6.0 Lesson - Swift 2

Paul Solt - Paul@SuperEasyApps.com
SuperEasyApps.com

- 6.0 Lesson - Swift 2

6.1 Tutorial - Guard Keyword

```
struct EmailSubscriber {
    var firstName: String
    var email: String
}

// Supporting function for validation
func validateEmail(email: String) -> Bool {
    return true // false // switch to test logic
}

func createEmailSubscriber(firstName: String, email: String) -> EmailSubscriber? {

    // Guard to protect against invalid input
    guard firstName.characters.count > 0 else { // no empty strings (or blank names)
        print("Invalid first name")
        return nil
    }

    // check email format
    guard validateEmail(email) else {   // Paul@SuperEasyApps.com
        print("Invalid email")
        return nil
    }

    // All valid information
    return EmailSubscriber(firstName: firstName, email: email)
}

let invalidSubscriber = createEmailSubscriber("", email: "")
print("Invalid subscriber:", invalidSubscriber)

let validSubscriber = createEmailSubscriber("Paul", email: "Paul@SuperEasyApps.com")
print("Valid subscriber: ", validSubscriber)
```

6.2 Tutorial - Defer Keyword

```
func fileProcessing() {
    print("1. Create file descriptor #1 and start file processing")

    defer {
        print("5. Close and cleanup file descriptor #1 (I/O resource)")
    }

    print("2. Create file descriptor #2 and start file processing")

    defer {
        print("4. Close and cleanup file descriptor #2 (I/O resource)")
    }
    print("3. Finish file processing")
}

// Look at Xcode's Console output for order: 1, 2, 3, 4, 5.
fileProcessing()
```

6.3 Tutorial - Repeat While and Do Scope

```
// A method that counts down and outputs to Xcode's Console
func repeatWhile() {
    var x = 10
    repeat {
        print("T-minus", x, "seconds")
        x = x - 1 // x-- is being deprecated in Swift 3.0!
    } while x > 0
    print("Blast off!")
}

// Call the method in Playgrounds
repeatWhile()

// do scope

// Outer scope
let x = 7
do {
    // Inner scope (new x variable masks outer scope x variable)
    let x = 10
```

```
    do {
        // Inner inner scope ... inception
        let x = 200
        print("x:", x) // x: 200
    }

    print("x:", x) // x: 10
}
// outer scope
print("x:", x) // x: 7
```

6.4 Tutorial - Error Handling

```
// Create a custom error type by conforming to the ErrorType protocol
enum ValidationError: ErrorType {
    case InvalidName
    case InvalidEmail
    case InvalidAge(age: Int)
}

// Check name against your name policies
func validateName(name: String) throws {
    // Use guard statements to prevent invalid user input
    guard name.characters.count > 0 else {
        throw ValidationError.InvalidName
    }
}

// Process a new customer using required attributes
func onboardNewCustomer(name: String, email: String, age: Int) {
    do {
        print("Started onboarding")
        // You must use the try keyword for any method that can throw an error
        try validateName(name)

        // Exercise: Validate other required attributes (age, email, etc.)

        // Finished processing if no errors
        print("Finished onboarding")
    } catch let error as ValidationError {
        // Using a local variable you can catch all ValidationErrors

        // The local error variable can be handled with a switch
        switch(error) {
        case ValidationError.InvalidName:
            print("Invalid name!")
        case ValidationError.InvalidEmail:
            print("Invalid birthday!")
```

```
        case ValidationError.InvalidAge(let age):
            print("Invalid age \(age)!")
        }
    }
    catch {    // default error catch
        print("Catch errors here:", error) // error is default name
    }
}

onboardNewCustomer("", email: "", age: 12223) // Invalid!
onboardNewCustomer("Paul", email: "Paul@SuperEasyApps.com", age: 29)
```

6.5 Tutorial - Protocol Extensions

```
// Extend the Array class when Elements are comparable
extension Array where Element: Equatable {

    // Remove a single Element if it is found in the Array
    mutating func removeObject(object: Element) {
        if let index = self.indexOf(object) {
            self.removeAtIndex(index)
        }
    }

    // Remove multiple Elements using the previous method and a loop
    mutating func removeObjectsInArray(array: [Element]) {
        for object in array {
            self.removeObject(object)
        }
    }
}

// Use the protocol extension
var students = ["John", "Sue", "Michael", "Chris", "David", "Benjamin"]
var studentsToRemove = ["Michael", "David", "Benjamin"]

print("Students: ", students)
// Remove an array of student names
students.removeObjectsInArray(studentsToRemove)
print("Students: ", students)

students.removeObject("John")
print("Students: ", students)
```

6.6 Tutorial - OptionSetType

```
// You can create custom flags or options using OptionSetType
struct VideoFormat: OptionSetType {
    let rawValue: Int
    // Use static let values within the struct and assign
    //  the raw value using bitmask values (1, 2, 4, 8, 16, etc)
    static let Video1080p  = VideoFormat(rawValue: 1)
    static let Video720p = VideoFormat(rawValue: 2)
    static let EnableStreaming = VideoFormat(rawValue: 4)
    static let EnableDownloads = VideoFormat(rawValue: 8)
}

// Use Swift Set notation to pass options
func startVideoPlaybackWithOptions(videoStreamOptions: [VideoFormat]) {
    print("Play video stream:", videoStreamOptions)
}

// Store a collection of options using Set notation
let videoOptions = [VideoFormat.Video1080p, VideoFormat.EnableDownloads]
startVideoPlaybackWithOptions(videoOptions)

// Set notation makes legacy bitmask operations much easier to read
if videoOptions.contains(VideoFormat.Video1080p) {
    print("Video is 1080p!")
}
```

6.7 Tutorial - API Changes

```
class ViewController : UIViewController {

//     Swift 1.1 (APIs used Objective-C NSSet type)
//    override func touchesBegan(touches: NSSet, withEvent event: UIEvent) {
//        if let touch = touches.anyObject() as UITouch? {
//            let location = touch.locationInView(self.view)
//            println("Touch: \(location)")
//        }
//    }
//
//    Swift 1.2 (Swift Set introduced along with new as? keyword)
//    override func touchesBegan(touches: Set<NSObject>, withEvent event: UIEvent) {
//        if let touch = touches.first as? UITouch {
//            let location = touch.locationInView(self.view)
//            println("Touch: \(location)")
//        }
//    }
```

```
    // Swift 2.1 (Objective-C gains lightweight generics + Swift API update)
    override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?) {
        if let touch = touches.first {
            let location = touch.locationInView(self.view)
            print("Touch: \(location)") // New print() method
        }
    }
}
```