Lecture 12

SwiftUI & Moving On

CIS 195-201 - Dominic Holmes

Final Project Schedule

- ~~Monday April 13th — **Milestone 1 Due**~~
  - ~~*Text Post: detail your idea, explain your challenges and upload a photo of your app's wireframe (hand drawn or digital)*~~
  - ~~*Code Submission: submit at least a created Xcode project*~~
- Friday April 17th — **Milestone 2 Due**
  - *By this point you should have a clear runway ahead. Aim for being >= 30% done.*
  - *Text Post: explain your progress so far and detail any future challenges*
  - *Code Submission: submit a zipped Xcode project, hopefully with good progress.*
- Thursday April 23rd — **Demo Days, Project Due**

Today

- SwiftUI
- My go-to resources
- Where to go from here

SwiftUI

What is SwiftUI?

- **An absolutely incredible new way to make apps**

What is SwiftUI?

- **An absolutely incredible new way to make apps**
- A powerful user interface toolkit that lets us design apps in a declarative way.
    - That means we *tell* SwiftUI how we want our app to look, and it figures out how to make that happen.
    - Declarative UI is best understood as compared to *Imperative* UI.
- *SwiftUI will replace UIKit, (& AppKit, WatchKit, and TVKit) as the way to build apps across Apple Platforms*.

Imperative (UIKit)

- Let’s make coffee!

Declarative (SwiftUI)

Imperative (UIKit)

- Gather all required supplies.
- Plug in machine. Clean it before beginning.
    - Place the machine at these precise coordinates. Clean with a mixture of 90% warm water, 10% soap.
- Measure out 6 spoonfuls of coffee grounds. 3 is regular, but we want extra strength.
- Put the grounds in the espresso basket. Tamp them down.
- Turn on the machine and wait.
- When the coffee finishes, pour into a cup.
- Add milk.

Declarative (SwiftUI)

Imperative (UIKit)

- Gather all required supplies.
- Plug in machine. Clean it before beginning.
    - Place the machine at these precise coordinates. Clean with a mixture of 90% warm water, 10% soap.
- Measure out 6 spoonfuls of coffee grounds. 3 is regular, but we want extra strength.
- Put the grounds in the espresso basket. Tamp them down.
- Turn on the machine and wait.
- When the coffee finishes, pour into a cup.
- Add milk.

Declarative (SwiftUI)

- **Make me an espresso.**
    - **... with an extra shot.**
    - **... with milk.**

Put simply…

- Rather than *enumerate the steps to create interface X*, we just say tell SwiftUI how we want our UI to look and work
    - SwiftUI takes it from there.
    - That means there are a lot of assumptions baked in, and that we can't customize certain properties we could before.
        - SwiftUI makes up for this by significantly expanding the level of customization in pretty much all other areas. Want your button to be a gradient? No problem.

But… what's wrong with UIKit?

- 3 things*

* that I could think of

1

UIKit is verbose

To build a simple UITableView, we need like 6 delegate functions. To create and configure a UIButton in code, we need entire functions. To hook anything up we need @IBActions, @IBOutlets, segues……

