the Int type for all general-purpose integer constants and variables in your code

- Integer Conversion
 - To convert one specific number type to another,
 - initialize a new number of the desired type with the existing value
 - let twoThousand: UInt16 = 2_000
 - let one: UInt8 = 1
 - let twoThousandAndOne = twoThousand + UInt16(one)
- Conversions between integer and floating-point numeric types
 - must be made explicit:
 - let three = 3
 - let pointOneFourOneFiveNine = 0.14159
 - let pi = Double(three) + pointOneFourOneFiveNine
 - // pi equals 3.14159, and is inferred to be of type Double
 - Floating-point to integer conversion
 - let integerPi = Int(pi) // integerPi equals 3

- Type Aliases
 - Define an alternative name for an existing type
 - typealias AudioSample = UInt16
 - var maxAmplitudeFound = AudioSample.min
 - // maxAmplitudeFound is now 0

Booleans

- a basic Boolean type, called Bool
- Boolean values; true and false:
 - let orangesAreOrange = true
 - let turnipsAreDelicious = false
 - if turnipsAreDelicious { println("Mmm, tasty turnips!") }
 - else { println("Eww, turnips are horrible.") }

- Swift's type safety prevents non-Boolean values from being substituted for Bool
 - Error Case:
 - let i = 1
 - if i { // this example will not compile, and will report an error}
 - True Case:
 - let i = 1
 - if i == 1 { // this example will compile successfully }
 - The result of the i == 1 comparison is of type Bool,
 - so this example passes the type-check

- Tuples
 - Group multiple values into a single compound value
 - The values within a tuple can be of any type
 - let http404Error = (404, "Not Found")
 - // http404Error is of type (Int, String), and equals (404, "Not Found")
 - Decompose a tuple's contents into separate constants or variables
 - let (statusCode, statusMessage) = http404Error
 - print("The status code is \((statusCode)\)")
 - // prints "The status code is 404"
 - print("The status message is \((statusMessage)\)")
 - // prints "The status message is Not Found

- Access using index numbers starting at zero
 - print("The status code is \((http404Error.0)")
 - // prints "The status code is 404"
- Name when the tuple is defined
 - let http200Status = (statusCode: 200, description: "OK")
 - print("The status code is \((http200Status.statusCode)\)")
 - // prints "The status code is 200"
- Tuples are useful for temporary groups of related values
 - If your data structure is likely to persist beyond a temporary scope
 - model it as a class or structure,

- Optionals (?)
 - use optionals in situations where a value may be absent
 - There is a value, and it equals x
 - or
 - There isn't a value at all
 - nil in Objective-C
 - "the absence of a valid object
 - Int?: meaning that it might contain some Int value, or it might contain no value at all. ?: meaning optional
 - Ex: Swift's String type has a method called toInt,
 - However, not every string can be converted into an integer
 - let possibleNumber = "123"

- let convertedNumber = possibleNumber.toInt()
- // convertedNumber : "Int?", or "optional Int"
- Because the tolnt method might fail, it returns an optional Int, rather than an Int

nil

- set to a valueless state
 - var serverResponseCode: Int? = 404
 - // serverResponseCode contains an actual Int value of 404
 - serverResponseCode = nil
 - // serverResponseCode now contains no value
- an optional constant or variable without a default value
 - var surveyAnswer: String? // automatically set to nil

- Swift's nil is not the same as nil in Objective-C.
 - In Objective-C, nil is a pointer to a nonexistent object.
 - In Swift, the absence of a value of a certain type
 - Optionals of any type can be set to nil, not just object types
- If Statements and Forced Unwrapping
 - forced unwrapping of the optional's value by (!)
 - if convertedNumber != nil {
 println("convertedNumber has an integer value of
 \(convertedNumber!).") }
 - // prints "convertedNumber has an integer value of 123."
 - Trying to use! to access a non-existent optional value triggers a runtime error.

- Optional binding
 - if let constantName = someOptional

```
{ statements }
```

- to find out whether an optional contains a value
 - if let actualNumber = possibleNumber.toInt()
 { println("\'\(possibleNumber)\' has an integer value of \(actualNumber)\")
 - // set a new constant called actualNumber to the value contained in the optional

```
} else { println("\'\(possibleNumber)\' could not be converted to an integer") }
```

– // prints "'123' has an integer value of 123"

- Implicitly unwrapped optionals
 - are useful when an optional will always have a value, after that value is first set.
 - Difference in behavior between an optional string and an implicitly unwrapped optional string
 - let possibleString: String? = "An optional string."
 - let forcedString: String = possibleString! // requires an exclamation mark

- let assumedString: String! = "An implicitly unwrapped optional string."
- let implicitString: String = assumedString // no need for an exclamation mark

- treat an implicitly unwrapped optional like a normal optional
 - if assumedString != nil {
 - println(assumedString) }
 - // prints "An implicitly unwrapped optional string."
- an implicitly unwrapped optional with optional binding
 - if let definiteString = assumedString {
 println(definiteString)
 - **—** }
 - // prints "An implicitly unwrapped optional string."

