INTRODUCTION
Programming Handheld Systems—iOS

TODAY

• More Swift!

• A bit about MVC

• Maybe a bit of a demo
SORTING WITH CLOSURES

// Fully declared closure
res = sorted(array, { (i1: Int, i2: Int) -> Bool
                      in return i1 > i2
                    } )

// Infer closure type
res = sorted(array, { (i1, i2) in return i1 > i2
                     } )

// Implicit returns
res = sorted(array, { (i1, i2) in i1 > i2 } )

SORTING WITH CLOSURES

// Shorthand argument names
reversed = sorted(array, { $0 > $1 } )

// Operator functions
reversed = sorted(array, >)

// Trailing closure: outside of () if final argument
reversed = sorted(array) { $0 > $1 }

// Optional parentheses
array.map({ $0 + 1 })
array.map { $0 + 1 } // equivalent
COMPUTED PROPERTIES

• Stored property
  ```swift
  var foo: String
  ```

• Computed property
  ```swift
  var foo: String {
    get {
      // return computer property
    }
    set(newValue) {
      // do something
    }
  }
  ```

Setter does not need to be implemented.

DATA STRUCTURING

• struct
• class
• enum
• protocols
**STRUCT**

- value type
  - when copied to another variable, or passed to a function
- copy-on-write changes

- requires write methods to specify `mutating`
- not on heap
- no inheritance (but *protocols...*)

```swift
struct Astruct {
    var one: Int
    var two: String

    func identity() -> String {
        return two + " \(one)"
    }

    mutating func changer() { // “mutating” allows method to change it’s instance
        one += 1
    }
}
```

```swift
let struct1 = Astruct()
let struct2 = Astruct(one: 3, two: "nice")
var struct3 = struct2 // changing struct3 doesn’t affect struct2
struct3.one = 42 // works
```

**CLASS**

Like *structs*, but:
- single inheritance
- allocate on heap
- *reference* type
- type-casting checks class type at runtime
- automatic reference counting

Much more to come...
MODEL/VIEW/CONTROLLER (MVC)

Divide objects into three categories.

Model: abstract application logic (not how shown)
**Model/View/Controller (MVC)**

**Controller**: how presented to user (UI design)

**View**: controller's minions: TableView, Button, Label

CMSC 436 (iOS) • Spring 2019 • Pete Keleher
Controllers can talk to directly to their model
Controllers can talk to directly to their view(s)
Can view talk to its controller?

Yes, through actions on targets (IBActions).
Controllers can also be notified through property observers.

Views do not own the data they display.
They must request it, almost always from the Controller.

Usually through a delegation protocol (e.g. UITableViewDelegate()).
**Model/View/Controller (MVC)**

Usually through a *delegation protocol* (e.g. `UITableViewDelegate()`).

**MULTIPLE VIEW CONTROLLERS**

Controllers control other controllers in a *hierarchy.*
MULTIPLE VIEW CONTROLLERS

TableViewController, ScrollViewController, NavigationController...

DEMO

- Set Game
  
  All this stuff can be very abstract until you see it in action.
  We'll start getting comfortable with Swift 5 and Xcode 11 by building something right away.
  Two part demo starting today, finishing Thursday.

- Today’s topics in the demo…
  
  - Creating a project in Xcode 11, including building a UI and running in the iOS simulator
  - Subclassing in Swift, including how to specify instance variables and methods
  - Connecting UI elements to invoke methods in our Swift code (actions)
  - Print (outputting to the console using the \( () \) notation)
  - Connecting properties (instance variables) from our Swift code to the UI (outlets)
  - Accessing iOS documentation from our code
  - Automatically doing something every time a property’s value changes
  - Arrays, optionals
SET GAME

- Rules:
  - Three cards either all same or all different on each axis

- Axis:
  - color
  - number
  - shade
  - shape