SwiftUI is succinct

Button { Text(“Tap me!”) }

Imperative UI


```
class ViewController: UIViewController {

    var imageView: UIImageView!
    var nameLabel: UILabel!
    var handleLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        createViews()
    }

    func createViews() {
        imageView = UIImageView()
        imageView.image = UIImage(named: "stevejobs")
        imageView.translatesAutoresizingMaskIntoConstraints = false
        view.addSubview(imageView)

        nameLabel = UILabel()
        nameLabel.text = "Steve Jobs"
        view.addSubview(nameLabel)

        handleLabel = UILabel()
        handleLabel.text = "@stevesie"
        view.addSubview(handleLabel)
    }

    func constrainViews() {
        imageView.translatesAutoresizingMaskIntoConstraints = false
        let horizontalConstraint = imageView.leadingAnchor.constraint(equalTo: view.leadingAnchor)
        let verticalConstraint = imageView.centerYAnchor.constraint(equalTo: view.centerYAnchor)
        let widthConstraint = imageView.widthAnchor.constraint(equalToConstant: 40)
        let heightConstraint = imageView.heightAnchor.constraint(equalToConstant: 40)
        NSLayoutConstraint.activate([horizontalConstraint, verticalConstraint, widthConstraint, heightConstraint])

        nameLabel.translatesAutoresizingMaskIntoConstraints = false
        let labelXConstraint = nameLabel.leadingAnchor.constraint(equalTo: view.centerXAnchor)
        let labelYConstraint = nameLabel.centerYAnchor.constraint(equalTo: view.centerYAnchor)
        let labelWConstraint = nameLabel.widthAnchor.constraint(equalToConstant: 100)
        let labelHConstraint = nameLabel.heightAnchor.constraint(equalToConstant: 100)
        NSLayoutConstraint.activate([labelXConstraint, labelYConstraint, labelWConstraint, labelHConstraint])

        handleLabel.translatesAutoresizingMaskIntoConstraints = false
        let handleXConstraint = nameLabel.centerXAnchor.constraint(equalTo: view.centerXAnchor)
        let handleYConstraint = nameLabel.centerYAnchor.constraint(equalTo: view.centerYAnchor)
        let handleWConstraint = nameLabel.widthAnchor.constraint(equalToConstant: 100)
        let handleHConstraint = nameLabel.heightAnchor.constraint(equalToConstant: 100)
        NSLayoutConstraint.activate([handleXConstraint, handleYConstraint, handleWConstraint, handleHConstraint])
    }
}
```

This view in UIKit

Declarative UI Demo


```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {
            Image("stevejobs")
                .resizable()
                .scaledToFill()
                .frame(width: 100, height: 100)
                .clipShape(Circle())
            VStack(alignment: .leading) {
                Text("Steve Jobs")
                    .font(.title)
                Text("@stevesie")
                    .font(.subheadline)
            }
        }
        .padding()
    }
}
```

This view in SwiftUI

UIKit Development Time

Basic UI and Setup

Awesome Custom Features

SwiftUI Development Time

SwiftUI Development Time

Basic UI and Setup

Awesome Custom Features

The goal of SwiftUI: You spend your time building custom components, rather that setting up Table view delegate functions!

2

UIKit and Storyboards are disconnected

Remember how long we spent in this class trying to figure out the Xcode interface? How many bugs came from people forgetting to hook up a single option in a menu somewhere? That sucks!!

SwiftUI has no storyboards

Instead, it has a preview view that ***updates with the code***

Problem solved. I'll show this in the Demo.

3

UIKit is imperative

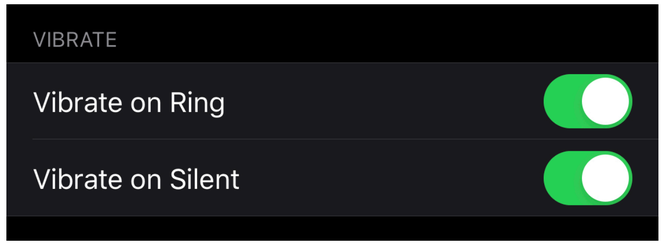
We quickly reach a state in UIKit apps where we can no longer reason about all the possible states a user could be in. That's dangerous!

