Building Apps with Swift

An Introduction

HAPPYFUNCORP TECHNOLOGY ACADEMY

HappyFunCorp

- We are a product company with over 50 developers and designers around the globe.

- We believe in Building tech solutions through **product design-driven engineering**.

- Over the last 5 years, we've launched over 130 different sites and mobile applications.

Who am I

- Will Schenk, CoFounder.
- Over 15 years of large scale internet software development experience.
- Fills the CTO role for many startups.
- Production code in C, C++, Java, Ruby, Erlang, Python, Perl, Objective-C, etc.
- Lead teams that used many more technologies.

Technology Academy

Other Events

- HFC Academy provides focused events and two-week deep dives into specific subjects concerning the tech startup community.
- Two tracks, starting January.
- hfcacademy.com

Product Track

- The anatomy of the product development process
- Deciding on an MVP
- Dealing with imperfection: Releasing early and often
- Pivoting and what that means to your brand

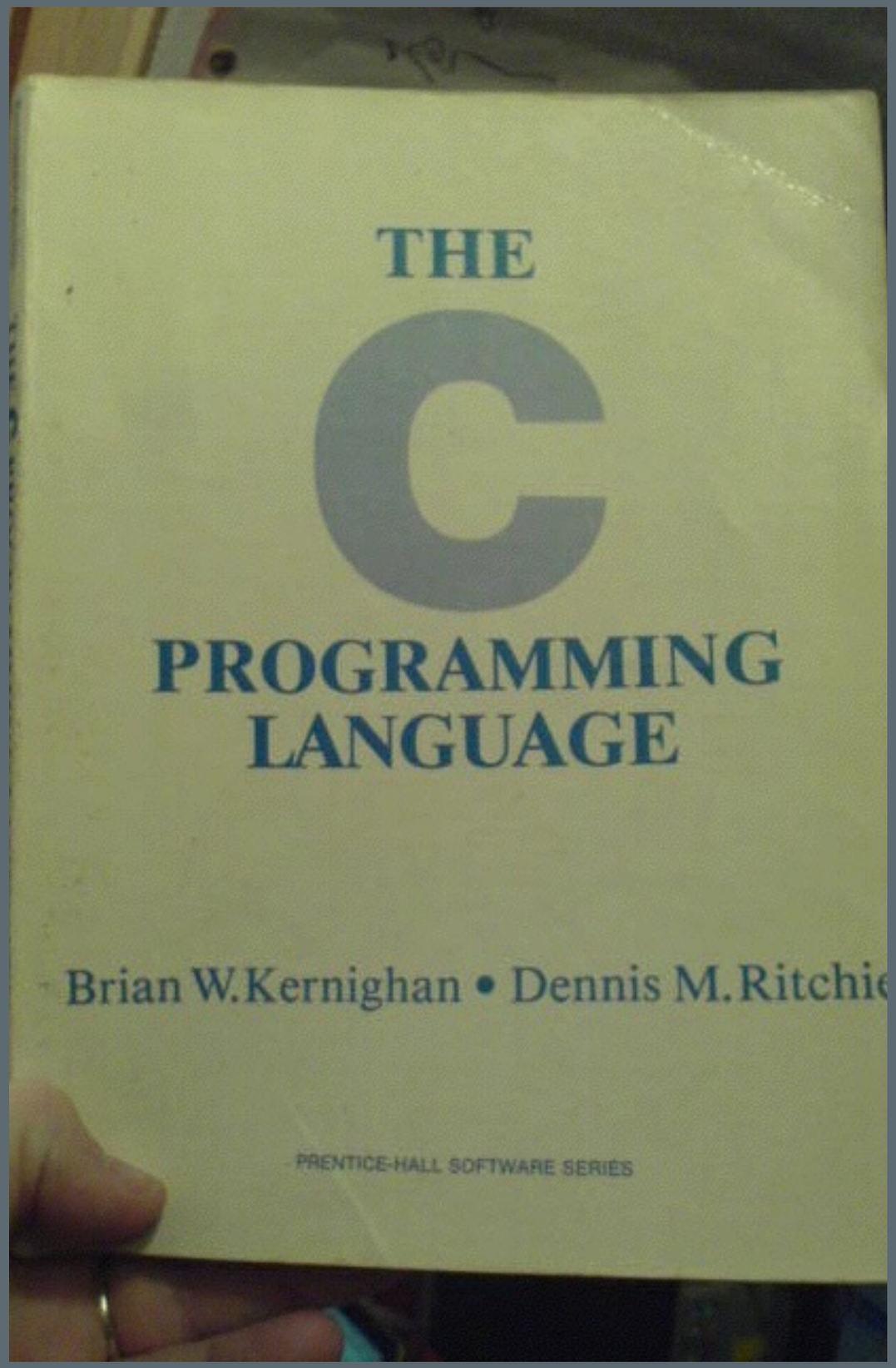
Technology Track

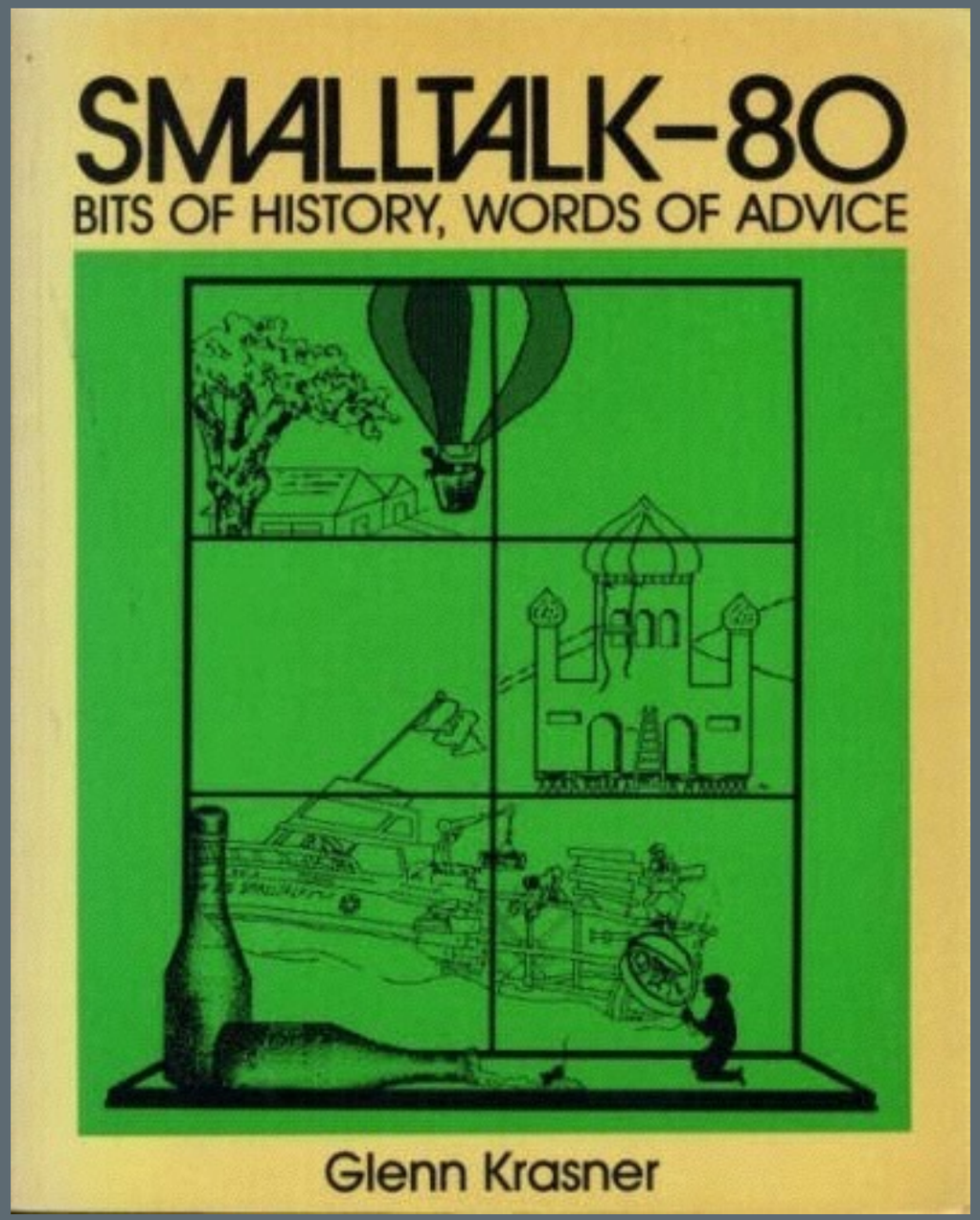
- The world of Parse
- Basic startup stacks
- Advanced Bootstrap
- ***Prototyping in Xcode : You are getting a preview***

History

Hello, World

Brian Kernighan actually wrote the first "Hello, World!" program as part of the documentation for the BCPL programming language developed by Martin Richards. BCPL was used while C was being developed at Bell Labs a few years before the publication of Kernighan and Ritchie's C book in 1978.





