iOS Application Development

Lecture 3: Unit 2

Simon Völker & Philipp Wacker
Media Computing Group
RWTH Aachen University

hci.rwth-aachen.de/ios
## Seminar Groups

<table>
<thead>
<tr>
<th>Topic</th>
<th>Group</th>
<th>Presentation</th>
<th>Discussions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Data</td>
<td>Group 10</td>
<td>3.12.19, first group</td>
<td>26.11.19</td>
</tr>
<tr>
<td>Scene Kit</td>
<td>Group 7</td>
<td>26.11.19, first group</td>
<td>19.11.19</td>
</tr>
<tr>
<td>Core ML + Create ML</td>
<td>Group 12</td>
<td>3.12.19, second group</td>
<td>26.11.19</td>
</tr>
<tr>
<td>SiriKit</td>
<td>Group 2</td>
<td>19.11.19, second group</td>
<td>12.11.19</td>
</tr>
<tr>
<td>WatchOS</td>
<td>Group 14</td>
<td>25.11.19, second group</td>
<td>18.11.19</td>
</tr>
<tr>
<td>MapKit, CoreLocation</td>
<td>Group 8</td>
<td>18.11.19, first group</td>
<td>11.11.19</td>
</tr>
<tr>
<td>Networking on iOS</td>
<td>Group 6</td>
<td>2.12.19, first group</td>
<td>25.11.19</td>
</tr>
<tr>
<td>Extensions &amp; Inter-App communication</td>
<td>Group 5</td>
<td>19.11.19, first group</td>
<td>12.11.19</td>
</tr>
<tr>
<td>Debugging in Xcode and Instruments</td>
<td>Group 4</td>
<td>18.11.19, second group</td>
<td>11.11.19</td>
</tr>
<tr>
<td>ClassKit</td>
<td>Group 3</td>
<td>2.12.19, second group</td>
<td>25.11.19</td>
</tr>
<tr>
<td>RealityKit &amp; Reality Composer</td>
<td>Group 1</td>
<td>09.12.19, first group</td>
<td>2.12.19</td>
</tr>
<tr>
<td>Bringing People into AR</td>
<td>Group 13</td>
<td>09.12.19, second group</td>
<td>25.11.19</td>
</tr>
<tr>
<td>GameplayKit</td>
<td>Group 9</td>
<td>26.11.19, second group</td>
<td>19.11.19</td>
</tr>
<tr>
<td>SwiftUI</td>
<td>Group 11</td>
<td>25.11.19, first group</td>
<td>18.11.19</td>
</tr>
</tbody>
</table>
Recap

• Basics of Swift

• 2 Types of Variables
  • let: constant
  • var: variable

• Basics of XCode
  • IBOutlets
  • IBActions
XCode Debugging

• Set breakpoints for execution on simulator and device

• Continue, Step over, Step into, Step out
Online SwiftPlayground

http://online.swiftplayground.run/
Strings

• A unicode string value that is a collection of characters

```swift
var string = "hallo"
string.count // -> 5
```

• Individual characters are of type `Character`. But since Strings are more common => single characters are also of type `String`

```swift
let a = "a" // 'a' is a String
let b: Character = "b" // 'b' is a Character"
```
Strings

• Check if String is empty:

```swift
var myString = ""

if myString.isEmpty {
    print("The string is empty")
}
```

• Concatenation and Interpolation

```swift
let string1 = "Hello"
let string2 = ", world!"
let myString = string1 + string2 // "Hello, world!"
var mySecondString = myString + ", world!" // "Hello, world!"
mySecondString += " Hello!" // "Hello, world! Hello!"

let a = 4
let b = 5
print("If a is \(a) and b is \(b), then a + b equals \(a+b)")
// "If a is 4 and b is 5, then a + b equals 9"
Strings

• Equality and Comparison:

```swift
let month = "January"
let otherMonth = "January"
let lowercaseMonth = "january"

if month == otherMonth {
    print("They are the same") //Output: “They are the same.”
}

if month != lowercaseMonth {
    print("They are not the same.") //Output: “They are not the same.”
}

let name = "Johnny Appleseed"
if name.lowercased() == "joHnnY aPPleseeD".lowercased() {
    print("The two names are equal.") // Output: The two names are equal.
}
```
Functions

• Basic function definition