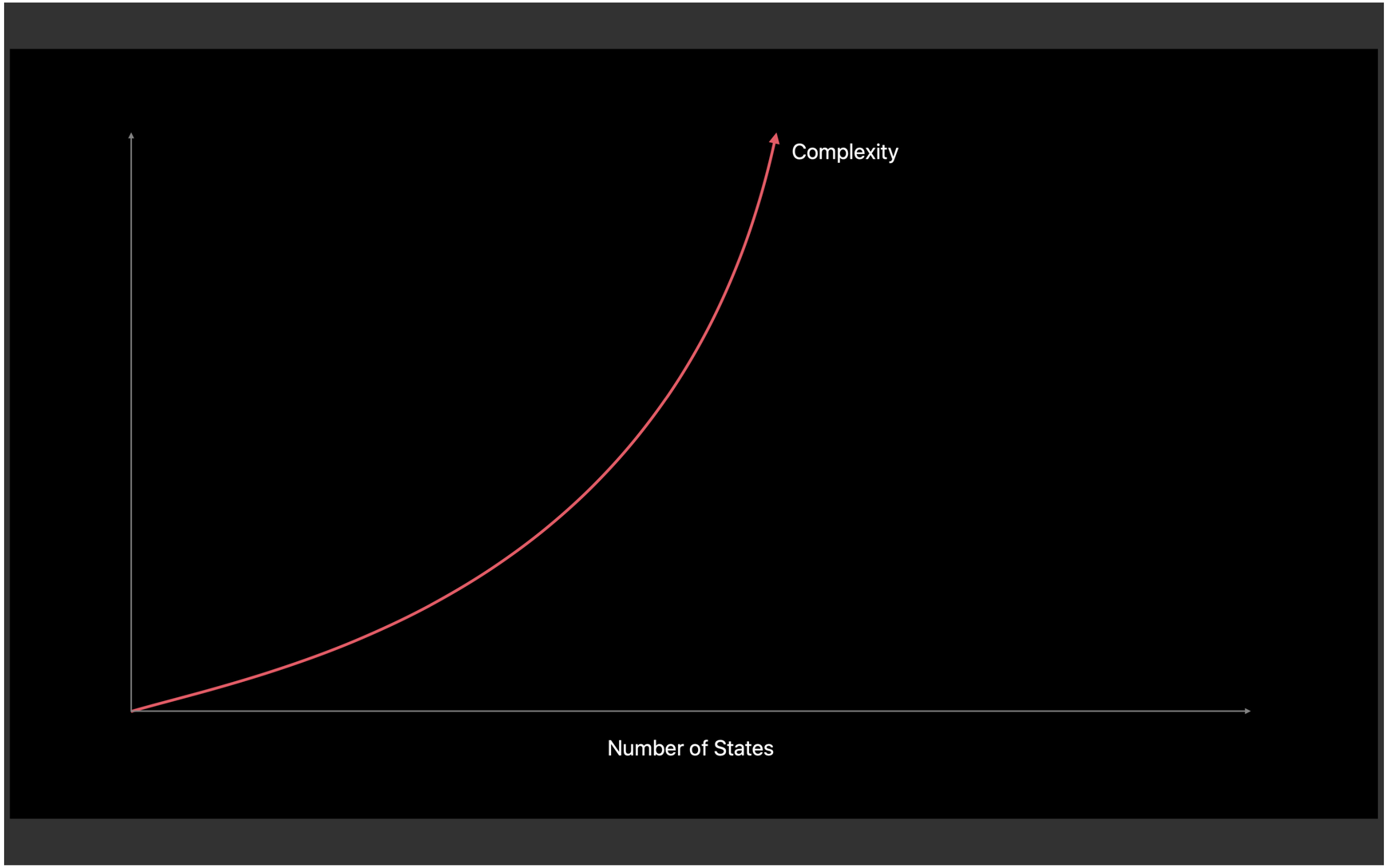
SwiftUI is declarative

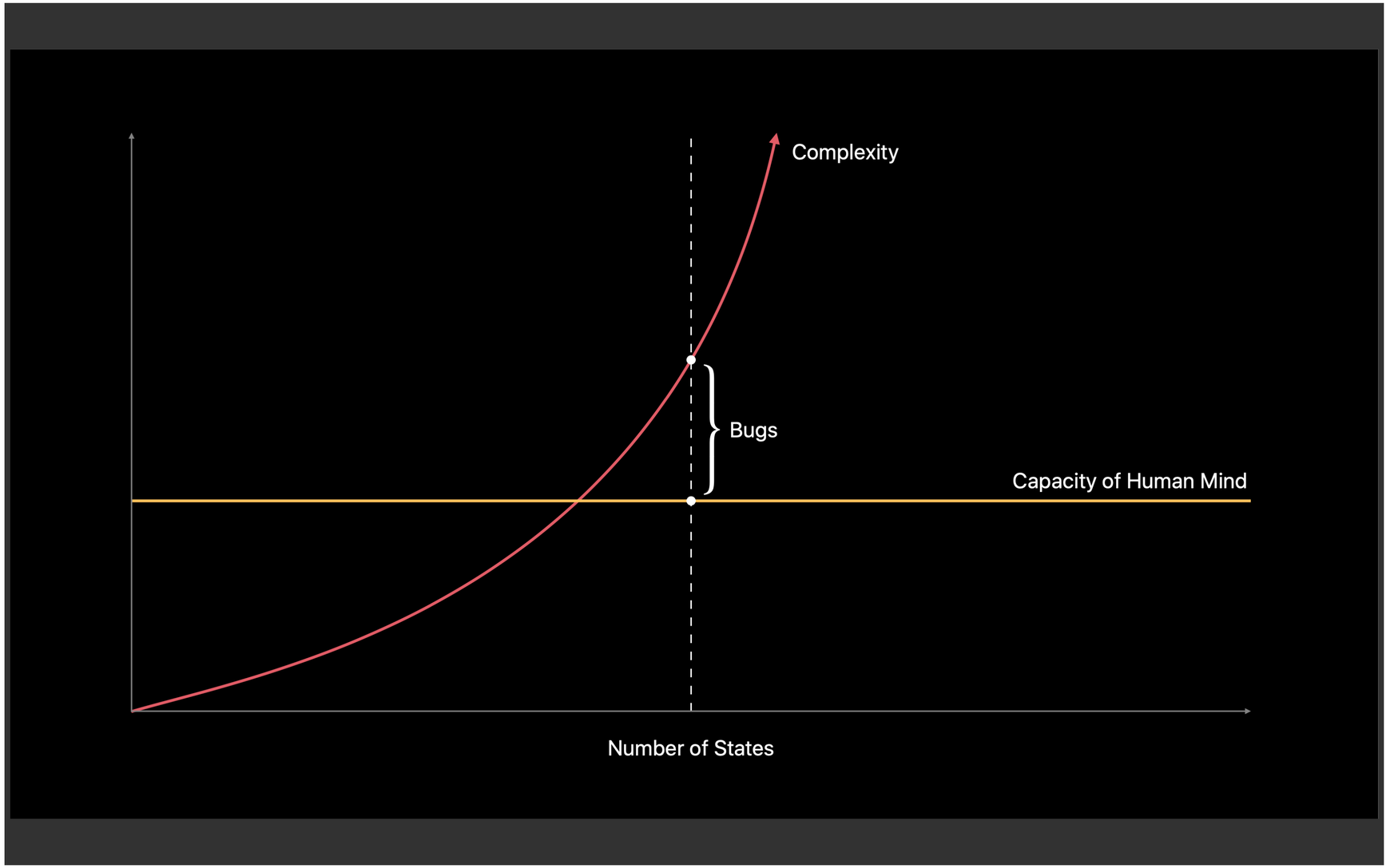
State is built-in.

Reactive > Imperative

- The problem is **state**
  - **In all interfaces — we must track what state our code is in, and make sure our UI correctly reflects that state**
- Say we have one Boolean that affects the UI, like “checkbox selected”
  - Now say we have 2 booleans — that gives 4 possible states
  - N booleans means 2^n states.
  - *Eventually impossible to reason about every state in an imperative way*




Complexity
Number of States


Complexity
Bugs
Capacity of Human Mind
Number of States

More advantages of SwiftUI (cont.)

- Animations are super easy & powerful
- Hot reloading means instant feedback
- AS IF THAT WASN'T ENOUGH....
  - SwiftUI is cross-platform! Learn it for iOS and you can port much of the code over to macOS, watchOS, and tvOS
    - Not "write once run anywhere". You still need to adapt your app for the paradigms of the platforms. Instead, "learn once, write anywhere".

The advantages of SwiftUI

- No more storyboards — a visual editor reflects the code **1 to 1** (!!!)
- Way less boilerplate — no more setting up table view delegates!
  - In my experience, SwiftUI takes about **~70% less code** (holy ****)
  - This means more maintainable codebases
- SwiftUI does a much better job of exposing and using advanced iOS features, like dynamic type, accessibility, and dark mode

This changes everything!

