Reading Assignment II: Functional Programming

Objective

There's not as big a volume of reading this week, but many of you might find the topics this week a bit more dense and foreign to what you know (depending on your experience, of course).

The primary goal is to understand the fundamental constructs in Swift that support functional programming: `protocol`s, `extension`s, generics and closures. An additional important topic is Optionals. We've mentioned all of these things in lecture, but now it's time to study them.

Due

You should have all of this reading done before lecture 6.

Materials

You'll continue reading from the same document(s) (e.g. Swift Programming Language) as last week.

Swift Programming Language

Don't gloss over reading any `NOTE` text (inside gray boxes) since 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.

Always read the overview at the top of each major section (e.g., in The Basics, be sure to read the part that starts "Swift provides many fundamental data types …").

You've read everything in A Swift Tour except the last two sections. We'll postpone Error Handling to next week (just to reduce volume this week).

Concurrency
Protocols and Extensions
Error Handling

In the Language Guide area, read the following sections in the following chapters. It is important to understand the information in these sections for you to be able to continue to follow along in lecture, so don't blow off this reading.

The Basics

Optionals are extremely important in Swift. We're still not going to worry about the Error Handling API in Swift for now.

Optionals
Error Handling
Assertions and Preconditions

Basic Operators

This is related to Optionals and thus important.

Nil-Coalescing Operator

Strings and Characters

If you skipped this section last week, here's another chance to read this.

String Literals
Initializing an Empty String
String Mutability
Strings Are Value Types
Working with Characters
Concatenating Strings and Characters
String Interpolation
Unicode
Counting Characters
Accessing and Modifying a String
Substrings
Comparing Strings
Unicode Representations of Strings

Collection Types

`Set` is not quite as commonly used as `Array` and `Dictionary`, but it has a lot of cool uses (it's sort of under-appreciated).

Sets
Performing Set Operations (and this)

Control Flow

Just finishing off a couple of interesting language features here.

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

Functions

In-Out Parameters are rarely used in Swift. Variadic parameters can be a clean way to pass in essentially an "array" of values as an argument, especially when specifying an array of one item would be common.

Function Argument Labels and Parameter Names
    Variadic Parameters
    In-Out Parameters

Closures

Another of the "most important topics of the week." Here you'll read the remaining parts leftover from last week. Definitely make sure you understand this entire topic (i.e. both last week's and this week's reading in this section).

Capturing Values
Closures are Reference Types
Escaping Closures
Autoclosures

Enumerations

Raw value enums are not really that important, but we might as well toss them in this week.

Raw Values
Recursive Enumerations

Structures and Classes

No new reading in this section, but be certain you understand the difference between a reference type and a value type.

Properties

You're finding out in the demos that computed properties are used a lot in SwiftUI.

Computed Properties
Property Observers
Property Wrappers

Methods

Only one section left here, but what it talks about is something we're more likely to do in an `enumeration`. Go ahead and read that section, but understand that you won't be doing it that often (I'll try to demo it later in the quarter).

Instance Methods
    Assigning to self Within a Mutating Method

Subscripts

Inheritance

We're kind of going to gloss over object-oriented programming in Swift because (outside of an `@Observable`), you're not going to use it much in SwiftUI. So I've left it gray again to reduce volume this week.

Initialization

The section in yellow is a really nifty feature of Swift and one that can get you around the Catch-22 of the inability to call functions on yourself while initializing a `var`. The gray sections are us skipping over object-oriented programming to reduce bulk in this reading assignment.

Setting Initial Values for Stored Properties
Customizing Initialization
Default Initializers
Initializer Delegation for Value Types
Class Inheritance and Initialization
Failable Initializers
Required Initializers
Setting a Default Property Value with a Closure or Function

Deinitialization

Optional Chaining

Optional Chaining is part of understanding Optionals (one of the "most important topics this week"). Read the entire section.

Error Handling

Concurrency

Macros

Type Casting

Nested Types

Nested Types in Action
Referring to Nested Types

Extensions

The first of three sections which cover the fundamentals of functional programming in Swift. Your goal this week is really to try to let this concept of functional programming sink in as you read these sections.

Extension Syntax
Computed Properties
Initializers
Methods
Subscripts
Nested Types

Protocols

Ditto Extensions explanation above.

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

Generics

Ditto above again.

We did not talk about generics as they apply to `protocols` themselves (i.e. the sections about "associated types" below). I'm leaving that out this week (although doing so might make some of the other sections refer to things you won't have read) just because there's so much protocol-oriented programming to absorb here, adding that is probably over the top.

The Problem that Generics Solve
Generic Functions
Type Parameters
Naming Type Parameters
Generic Types
Extending a Generic Type
Type Constraints
Associated Types
Generic Where Clauses
Extensions With a Generic Where Clause
Contextual Where Clauses
Associated Types with a Generic Where Clause
Generic Subscripts

Opaque and Boxed Protocol Types (this explains "some View")

Automatic Reference Counting

Memory Safety

Access Control

Modules and Source Files
Access Levels
Access Control Syntax
Custom Types
Subclassing
Constants, Variables, Properties, and Subscripts
Initializers
Protocols
Extensions
Generics
Type Aliases

Advanced Operators

Swift API Guidelines

Read this Swift API Guidelines document in its entirety.

Read this over **again** this week. It should be starting to sink in. 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 ignore (for now), points that reference Protocols. When we learn about Protocols next week, be sure to check back with this document after that.

You can also ignore the final subsection of the final section "Special Instructions `->` Take extra care with unconstrained polymorphism". We won't be doing anything with the `Any` and `AnyObject` types.