Object Oriented

Message Passing

Dynamically Typed

“human-computer symbiosis”

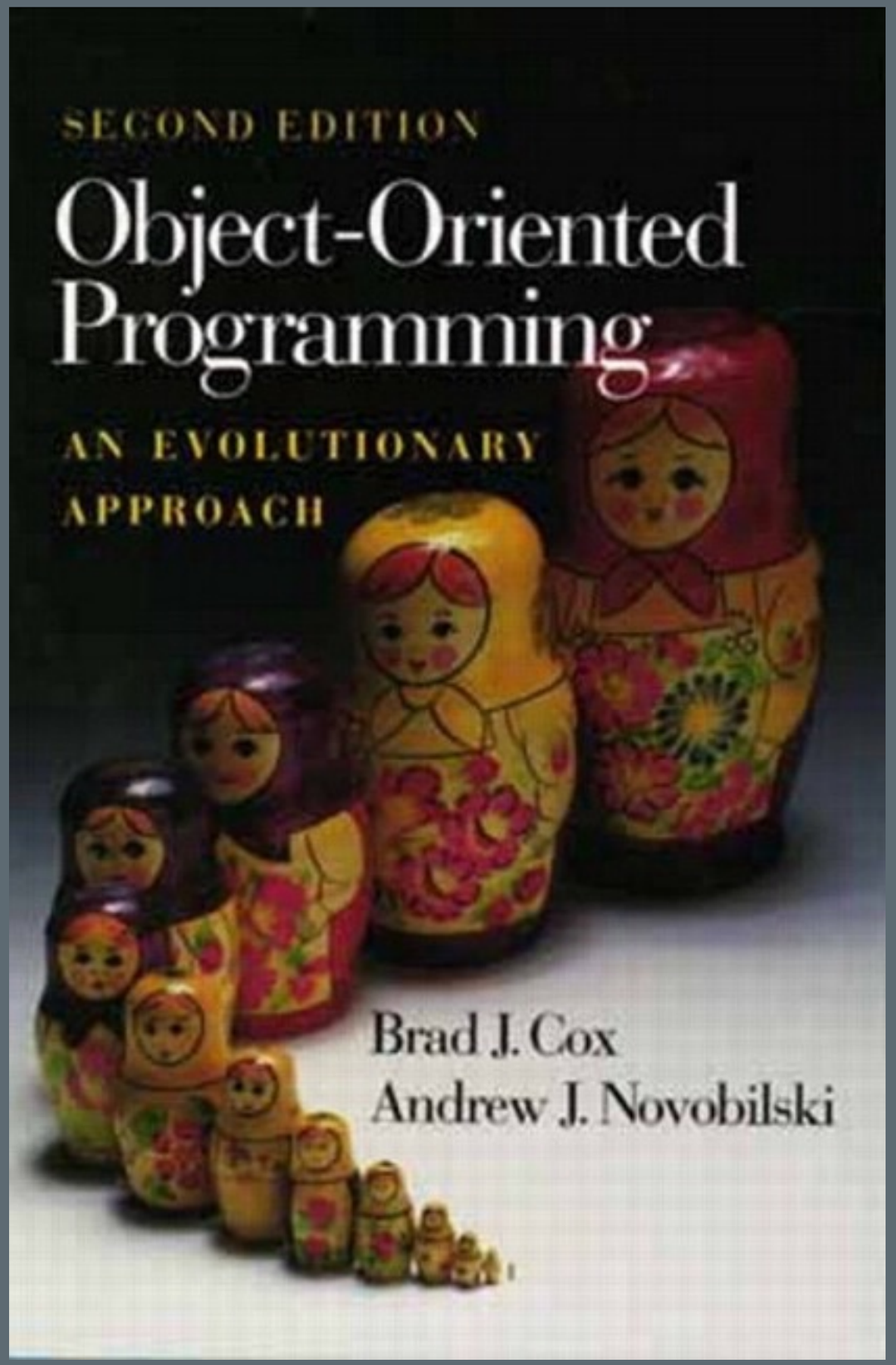
The best way to predict the future is to invent it.
— Alan Kay