Drawbacks of SwiftUI

- Beta-level error reporting (really bad)
  - This has improved over time!
- Incomplete coverage for important frameworks
  - No UICollectionView
- iOS 13+
- These factors combine to make SwiftUI at least 1-2 years away from mainstream adoption

If you’re okay with a few rough edges, its an amazing experience.

Let’s check it out!

LIVE DEMO

A SwiftUI Tour

Onward!

Where to go from here

- If you like iOS
  - Keep building! It's an amazing platform that's only increasing in importance
- If you don't like iOS (._.)
  - It's not for everyone — keep hacking until you find something you love. But give SwiftUI a try :)

Continued iOS Education

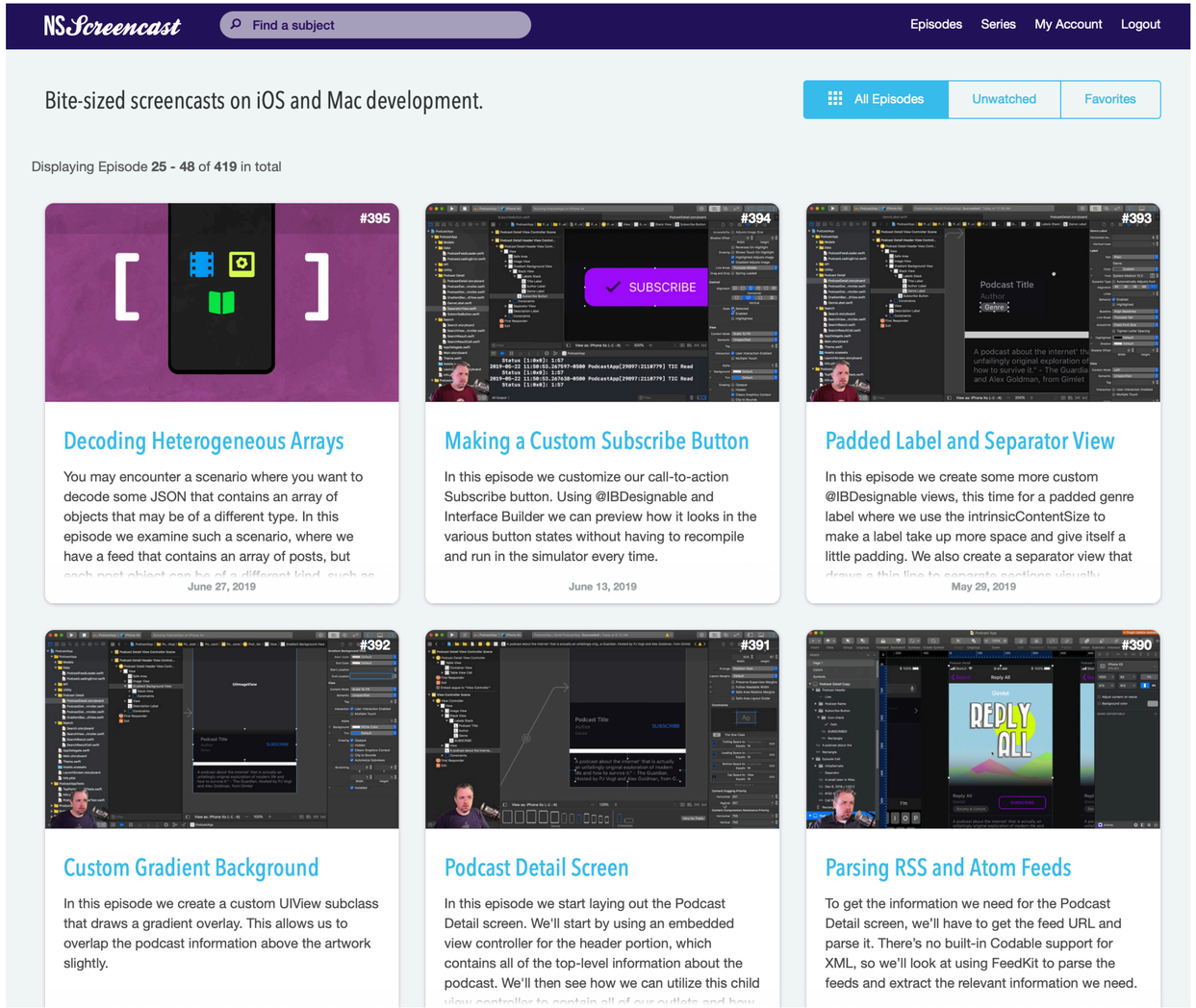
- UIKit
  - Advanced books available by Ray Wenderlich and HackingWithSwift. You can build anything you want with UIKit, just a matter of learning the frameworks.
- **SwiftUI (!!!!)**
  - I fully recommend **100 Days of SwiftUI** by HackingWithSwift. It's free, incredible well-paced, and has really interesting projects (honestly, more interesting than my homeworks! I tried!).
- Something else (ReactNative, Flutter, etc..)
  - I wouldn't really recommend this unless you're big into React already — but anything you learn here will make you a better dev so... go for it!

