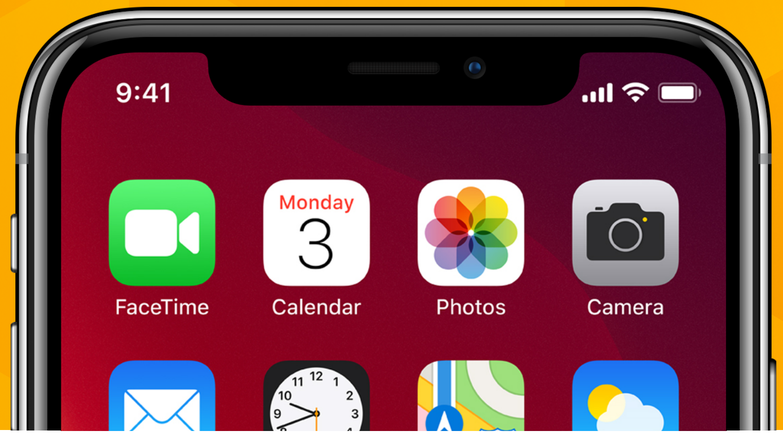


iOS Application Development

Lecture 3: More Swift, and Introduction to UIKit

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

WS ’22/’23 • hci.rwth-aachen.de/ios

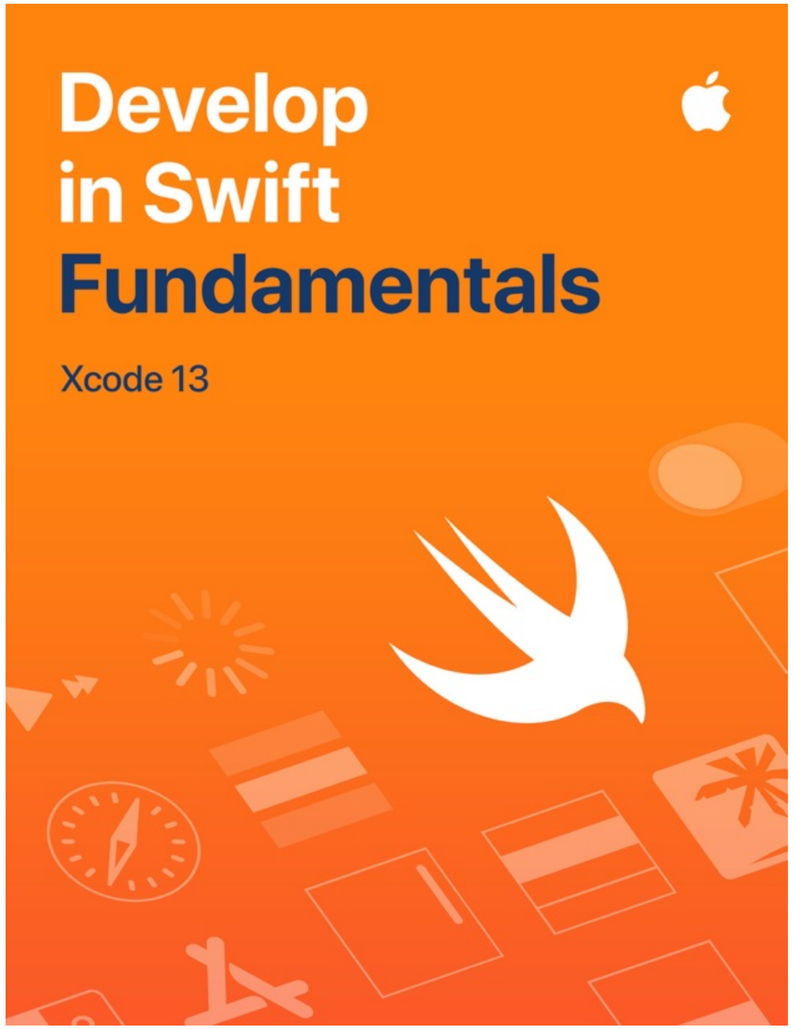
RWTH AACHEN UNIVERSITY

Recap

- Swift Basics
  - let vs. var
  - Type inference
  - Optionals
  - Tuples
  - …
- IDE Basics
  - Terminal, Swift Playgrounds, Xcode
  - Project settings, simulator, IBOutlet & IBAction

Textbook Reminder, and Other Resources

- Read along in the free eBooks from Apple Books
  - Also include Xcode, Interface Builder, and Documentation Browser tutorials
  - Try their interactive multiple-choice tests!
  - We’re currently about halfway(!) through the “Fundamentals” book
- These eBooks also point you to the corresponding sections in the official **Swift Programming Language Book** at swift.org (online & ePub)