- Error Handling
 - allows you to determine the underlying cause of failure, and, if necessary, propagate the error to another part of your program
 - A function indicates that it can throw an error by including the throws keyword in its declaration
 - prepend the try keyword to the expression
 - func canThrowAnError() throws {// this function may or may not throw an error }
 - do {try canThrowAnError()// no error was thrown
 - } catch {// an error was thrown }

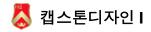
- Example
 - func makeASandwich() throws { // ... }
 - do {try makeASandwich()eatASandwich()
 - } catch Error.OutOfCleanDishes { washDishes()
 - } catch Error.MissingIngredients(let ingredients) {
 buyGroceries(ingredients)
 - }



- Assertions
 - If the condition evaluates to true, code execution continues as usual; if the condition evaluates to false, code execution ends, and your app is terminated.
 - provide a suitable debug message
 - let age = -3
 - assert(age >= 0, "A person's age cannot be less than zero")
 - display Message if the result of the condition is false
 - Use an assertion whenever a condition has the potential to be false



- Swift supports most standard C operators and improves several capabilities to eliminate common coding errors
 - assignment operator (=) does not return a value, to prevent it from being mistakenly used when the equal to operator (==) is intended
 - Arithmetic operators (+, -, *, /, % and so forth) detect and disallow value overflow, to avoid unexpected results
 - perform remainder (%) calculations on floating-point numbers
 - provides two range operators (a..<b and a...b) not found in C
- Terminology
 - Unary operators; -a, i++,
 - Binary operators; 2+3
 - Ternary operators ; a ? b : c

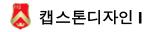


- Assignment Operator
 - decomposed into multiple constants or variables at once
 - let (x, y) = (1, 2)
 - // x is equal to 1, and y is equal to 2
 - does not itself return a value
 - if x = y {
 - // this is not valid, because x = y does not return a value }
 - helps you to avoid these kinds of errors in your code(==)
- Arithmetic Operators; +, -, *, /
 - not allow values to overflow by default
 - String concatenation for addition operator
 - Floating-Point Remainder Calculations; 8 % 2.5 // equals 0.5

- let b = ++a increments a before returning its value
- let c = a++ increments a after returning its value
- Compound Assignment Operators
 - compound assignment operators that combine assignment (=) with another operation
 - a += 2
- Comparison Operators
 - Equal to (a == b), Not equal to (a != b)
 - Greater than (a > b), Less than (a < b)
 - Greater than or equal to $(a \ge b)$, Less than or equal to $(a \le b)$
 - identity operators (=== and !==),
 - test whether two object references both refer to the same object instance



- Ternary Conditional Operator
 - question ? answer1 : answer2
 - If question is true, it evaluates answer1 and returns its value; otherwise, it evaluates answer2 and returns its value
- Nil Coalescing Operator (a ?? b)
 - Unwraps an optional a if it contains a value, or returns a default value b if a is nil
 - The expression a is always of an optional type
 - The expression b must match the type that is stored inside a
 - shorthand for the code , a != nil ? a! : b
 - access the value wrapped inside a when a is not nil, and to return b otherwise



- Ex: uses the nil coalescing operator to choose between a default color name and an optional user-defined color name:

 - userDefinedColorName = "green"
 colorNameToUse = userDefinedColorName ?? defaultColorName
 // userDefinedColorName is not nil, so colorNameToUse is set to "green"

- Range Operators
 - Closed range operator (a...b)
 - defines a range that runs from a to b
 - a must not be greater than b
 - for index in 1...5 {
 - println("\(index) times 5 is \(index * 5)") } //1,2,.. 5
 - Half-open range operator (a..<b)
 - defines a range that runs from a to b, not include b
 - let names = ["Anna", "Alex", "Brian", "Jack"]
 - let count = names.count // count=4
 - for i in 0..<count {</pre>
 - println("Person \(i + 1\) is called \(names[i])") \} // 0,1,2,3

- Logical Operators
 - three standard logical operators found in C-based languages:
 - Logical NOT (!a), Logical AND (a && b), Logical OR (a || b)
 - Explicit Parentheses
 - let knowsOverridePassword = true ; let hasDoorKey = false
 - let enteredDoorCode = true ; let passedRetinaScan = false
 - if (enteredDoorCode && passedRetinaScan) || hasDoorKey || knowsOverridePassword {
 println("Welcome!")
 } else {
 println("ACCESS DENIED")
 - // prints "Welcome!"