My Go-To Resources

- HackingWithSwift
  - Has lots of useful *stub code*, but often pretty shallow.
  - Has a few full-length textbooks and free tutorial series that are absolutely fantastic, well-paced, and deep. This guy can do no wrong.
- Ray Wenderlich
  - I learned on their textbook (iOS Apprentice) — it was pretty good!
  - They have tons of really useful, well researched, long tutorials. Often I find the answer to my question embedded inside one of those, explained well.

My Go-To Resources (cont.)

- NSScreenCast
  - Costs $15 / month
  - Like Youtube, but way higher quality. Whenever there's an NSScreenCast episode on a topic, I watch that first before trying anywhere else!
  - Worth it if you decide to "get serious"
  - Regularly goes way beyond the basics for really advanced stuff



My Go-To Resources (cont. cont.)

- Youtubers
  - *Brian Advent*
  - *Lets Make That App*
  - Many others...
- They mostly stay with the basics... but if you need refreshers on those basics, they're great.
- Be careful of outdated videos.

My Go-To Resources (cont. cont.)

- Blogs (best first)
  - https://nshipster.com
  - https://www.avanderlee.com
  - https://swift.org/blog/
  - https://www.swiftbysundell.com
  - https://medium.com/ios-os-x-development

SwiftUI Resources

- Apple's Tutorials (surprisingly, very good)
  - https://developer.apple.com/tutorials/swiftui/tutorials
- 100 Days of SwiftUI (recommended)
  - https://www.hackingwithswift.com/100/swiftui
- SwiftUI by Example
  - https://www.hackingwithswift.com/quick-start/swiftui

Where to go from here

- Keep making apps!
  - With every project you'll get better and better.
  - Even if the app has already been made — *just do it better*.

- Freelancing and contract work
  - Amazing if you can get going with it.
  - Beware — make sure to get paid. Never ever ever work for free or "experience" :)

Apply to Penn Labs!

- pennlabs.org
  - We recruit in fall & spring
  - Apply again if you don’t get in the first time (its getting popular)
  - This has been the single biggest catalyst for my dev skills.
  - Also, just a fantastic group of people.

Penn Labs

100,000+ total users

across over 10 student-developed applications.

60+ GitHub contributors

building high-quality open source software.

7+ years

of supporting the Penn community at large.