The Love Child

1986

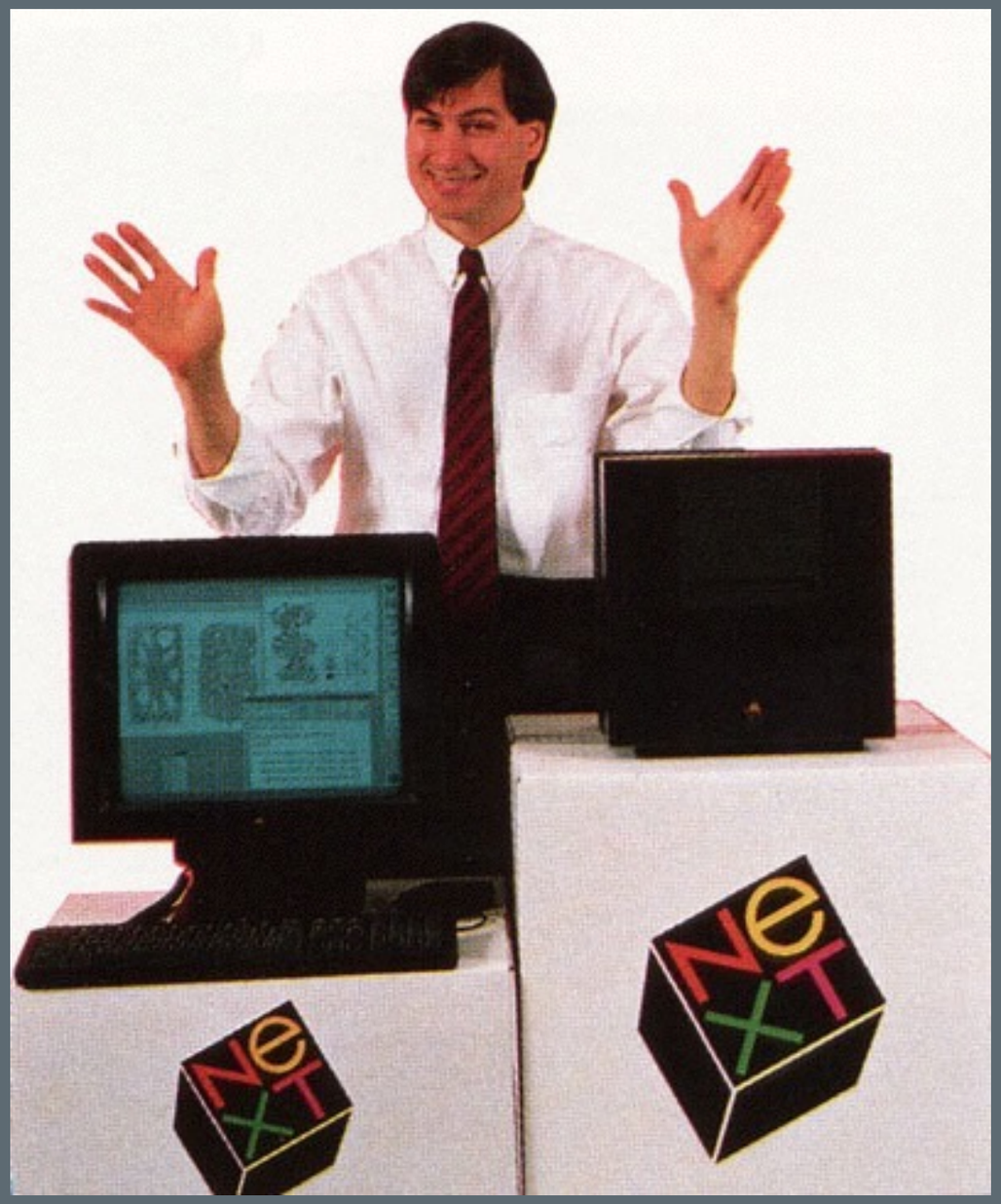
Strict Superset of C

Added Objects, Message Passing, Prototypes



Objective-C is Weird

- An evolutionary dead-end from the 80s.
- It’s a strict C superset.
  - C preprocessing continues to be gross.
  - C memory model and runtime.
  - Weird syntax.
- Everyone else uses C++, but that wasn’t around yet.



NeXTStep

On his sabbatical, he did some stuff with 3D graphics that he sold to Disney, and built another computer company that he sold to Apple.
In 1989, that company released their UI built with Objective-C.

iOS Apps are NeXTStep apps

- Apple bought NeXT.
- OS X is based on NeXTStep.
- iOS has the same core as OS X.
- Cocoa (App) vs Cocoa Touch (UIKit)

30 years

- 1978: C first released
- 1986: Objective-C
- 1989: NeXTStep
- 2001: OSX
- 2008: iOS SDK

Meanwhile…

- REPL — Optimized coding process
- IDEs — Navigating code bases
- Debuggers — Forensics
- Optimizing Compilers, JIT — Better runtimes.
- Real type systems — Software Engineering
- Software Packaging — Distribution

Enter Swift

Objective-C without the C

Way more modern

- Designed by Chris Lattner, who architected LLVM and leads Apple's Developer Tools department.

