A Functional Programming Kickstart

Introducing Functional Programming Fundamentals In Swift

by Daniel H Steinberg

Editors Cut
Copyright

"A Functional Programming Kickstart", by Daniel H Steinberg

Copyright © 2020 Dim Sum Thinking, Inc. All rights reserved.


Book Version

This is version 0.9 for Swift 5.3, Xcode 12, macOS Big Sur, and iOS 14 released November 2020.

Legal

Every precaution was taken in the preparation of this book. The publisher and author assume no responsibility for errors and omissions, or for damages resulting from the use of the information contained herein and in the accompanying code downloads.

The sample code is intended to be used to illustrate points made in the text. It is not intended to be used in production code.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks or service marks. Where those designations appear in this book, and Dim Sum Thinking, Inc. was aware of the trademark claim, the designations have been printed with initial capital letters or in all capitals.

This book uses terms that are registered trademarks of Apple Inc. for which the terms of use don’t permit rendering them in all caps or initial caps. You can view a complete list of

The Editor’s Cut name and logo are registered trademarks of Dim Sum Thinking, Inc.
Chapter 1: Magic

The Trick
There is something magical about Functional Programming.

Only a secret cabal of wizards understand it and they use words like Monads and Applicative Functors to scare you away.

That's non-sense.

You already know a lot about Functional Programming but you haven't organized your thoughts yet in a way that it is useful to you on a day to day basis.
We'll start with some fundamentals that are probably familiar to you as an experienced Swift developer but we'll emphasize them and look at them in some new ways.

As I will say several times during this book, I'm not asking you to throw out any of the programming practices that currently serve you well.

In this book I just offer you insight into some techniques you may not yet use and show you how you might integrate them into the work you do.

I was recently leading a workshop and one of my students said, "I just think that objects are a more natural way of looking at the world."

I smiled.

The student looked at me quizzically and asked, "what?"

I explained that I'm very old. I'm old enough to remember when we first started to teach Object-Oriented programming and students told us that it wasn't a very natural way to look at the world. They argued that it was contrived.

All of this is contrived.

You need to pick the metaphors that best work for you.

At the end of the book you will have new tools that you can mix and match with your existing tools.
Please don't fight against the new tools before you've tried them out in real world projects.

At the same time, please don't get so excited about a new set of toys that you discard the perfectly-good existing ones.

Functional programming isn't new.

As old as I am, Functional Programming is older. Its roots in mathematics go back one hundred years. Its roots in programming in languages such as LISP go back more than sixty years.

In many ways it is magical.

It will change you.

Let's begin with a magic trick.
CHAPTER 1 : SECTION 1

The Trick

In order to play along, look in the code download. You'll see one folder for each chapter. This chapter's folder is 01. Inside 01 you'll find two playgrounds. The starting point for this chapter is the one named Magic.playground. The completed version of the Magic.playground is in the same directory. It's named MagicFinal.playground.

Open Magic.playground now in Xcode. You can toggle the navigation using Command - 0. The first playground page is 01TheTrick.

Start there.

A couple more playground notes.

You should see hyperlinks to Previous and Next on the top and bottom of the playground page. If instead you see something like this:

01/TheCardTrick.playground/01TheTrick

//: [Previous](@previous)

//: [Next](@next)
go to the Editor menu and towards the bottom you'll find Show Rendered Markup. Choose that and you should now have the hyperlinked text.

Also, there is some code tucked away in the playground page's Source directory and in the Source directory for the entire playground. Don't look at either yet.

We'll perform a magic trick together.

**The Deck**

I have a deck of cards which I have shuffled thoroughly.

Add the following line to the playground page to see the Deck.

```
01/TheCardTrick.playground/01TheTrick

let theDeck = MagicDeck()
```

Run the playground.

On the right side of the screen you will see this representation of theDeck.

```
[A ♠, 2 ♠, 3 ♠, 4 ♠, 5 ♠, 6 ♠, (…)]
```

I've abbreviated the output with (…). You'll see all of the cards listed, A through K for each of the four suits: ♠, ♦, ♣, and ♥.
I'll sometimes represent this playground feedback as part of the code listing at the right side of the page like this.

```
01/TheCardTrick.playground/01TheTrick

let theDeck = MagicDeck()
    [A ♠, 2 ♠, 3 ♠, 4 ♠, 5 ♠, 6 ♠, (...)]

Let's perform a trick with theDeck.

Pick a card - any card

I spread the cards in my deck face down and invite you to pick any card you want.

You are free to choose any one of the fifty-two cards in front of you. You can pick the top card (which is curiously number 0), the second card (number 1) and so on. Any card you wish.

Once you've chosen any card from 0 to 51, we'll remove that card from the deck. That's yourCard.

You look at your card while I organize the other cards face down into a nice neat pile.

Don't show me your card. I'd like you to memorize it.
let theDeck = MagicDeck()

let yourCard = theDeck.selectCard(at: 22)

In this case yourCard is the 10 ♦.

Oops. I guess I've seen your card. No matter, let's continue.

Return the card to the deck

Place yourCard face down on the top of theDeck.

Notice in this code listing, the existing code is blue and the code you need to add is red. This should make it easier for you to code along with me as we work through the book. I will use the red color to indicate code we are adding or highlighted code I want to discuss with you.

let theDeck = MagicDeck()
let yourCard = theDeck.selectCard(at: 22)

theDeck
    .top(with: yourCard)

Comments will be set off in a light grey.

In the following listing you see existing code, a comment, code to be added, and the playground feedback that shows yourCard on top
of `theDeck`. Also, at the top of the listing is the location of the code including the path to the file and the particular page or file being modified.

`01/TheCardTrick.playground/01TheTrick`

```swift
let theDeck = MagicDeck()
let yourCard = theDeck.selectCard(at: 22)
// add the code below to place yourCard on top of theDeck

theDeck
  .top(with: yourCard)

[10 ♦, A ♠, 2 ♠, 3 ♠, 4 ♠, (...)]
```

Here's is a comic rendering of a summary of the styling I'll use in this book to represent the code in listings.
Let's continue with our trick.

The Misdirection
Without looking at yourCard, I take the top card off the deck and tear it into little pieces.

You glance over and see from the scraps that the torn up card is the card you selected.

Incidentally, if I don't show you previous code in a listing, you should add the new code below the existing code.

```
// The Card Trick playground/01TheTrick

let topCard = theDeck.topCard

let rippedCard = rip(topCard)

I put the pieces into a metal bowl, safely pour a fluid over them, and light the fluid on fire.

When the flames burn out we look in the bowl and there's nothing but some burnt, unrecognizable residue.

I cover the bowl with a silk handkerchief and set the bowl and the handkerchief to one side.

// The Card Trick playground/01TheTrick

let burnedCard = burn(rippedCard)

let coveredCard = cover(burnedCard)
```
It is actually the burned, ripped 10 ♦ but once it's been burned, it hardly matters that it was ripped as well.

**The reveal**

I tap `theDeck` three times and take the `topCard` off `theDeck` and ask you, "is this yourCard?"

To your amazement, it is.

01/TheCardTrick.playground/01TheTrick

```swift
let this = theDeck.tapThreeTimes.topCard

this == yourCard
```

"How did you do that?" you ask.

I could tell you, but you'll be awfully disappointed.

There's not much to the magic you'll see in this chapter. We're reviewing some of what you know about mutating and non-mutating functions in Swift. We're looking at structs and classes, `lets` and `vars`.

Mostly, we're warming up.

Let's begin with mutable objects and the dangers of letting others change our world out from under us.