Strings

- A Unicode collection of characters

```
var string = "hello"

let l = string.count // -> l = 5
```

- Individual characters are of type Character. But single-character strings are inferred as type String unless a type annotation says otherwise:

```
let letter1 = "a"                // letter1 is a String

let letter2: Character = "b"  // letter2 is a Character (no single quotes in Swift!)
```

Strings

- Check if String is empty:

```
var myString = ""

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

- Concatenation and Interpolation

```
let s1 = "Hello"
let s2 = ", world!"
let s3 = s1 + s2                // "Hello, world!"
var s4 = s3 + " Yay!"           // "Hello, world! Yay!"
s4 += " Greetings!"             // "Hello, world! Yay! Greetings!"

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:

```
let month = "August"
let otherMonth = "August"
let lowercaseMonth = "august"

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

- General function definition

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

- Simplest case: Functions without parameters and return type

```
func displayPi() {
    print("3.1415926535")
}

displayPi() // Output: 3.1415926535
```

Functions

- One parameter: Parameters must usually be **named** when calling!

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

- Multiple parameters: Note how naming parameters makes call more readable

```
func printPower(base: Double, exponent: Double) {
    let result = pow(base, exponent)
    print("The result is \(result).")
}
printPower(base: 2, exponent: 3) // Output: The result is 8.
```

Functions

- **Argument labels** make function calls more readable and meaningful

- “_” as argument label allows omitting the label when calling (but readability?)

```
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

```
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

```
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)”)



// Shorter version:
func multiply(firstNumber: Int, secondNumber: Int) -> Int {
    firstNumber * secondNumber // no return statement needed here
}
```

Structs

- A struct defines a new **type**
- Most basic Swift types are actually structs: String, Int, Double, Bool
- Structs are value types, while classes are reference types

```swift
struct Person {
    var name: String // name is a property of the Person type
}

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

Structs

- Like classes, structs can have **methods:**

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

let person = Person(name: “Jasmine") // Creating an instance of type Person
person.sayHello() // Output: Hello, there! My name is Jasmine!
```

Structs

- Creating **instances** of structs (and classes) requires **initializing** their properties:

```
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 odometer1 = Odometer()                // initializes count to default (here: 0)
let odometer2 = Odometer(count: 27000)    // initializes count to 27000
```

Structs

- Custom initializers:

```
struct Temperature {
    var celsius: Double

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

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

var currentTemperature = Temperature(celsius: 18.5)  // celsius: 18.5
let boiling = Temperature(fahrenheit: 212.0)         // celsius: 100.0
```

Structs

- Changing values:

```
struct Temperature {
    var celsius: Double

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

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

var currentTemperature = Temperature(celsius: 18.5)  // celsius: 18.5
let boiling = Temperature(fahrenheit: 212.0)         // celsius: 100.0

currentTemperature.celsius = 20.0
```

change property

Structs

- Changing values: struct methods cannot change properties by default!

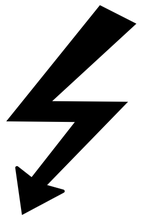
```
struct Temperature {
    var celsius: Double

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

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

    func changeCelsius(to newValue: Double){
        celsius = newValue
    }

}
```



Structs

- Such methods must be marked as “mutating”:

```
struct Temperature {
    var celsius: Double

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

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

    mutating func changeCelsius(to newValue: Double){
        celsius = newValue
    }
}
var currentTemperature = Temperature(celsius: 18.5)  // celsius: 18.5
currentTemperature.changeCelsius(to: 23.0)           // celsius: 23.0
```

Structs

- Computed properties:

```
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
```

Structs

- Property observers:

```
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() //Output:
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:

```
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:

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

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

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

In Swift, Use Structs Rather Than Classes

- Only use classes if you need
  - Inheritance (subclassing)
  - Multiple variables sharing a common reference to an object in memory
    - Classes are reference types, structs are value types
  - To subclass from classes used in frameworks (e.g., from UIView)

Collections

- Groups of objects
- Two common collection types:
  - Array:

```
var names: [String] = ["Anne", "Gary", "Keith"]
```

  - Dictionary:

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

- Collections are **value types!**

Arrays

- Array Types:

```
var myArray: [Int] = []

var myArray: Array<Int> = [] // Alternative initialization

var myArray = [Int]()        // Alternative initialization
```

```
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:

```
var myDictionary = [String: Int]()
var myDictionary = Dictionary<String, Int>()// Alternative initialization
var myDictionary: [String: Int] = [:]       // Alternative initialization
```