- Draws ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and many others.

- Safe pointers, better type system, dot-style syntax, no header files, better parameter naming, less junk.

Objective-C

```
NSString *str = @"hello,";
str = [str stringByAppendingString:@" world"];
```

Swift

```
var str = "hello,"
str += " world"
```

Read Eval Print Loop

- LISP, Ruby, Python, etc.
- Very useful to explore ideas.
- Type in code, see the values as you go.
- Great to explore syntax.

Swift Playgrounds

- Like REPLs and Mathematica's and IPython's Notebooks, lets you type in code and the result appears.

- If your code runs over time, for instance through a loop, you can watch its progress in the timeline assistant. The timeline displays variables in a graph, draws each step when composing a view.

- It's really hard to do this in a graphical environment.

Lets Look at the Playground

```
import Foundation

var str = "Hello, World"

str

println(str)

var greeting = ["Hello", "World" ]

", ".join( greeting )

greeting[1] = "Playground"

", ".join( greeting )
```

"Hello, World"

"Hello, World"

"Hello, World"

["Hello", "World"]

"Hello, World"

"Playground"

"Hello, Playground"

Variables are defined with var
Constants are defined with let

- Type is inferred
- var a = “String”
- var a = [“String”]
- If it can’t tell, you need to specify
- var emptyStringArray: [String] = [];

Playing with arrays

```
// We need to specify the type because it
// can't infer it from an empty array
var name_array:[String] = []

name_array += ["Happy"]
name_array += ["Fun"]
name_array += ["Corp"]

let hfc = join("", name_array);
```

```
0 elements

["Happy"]
["Happy", "Fun"]
["Happy", "Fun", "Corp"]

"HappyFunCorp"
```

Generic Arrays

```
// What is the size of the string
countElements( hfc )
// What is the size of the array
countElements( name_array )
contains(name_array, "Fun")
dropFirst(name_array)
equal(name_array, ["Happy","Fun","Corp"])
find(name_array, "Corp")
startsWith(name_array, ["Happy", "Fun"] )
join(":", name_array)
name_array.reverse();
```

```
12

3
true
["Fun", "Corp"]
true
{Some 2}
true
"Happy:Fun:Corp"
["Corp", "Fun", "Happy"]
```

Numerics

```
var number_array = [7,3,5,2,9,4]
```
[7, 3, 5, 2, 9, 4]

```
maxElement(number_array)
minElement(number_array)
```
9
2

```
let sorted = number_array.sorted{$0 < $1}
sorted
```
(12 times)
[2, 3, 4, 5, 7, 9]

```
let odd = number_array.filter{$0 % 2 == 1}
odd
```
(7 times)
[7, 3, 5, 9]

```
let lt5 = number_array.filter{$0 < 5}
lt5
```
(7 times)
[3, 2, 4]

```
let odd_func = {$0 % 2 == 1}
filter(1...10, odd_func)
```
(11 times)
[1, 3, 5, 7, 9]

```
let squares = number_array.map{$0 * $0}
squares
```
(7 times)
[49, 9, 25, 4, 81, 16]

```
let sum = number_array.reduce(0){$0 + $1}
sum
```
(7 times)
30

First Class Functions

- number_array.sorted{$0 < $1}
- { } is a function that is passed to the sorted method.
- $0 and $1 get bound when sorted invokes the function.

This is the longer version

```
let sortFunc = { (i1: Int, i2: Int) ->
    Bool in
    return i1 < i2 }

number_array.sorted(sortFunc);
```

```
(Function)

(11 times)

[2, 3, 4, 5, 7, 9]
```

Looping

```
// Loops
var numbers = (0...10)

var total = 0
for i in numbers {
    total += i
}
total

while total < 0 {
    total -= 5
}
total
```

```
"0..<11"

0

(11 times)

55

55
```

Classes

```
// Lets make a class

class Rectangle {
    var width:Double = 0, height:Double = 0;
    init( width:Double, height:Double ) {
        self.width = width
        self.height = height
    }

    func area() -> Double {
        return self.width * self.height;
    }
}

var shape1 = Rectangle( width: 5, height: 10 )

shape1.area()
```

(2 times)