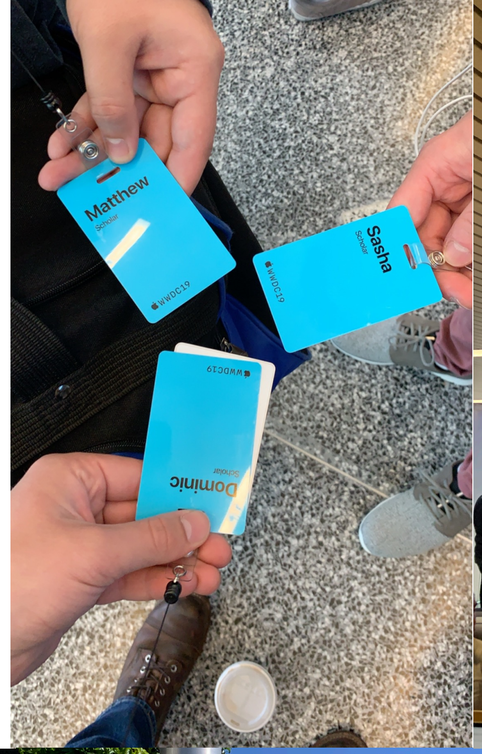
$930,000+ handled

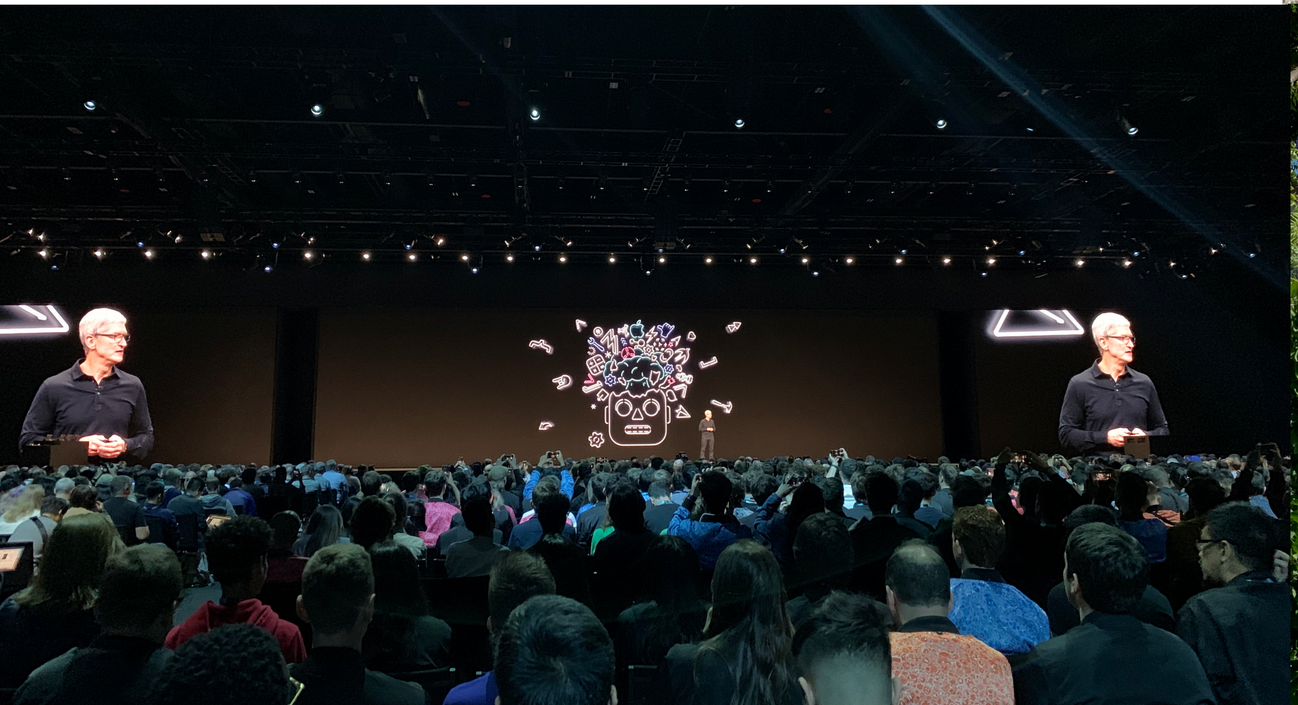
through our Common Funding Application portal.

WWDC19

WWDC

- Happens in early June. **Scholarships available for students.**
- One of my favorite life experiences.
- Unfortunately online this year. **Watch online this year, and apply next year!**





That’s it!

Thanks for a great semester :)