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

scores["Oliver"] = 399              // Set 399 for new key "Oliver"

let oldValue = scores.updateValue(100, forKey: "Richard")
                                    // Get old value then update

let players = Array(scores.keys)   // ["Richard", "Luke", “Cheryl", "Oliver"]
let points = Array(scores.values)  // [100, 400, 800, 399]

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

For Loops

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

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

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

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

```
for (index, letter) in "ABCD".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

- Basic while loop:

```
var numberOfLives = 3

while numberOfLives > 0 {
    print("I still have \(numberOfLives) lives.")
    numberOfLives -= 1
}
```

```
var numberOfLives = 3
var stillAlive = true
while stillAlive {
    numberOfLives -= 1
    if numberOfLives == 0 {
        stillAlive = false
    }
}
```

```
var numberOfLives = 3
var stillAlive = true
while stillAlive {
    numberOfLives -= 1
    if numberOfLives == 0 {
        break
    }
}
```

Introduction to UIKit

- 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

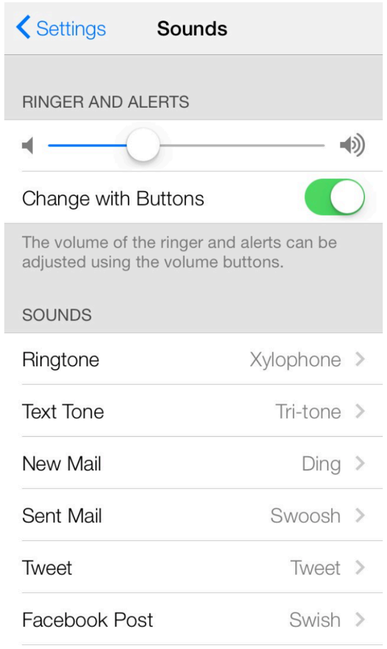
UILabel

Hello this is a text view

UITextView

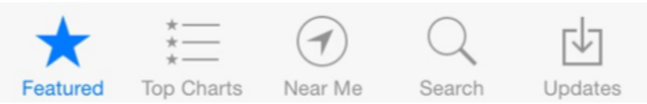
UIImageView

UIScrollView

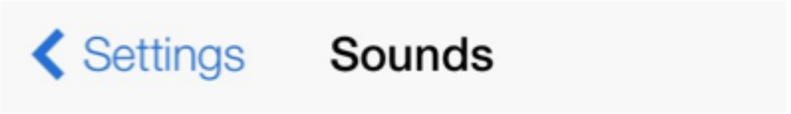


UITableView

UIToolbar



UITabBar



UINavigationBar

UIView Subclasses (Controls)

UIButton

UISlider

UISwitch

UITextField

UIDatePicker

UISegmentedControl

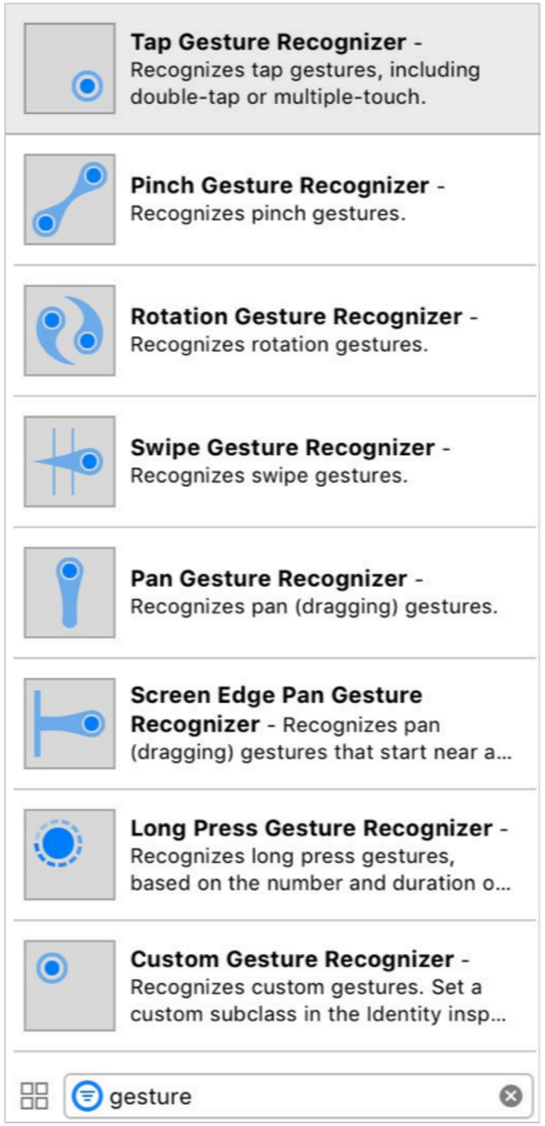
Gesture Recognizers

- Basic touch gestures
- Can be included using Interface Builder
- Can also be added using code
- Custom touch input:

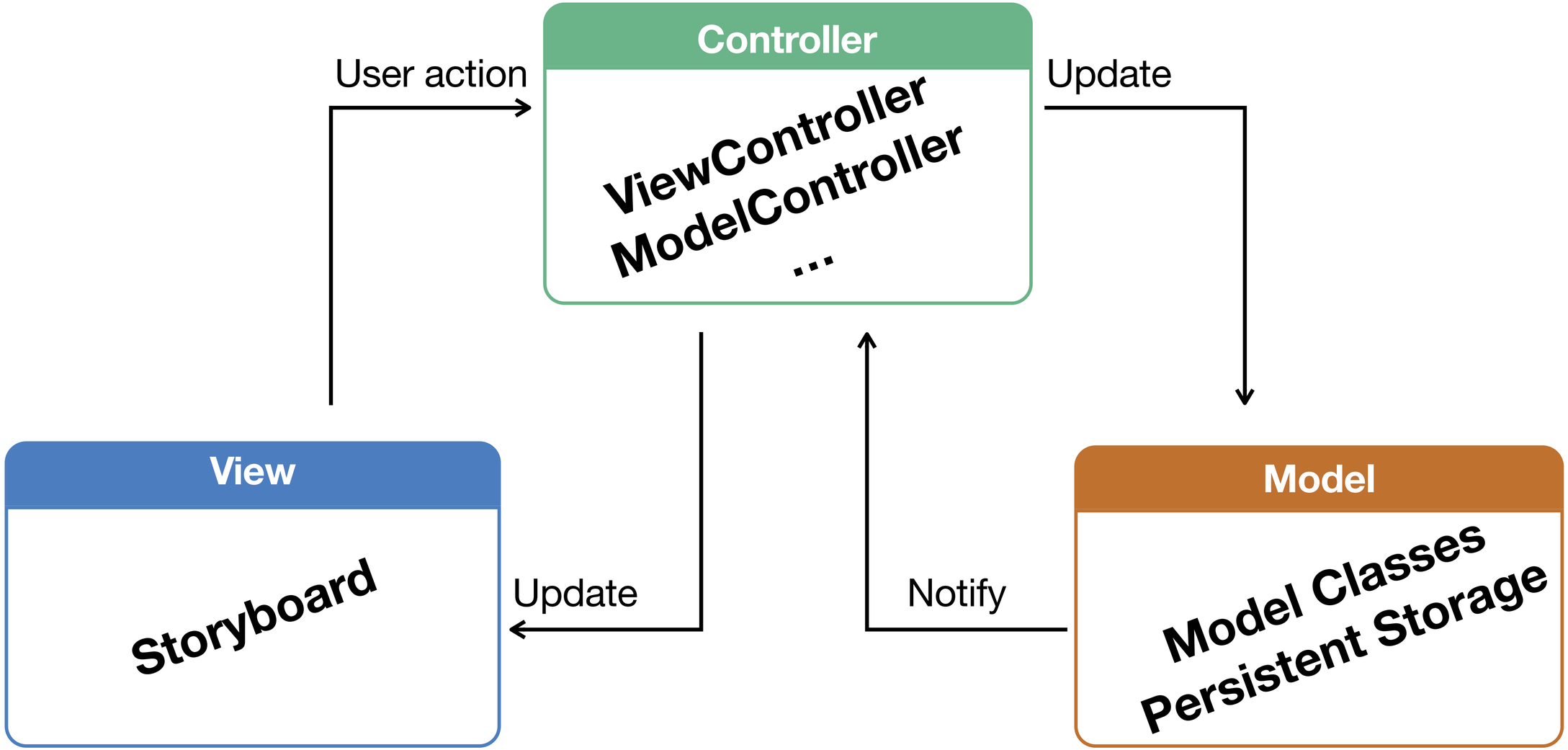
```
 override func touchesBegan(_ touches:
Set<UITouch>, with event: UIEvent?) { }

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

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



MVC in UIKit



Summary

- Strings
- Functions
- Classes and Structs
- Collections
- Next week:
  - Swift Optionals, `guard`, and Inspection
  - Segues and Navigation Controllers