{width 5.0 height 10.0}

50.0

Subclasses

```
class Square: Rectangle {
    init( size: Double ) {
        super.init(width: size, height: size)
    }
}

var shape2 = Square( size: 10 )
shape2.area()
```

```
{{width 10.0 height 10.0}}
100.0
```

Wrapping a Object-C class

```
class Regex {
    let internalExpression: NSRegularExpression?
    let pattern: String

    init(_ pattern: String) {
        self.pattern = pattern
        var error: NSError?
        self.internalExpression = NSRegularExpression(pattern:
            pattern, options: .CaseInsensitive, error: &error)
    }

    func test(input: String) -> Bool {
        let matches = self.internalExpression!.matchesInString(input,
            options: nil, range:NSMakeRange(0, countElements(input)))
        return matches.count > 0
    }
}
```

Using it

```
if Regex("Fun").test(str) {
    println("matches pattern")
} else {
    println( "Doesn't match" );
}

if Regex("Happy").test(hfc) {
    println( "Is happy!" )
}
```

```
"Doesn't match"

"Is happy!"
```

Optional Values

- Variables either have values, or are unset.
- No more null variables.
- You should check to see if they are set before using them.
- Use ! after the name to tell the compiler you've checked.

Custom Operators

```
// And a custom operator

infix operator =~ {}

func =~ (input: String, pattern: String) -> Bool {
    return Regex(pattern).test(input);
}

// This is more like Ruby!
if hfc =~ "Fun" {
    println( "Is also fun!" )
}
```

```
true

"Is also fun!"
```

This is generally too much power

XCPlayground

- XCPlayground lets you interact with the playground environment.
- You can add views!
- Good to explore data, as well as play with layouts.

Plotting a Sin Wave

```
import UIKit
import XCPlayground

let values = (0...100)

let sineArraySize = 64
let frequency1 = 4.0
let phase1 = 0.0
let amplitude1 = 2.0
let sineWave = (0..<sineArraySize).map {
    amplitude1 * sin(2.0 * M_PI / Double
        (sineArraySize) * Double($0) * frequency1
        + phase1)
}

for value in sineWave {
    XCPCaptureValue( "SineWave \(frequency1)",
        value )
}
```

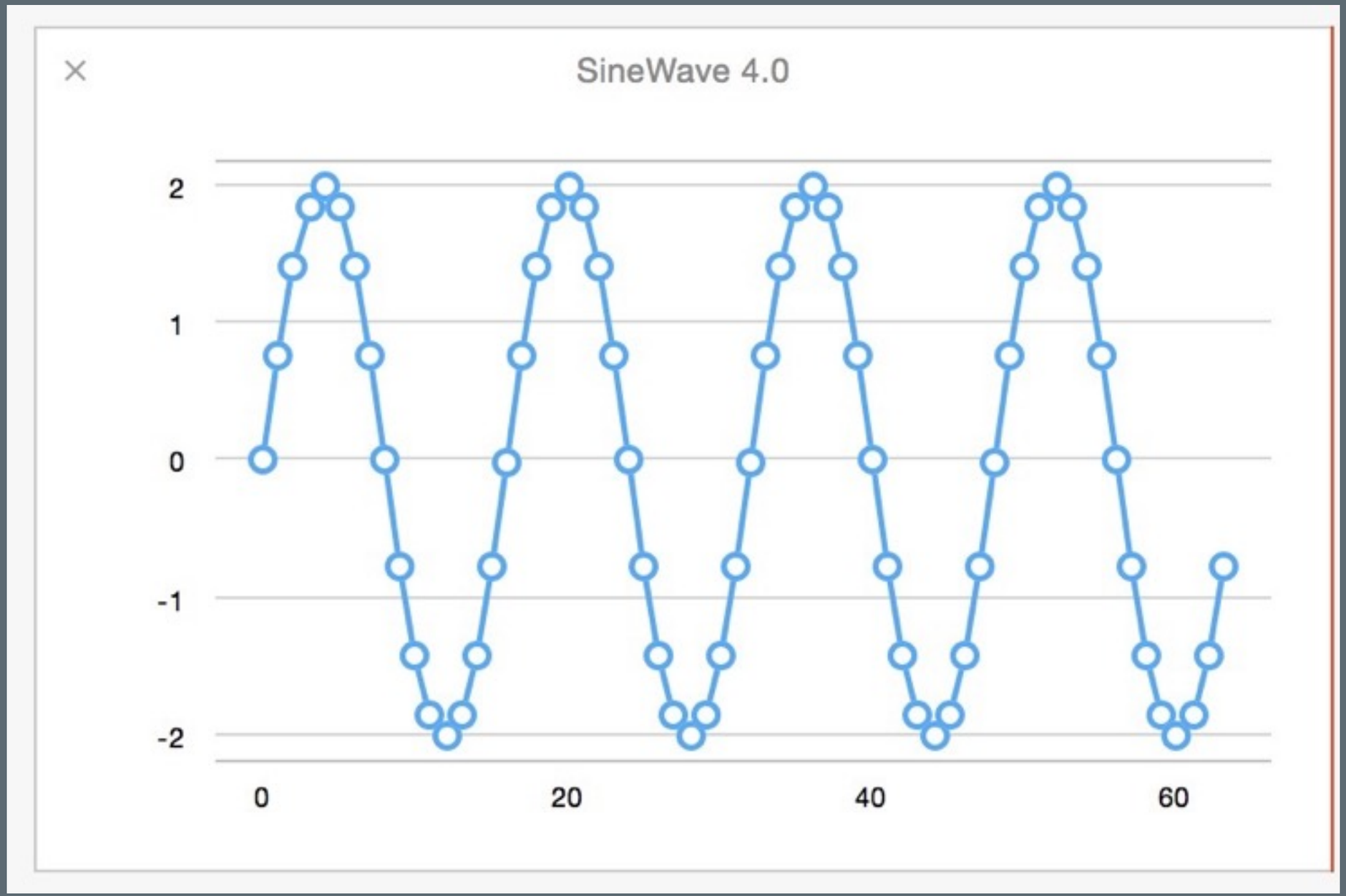
```
"0..<101"

64
4.0
0.0
2.0
[0.0, 0.7653668647301..
(64 times)
```

