Reading Assignment II: Intro to Swift

Objective

We move on this week to cover important topics like `protocol`s, functions as types (including closures), computed properties, `init`ialization, access control, `enum` associated data and `extension`. This is probably the most important of the three reading assignments. Complete this reading before the start of Lecture 5.

Most of you have not had experience with Objective-C, but don't worry about that. Nothing in the Swift documentation really assumes that. However, if you have never programmed in C (or C++ or any other variant), then Swift might be extremely new to you (but hopefully still not too steep a hill to climb to learn).

Materials

- The reading in this assignment comes from two on-line documents: the Swift Programming Language and the Swift API Guidelines.

Swift Programming Language

Read the sections described below in the Swift Programming Language. To better utilize your valuable time and to emphasize important concepts, the sections in the reading have been annotated with three colors:

Red sections are VERY IMPORTANT and/or might be more difficult to understand. Read these carefully.

Yellow sections are important but probably won't be that difficult to understand.

Grayed-out sections are not required reading (this week). They may be in future weeks.

Don't gloss over reading any NOTE text (inside gray boxes)—many of those things are quite important. However, if a NOTE refers to Objective-C or bridging, you can ignore it.

If there is a link to another section in the text, you don't have to follow that link unless what it links to is also part of this week's reading assignment.

Note that a random sampling of the topics in the list below have links. There are not link destinations available for all topics, unfortunately, but for ones that exist, the link is included. This is just a way to help you jump to the "ballpark" of where a topic is. Linked topics are not any more or less important than any other topic.

In the **Language Guide** area, read the following sections in the following chapters:

The Basics

- Type Aliases
- Tuples
- Error Handling

Basic Operators

- Comparison Operators

Strings and Characters

- Unicode Representations of Strings

Collection Types

- Sets
- Performing Set Operations

Control Flow

- Conditional Statements
  - Tuples
  - Value Bindings
  - Where
- Control Transfer Statements
  - Continue
  - Labeled Statements
- Checking API Availability

Functions

- Function Parameters and Return Values
  - Functions With Multiple Return Values
- Function Argument Labels and Parameter Names
  - Variadic Parameters (optional, do not use these in this course)
  - In-Out Parameters (optional, do not use these in this course)
- Function Types

Closures

- Closure Expressions
- Trailing Closures
- Capturing Values
- Closures Are Reference Types
- Escaping Closures
- Autoclosures

Enumerations

- Enumeration Syntax
- Matching Enumeration Values with a Switch Statement
- Associated Values
- Raw Values
- Recursive Enumerations

Properties

- Computed Properties

Methods

- Instance Methods
  - Assigning to self Within a Mutating Method

Subscripts

- Subscript Syntax
- Subscript Usage
- Subscript Options

Inheritance

- Overriding
  - Overriding Properties

Initialization

- Class Inheritance and Initialization (complicated, but don't fret!)
- Failable Initializers
- Required Initializers
- Setting a Default Property Value with a Closure or Function

NOTE: You should not need a `UIViewController` initializer for assignment 1 or assignment 2 (and hopefully not for any assignment all quarter long!). So unless you disagree with that, you can skip to the next page!

But … in the interests of full disclosure … here is the bare minimum you need to know if you for some reason feel you absolutely must have an initializer in your `UIViewController` subclass (again, hopefully never).

`UIViewController` has two initializers and both (or **neither**) should be implemented in a subclass …

```
override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil: NSBundle?) {
    super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)
    <your initialization code here>
}
```

and

```
required init?(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)
    <your initialization code here>
}
```

Don't forget the `override` and `required` keywords. Obviously you'd likely want to factor your initialization code into some other method you can call from both of these.

But if you can avoid implementing these (which you almost always can), please do. It's an annoying historical artifact (IMHO). Most `UIViewController` initialization occurs in the following View Controller Lifecycle method (which we will talk about in lecture):

```
override func viewDidLoad() {
    super.viewDidLoad()
    <your initialization code here>
}
```

When this is called, all of your outlets have been connected, but your Controller's View is not on-screen yet, so this is a great place to do all your one-time initialization. I would recommend you always design your `UIViewController` subclass so that initialization can occur here instead of futzing with the (somewhat arcane) initializers of `UIViewController`. Don't forget the strategy of making a property be an *implicitly unwrapped optional* if you have to (and initialize it in `viewDidLoad`). This is how `UIViewController` handles outlets (although it initializes those *just before* `viewDidLoad` is called, not in `viewDidLoad` itself).

Deinitialization

Optional Chaining

This chapter is (no pun intended) optional reading at this point. It is a very cool way to make your code very concise, but if you are struggling with understanding Optionals at this point, you can wait to read this until next week.

Error Handling

Type Casting

Nested Types

Nested Types in Action
Referring to Nested Types

Extensions

Extension Syntax
Computed Properties
Initializers
Methods
Subscripts
Nested Types

Protocols

- Protocol Syntax
- Property Requirements
- Method Requirements
- Mutating Method Requirements
- Initializer Requirements
- Protocols as Types
- Delegation
- Adding Protocol Conformance with an Extension
- Collections of Protocol Types
- Protocol Inheritance
- Class-Only Protocols
- Protocol Composition
- Checking for Protocol Conformance
- Optional Protocol Requirements
- Protocol Extensions

Automatic Reference Counting

Memory Safety

Access Control

Only Access Levels section is required for now.

Advanced Operators

Swift API Guidelines

Read the Swift API Guidelines document in its entirety.

Given that you are completely new to Swift, some of what is in this document will be a bit hard to fully absorb at first. But familiarizing yourself with what is in this document is crucial to writing good Swift code. So, for this assignment, the goal is to know what's there rather than completely and fully master the guidelines right off the bat. As the quarter progresses, you should eventually become an expert namer of properties, methods and other Swift constructs. This will require you to refer back to this document often.

Be sure to click everywhere that it says "MORE DETAIL".

Pay special attention to the "Write a documentation comment" section.

Pay special attention to the "Follow case conventions" section.

Pay special attention to the entire "Argument Labels" section.

You can also ignore the final section (Special Instructions) for now.