```swift
func functionName (parameters) -> ReturnType {
    //body of the function
}
```

• Function without parameters and ReturnType

```swift
func displayPi() {
    print("3.1415926535")
}
displayPi() //Output: 3.1415926535
```
Functions

• Function with one parameter

```swift
func triple(value: Int) {
    let result = value * 3
    print("If you multiply \(value) by 3, you'll get \(result).")
}
triple(value: 10) //Output: “If you multiply 10 by 3, you'll get 30.
```

• Function with multiple parameters

```swift
func multiply(firstNumber: Int, secondNumber: Int) {
    let result = firstNumber * secondNumber
    print("The result is \(result).")
}
multiply(firstNumber: 10, secondNumber: 5) //Output: The result is 50.
```
Functions

• Argument labels

```swift
func sayHello(to person: String, and anotherPerson: String) {
    print("Hello \\
        \(person) and \\
        \(anotherPerson)")
}

sayHello(to: "Miles", and: "Riley")

func sayHello(_: person: String, _ anotherPerson: String) {
    print("Hello \\
        \(person) and \\
        \(anotherPerson)")
}

sayHello("Luke", "Dave")
```
Functions

• Default parameter values

```swift
func display(teamName: String, score: Int = 0) {
    print("\(teamName): \(score)")
}

display(teamName: "Wombats", score: 100) //"Wombats: 100"
display(teamName: "Wombats") //"Wombats: 0"
```
Functions

• Return Values:

```swift
func multiply(firstNumber: Int, secondNumber: Int) -> Int {
    let result = firstNumber * secondNumber
    return result
}

let myResult = multiply(firstNumber: 10, secondNumber: 5)
//myResult = 50
print("10 * 5 is \(myResult)")
```
Functions

• Returning Multiple Values using Tuples:

```
func tupleReturn() -> (Int, String, Double)
{
    return (1, "String", 2.0)
}

let myTuple = tupleReturn()
print("This is my tuple: \(myTuple.0), \(myTuple.1), \(myTuple.2)")
```
**Structs**

- Most basic Swift types are Structs:
  - String, Int, Double, Bool

- Value types (structs) vs reference types (classes)

- Basic struct:

  ```swift
  struct Person {
    var name: String
  }

  let firstPerson = Person(name: "Jasmine")
  print(firstPerson.name) //Output: Jasmine
  ```
Structs

- Struct methods:

```swift
struct Person {
    var name: String
    func sayHello() {
        print("Hello, there! My name is \(name)!")
    }
}

let person = Person(name: "Jasmine")
person.sayHello() // Output: “Hello, there! My name is Jasmine!”
```
Structs

- Initializers:

```swift
var string = String.init() // ""
var integer = Int.init() // 0
var bool = Bool.init() // false

// Shorthand version of the default initializer:
var string = String() // ""
var integer = Int() // 0
var bool = Bool() // false

struct Odometer {
    var count: Int = 0
}

let odometer = Odometer() // default value for count = 0
let odometer = Odometer(count: 27000) // set count value
```
Structs

• Custom initializers:

```swift
struct Temperature {
    var celsius: Double

    init(celsius: Double) {
        self.celsius = celsius
    }

    init(fahrenheit: Double) {
        celsius = (fahrenheit - 32) / 1.8
    }
}

let currentTemperature = Temperature(celsius: 18.5) // celsius: 18.5
let boiling = Temperature(fahrenheit: 212.0) // celsius: 100.0
```
struct Odometer {
    var count: Int = 0 // Assigns a default value to the `count` property.

    mutating func increment() {
        count += 1
    }

    mutating func increment(by amount: Int) {
        count += amount
    }

    mutating func reset() {
        count = 0
    }
}

var odometer = Odometer() // odometer.count defaults to 0
odometer.increment() // odometer.count is incremented to 1
odometer.increment(by: 15) // odometer.count is incremented to 16
odometer.reset() // odometer.count is reset to 0
Structs

• Computed properties:

```swift
struct Temperature {
    var celsius: Double

    var fahrenheit: Double {
        return celsius * 1.8 + 32
    }

    var kelvin: Double {
        return celsius + 273.15
    }
}

let currentTemperature = Temperature(celsius: 0.0)
print(currentTemperature.fahrenheit) //Output: 32.0
print(currentTemperature.kelvin) //Output: 273.15
```
struct StepCounter {
    var totalSteps: Int = 0 {
        willSet {
            print("About to set totalSteps to \(newValue)")
        }
        didSet {
            if totalSteps > oldValue {
                print("Added \(totalSteps - oldValue) steps")
            }
        }
    }
}

var stepCounter = StepCounter()
stepCounter.totalSteps = 40 //About to set totalSteps to 40       Added 40 steps
stepCounter.totalSteps = 100 //About to set totalSteps to 100      Added 60 steps
Classes

- Classes are reference types
- Basic class:

```swift
class Person {
    let name: String

    init(name: String) {
        self.name = name
    }

    func sayHello() {
        print("Hello, there!")
    }
}

let person = Person(name: "Jasmine")
print(person.name) // Jasmine
person.sayHello() // Hello, there!
```

- Subclass and override functions:

```swift
class Student: Person {
    var favoriteSubject: String

    override func sayHello() {
        print("Hello, student!")
    }

    init(name: String, favoriteSubject: String) {
        self.favoriteSubject = favoriteSubject
        super.init(name: name)
    }
}
```
Collections

• Groups of objects

• Two Common collection types:
  • Array:
    ```swift
    var names: [String] = ["Anne", "Gary", "Keith"]
    ```
  • Dictionary:
    ```swift
    var scores = ["Richard": 500, "Luke": 400, "Cheryl": 800]
    ```
  • Collections are **value types!!**
Arrays

• Array Types:

```swift
var myArray: [Int] = []
var myArray: Array<Int> = []
var myArray = [Int]()
var myArray = [Int](repeating: 0, count: 100)
let firstNumber = myArray[0] //Access values
myArray[1] = 2 //Set values
myArray.append(101) //Add Values
```
Dictionaries

- Dictionary Types:

```swift
var myDictionary = [String: Int]()
var myDictionary = Dictionary<String, Int>()
var myDictionary: [String: Int] = []

var scores = [
    "Richard": 500,
    "Luke": 400,
    "Cheryl": 800
]

scores["Oli"] = 399 // Set 399 for key "Oli"
let oldValue = scores.updateValue(100, forKey: "Richard") // Get old value before updating

let points = Array(scores.values) // [500, 400, 800, 399]

if let myScore = scores["Luke"] {
    print(myScore) // 400
}
```
For Loops

```swift
for index in 1...5 {
    print("This is number \(index)")
}

let names = ["Tim", "Cathy", "Jan"]
for name in names {
    print("Hello \(name)")
}

for _ in 1...3 {
    print("Hello!")
}

for letter in "ABCD".characters {
    print("The letter is \(letter)")
}

for (index, letter) in "ABCD".characters.enumerated() {
    print("\(index): \(letter)")
}

let vehicles = ["unicycle" : 1, "bike" : 2, "car" : 4, "truck" : 6]
for (vehicleName, wheelCount) in vehicles {
    print("A \(vehicleName) has \(wheelCount) wheels")
}
```
### While loop

**Basis while loop:**

```swift
var numberOfLives = 3
while numberOfLives > 0 {
    print("I still have \(numberOfLives) lives.")
}
```

- **Break loops:**

```swift
var numberOfLives = 3
var stillAlive = true
while stillAlive {
    numberOfLives -= 1
    if numberOfLives == 0 {
        break
    }
}
```
Introduction to UIKit

• The UIKit provides most important parts of an iOS app

• UIView is the foundation class for all visual elements in UIKit

• Subclasses of UIView:
  • UILabel
  • UIButton
  • UIImageView
  • UIScrollView
UIView subclasses

This is a label

UILabel

Hallo this is a text view

UITextView

UIImageView

UIScrollView

UITableView

UIToolbar

UITabBar

UINavigationBar
Gesture Recognizers

• Basic touch gestures

• Can be included using the Interface Builder

• Can also be added using code

• Custom touch input:

```swift
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) { }

override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) { }

override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) { }
```
Summary

• Strings

• Function

• Classes and Structs

• Collections

• Next week: Unit 3:
  • Auto layout
  • Swift Optionals, Guard, and Inspection
  • Segues and Navigation Controllers