Assistant Editor

- View > Assistant Editor > Show Assistant Editor

XCPCaptureValue



XCPShowView

- Lets import some more libraries.
- UIKit
- QuartzCore
- XCPlayground

```
import UIKit
import XCPlayground
import QuartzCore

let zoom:CGFloat = 0.5                                                    0.5
let deviceSize = CGSize(width: 640*zoom, height: 1136*zoom)               {w 320 h 568}

let helloWorldView = UIView(frame: CGRect(x: 0, y: 0, width: deviceSize.width,   UIView
    height: deviceSize.height))
helloWorldView.backgroundColor = UIColor(white: 0, alpha: 0.01)           UIView



let font = UIFont(name: "HelveticaNeue-Light", size: 62)                  {UICTFont}

let label = UILabel(frame: CGRect.zeroRect)                               UILabel
label.frame.origin.y = 100                                                {x 0 y 100}
label.text = "Hello\nWorld!"                                              UILabel
label.numberOfLines = 3                                                   UILabel
label.font = font                                                         UILabel
label.sizeToFit()                                                         UILabel
helloWorldView.addSubview(label)                                          UIView

XCPShowView("iphone", helloWorldView)
```

iphone

Hello
World!

And add a button

```
let button   = UIButton.buttonWithType(UIButtonType.System) as UIButton
button.frame = CGRectMake(deviceSize.width/2 - 50, deviceSize.height-130, 100, 50)
button.setTitle("Test Button", forState: UIControlState.Normal)
//button.addTarget(self, action: "buttonAction:", forControlEvents:
    UIControlEvents.TouchUpInside)

helloWorldView.addSubview(button)

XCPShowView("iphone", helloWorldView)
```

iphone

Hello
World!

Test Button

XCPlayground doesn't have UI events

- In order to plug this button into something, we need to create an iOS project. Let do that.
- New > New Project
- Single View Application
- Language: Swift
- No Core Data

ViewController.swift

- Copy the code from the playground into the function viewDidLoad, after super.viewDidLoad
- Move the imports to the top of the file
- Remove import XCPlayground
- Change XCPShowView("iphone", helloWorldView) to view.addSubView(helloWorldView)
- Run it in the Simulator!

Fixing the background size

- Change the deviceSize line to:

- let deviceSize = UIScreen.mainScreen().bounds;

Lets wire up the button

- Uncomment the addTarget line:
- `button.addTarget(self, action: "buttonAction:", forControlEvents: UIControlEvents.TouchUpInside)`

```
@IBAction func buttonAction( sender : AnyObject) {
    println( "Pushed the button!" )
}
```

Press the button

- You should see “Pushed the button!” in the Xcode console!
- @IBAction annotates the method so you can wire it up to the views. We have been making manual views here, lets do it from the Interface Builder and Storyboard instead.

Create a new Project

- Open Storyboard
- Select Object Viewer on the right
- Drag label in the view
- Attribute editors for label
- Font 62 pts, lines 2

Create a Button

- Change the text to be “Test Button”
- Right click, scroll to “Touch Up Inside”
- Click on the “+” and drag to the view
- Select “buttonAction”
- Press Run

Much less code

- Lets make it so the button changes the label text, instead of just printing on the console.
- Add an @IBOutlet to the ViewController, and reference that in the buttonAction Method

```
@IBOutlet var helloLabel : UILabel!

@IBAction func buttonAction( sender : AnyObject) {
    println( "Pushed the button!" )
    helloLabel.text = "Hello Button!";
}
```

Go back to the storyboard

- Right click on the label itself.
- Drag “New Referencing Outlet” to the View Controller Object on top
- Select “helloLabel”.
- Rerun!

@IBOutlet and @IBAction

- Use @IBOutlet to get references to Interface Builder Elements.
- Use @IBAction to attach methods to those elements.

Lets add a new View

- Drag a View Controller on the story board
- Add a label
- Add a button
- Right click and drag the button on the first page to the second page
- Do the reverse with the reverse button

Temperature Converter

- Create a new project, single page
- In ViewController, enter the following outlets and actions:

```
@IBOutlet var slider: UISlider!
@IBOutlet var temperatureInC: UILabel!
@IBOutlet var temperatureInF: UILabel!

@IBAction func convertFotC( sender: AnyObject ) {
```

Storyboard

- Add a slider and a couple labels
- Set the slider delegate by right clicking, adding a reference outlet, and dragging to the view on the top.
- Do the same for the labels, connecting them to temperatureInC and temperatureInF

View Controller

- Also add a call to convertFtoC in viewDidLoad.
- Add this method to the class.

```
@IBAction func convertFotC( sender: AnyObject ) {
    var c = ((slider.value - 32) * 5) / 9;

    temperatureInF.text = "\(slider.value) degrees F"
    temperatureInC.text = "\(c) degrees C"
}
```

Now we have a fun slider

CoreLocation

- iOS devices know where they are, using a combination of the GPS and Cell Towers.
- This service is called CoreLocation. (Also on OS X)

Import CoreLocation, configure the manager

```
import UIKit
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {
    let locationManager:CLLocationManager = CLLocationManager()

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        convertFotC("")

        locationManager.delegate = self
//        locationManager.desiredAccuracy = kCLLocationAccuracyBest
        locationLabel.text = "Looking for location"

        locationManager.requestAlwaysAuthorization()
        locationManager.startUpdatingLocation()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
```

Implement the CLLocationManagerDelegate

```
func locationManager(manager: CLLocationManager!, didUpdateLocations locations: [AnyObject]!) {
    var location:CLLocation = locations[locations.count-1] as CLLocation

    locationLabel.text = "Lat \(location.coordinate.latitude) Long \(location.coordinate.longitude)"

    if (location.horizontalAccuracy > 0) {
        self.locationManager.stopUpdatingLocation()
        locationLabel.text = "Lat \(location.coordinate.latitude) Long \(location.coordinate.longitude)"
    }
}

func locationManager(manager: CLLocationManager!, didFailWithError error: NSError!) {
    println(error)
    locationLabel.text = "Can't get your location!"
}
```

Now the label

- @IBOutlet var locationLabel: UILabel!

- And in the story board, create the label and wire it up.

Now Configure the app

- In supporting files, edit “Info.plist”
- Add “NSLocationAlwaysUsageDescription"

Now the app knows where you are

- In the simulator, you can configure the location in Debug > Location

Next Steps

- Learn the Storyboard
- Learn Auto Layout
- Learn more about ViewControllers
- Learn more of the awesome libraries, like CoreAnimation or networking.

Conclusion

Swift is very new

- It’s a modern rethinking of how to build Objective-C apps.
- You have full access to the existing libraries, they aren’t going away.
- The tooling is nice, but the language is still in flux.
- Examples on the web are often out of date.

What should I learn?

- If all you care about is building an app, learn Swift.

- If you want to get a job in App development, you will need to know Objective-C.

- Some things (e.g. Frameworks) can only be written in Objective-C.

It’s not really about the language

- Understanding the libraries, the tooling, and all of the details is the hard thing.

- The language helps, but you will still need to learn a lot of stuff.

- Luckily you can take it one piece at a time.

The End

Will Schenk

@wschenk

willschenk.com

HappyFunCorp.com

hfcacademy.com

github: hfc-tech-academy/Intro-To-Swift