# Build iOS apps with SwiftUI

Present by Wing Chan

## Introduction

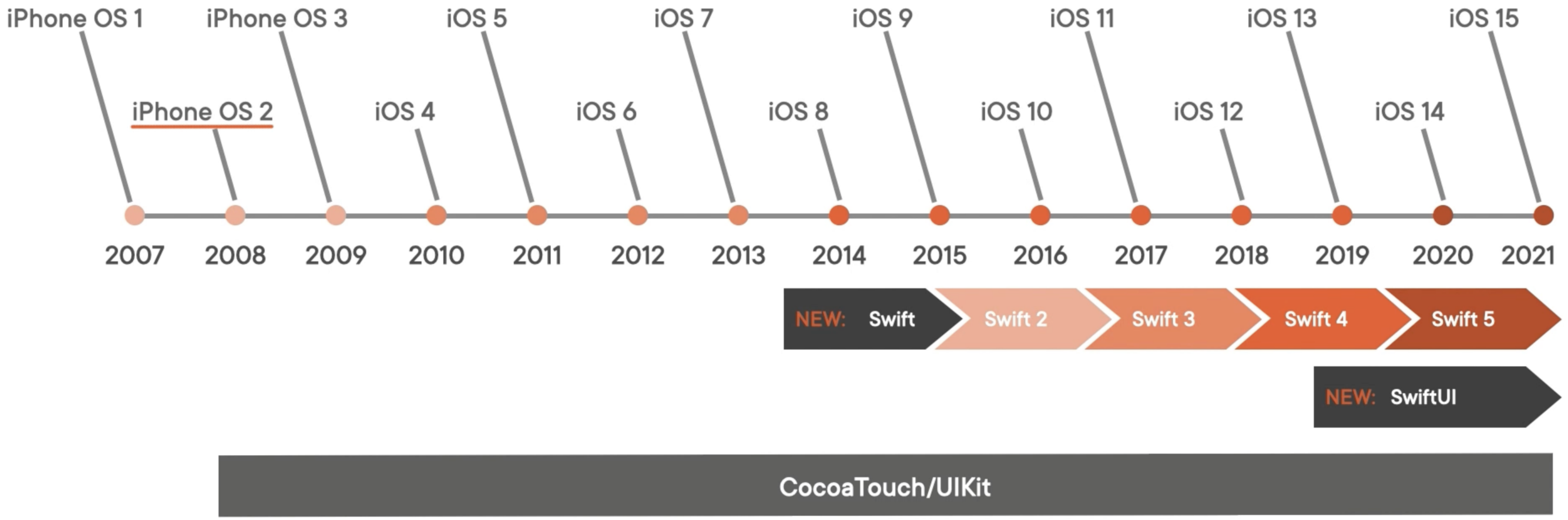
SwiftUI is a modern and intuitive framework for building iOS apps.

It provides a declarative approach to app development, making it easier and faster to create stunning user interfaces.

In this presentation, we will explore the key features and benefits of SwiftUI and demonstrate how to build iOS apps using SwiftUI.

## History

- Swift was first announced by Apple at the Apple Worldwide Developers Conference (WWDC) in June 2014.
- Swift was designed to address the limitations of Objective-C, which was the primary programming language for Apple's platforms at the time. It aimed to provide a more efficient, expressive, and beginner-friendly language.
- SwiftUI was introduced by Apple at WWDC 2019 as a modern framework for building user interfaces across Apple platforms.
- SwiftUI is designed to be cross-platform, supporting iOS, macOS, watchOS, tvOS and visionOS with a shared codebase.
- It is compatible with iOS 13 and later.



## Key features

DECLARATIVE SYNTAX

SWIFT INTEGRATION

LIVE PREVIEWS.

NATIVE PERFORMANCE

## Elements

- Views
- Modifiers
- Layouts
- Gestures
- Animation
- Integration with UIKit and AppKit

# State and Data Flow

## State

State represents the dynamic and mutable data within a SwiftUI view.

It is declared using the **@State** property wrapper.

When a state property changes, SwiftUI automatically updates the associated view.

## Data Flow

In SwiftUI, data flows unidirectionally from top to bottom.

Views receive data from their parent views and pass it down to their child views.

Data flow is achieved through property bindings and environment variables.

## Property Bindings

A binding is a two-way connection between a view and its underlying data.

It allows changes made in the view to update the data and vice versa.

Property bindings are created using the **$** symbol, e.g., **$property**.

## Environment Variables

Environment variables are used for sharing data across multiple views.

They are defined using the **@EnvironmentObject** property wrapper.

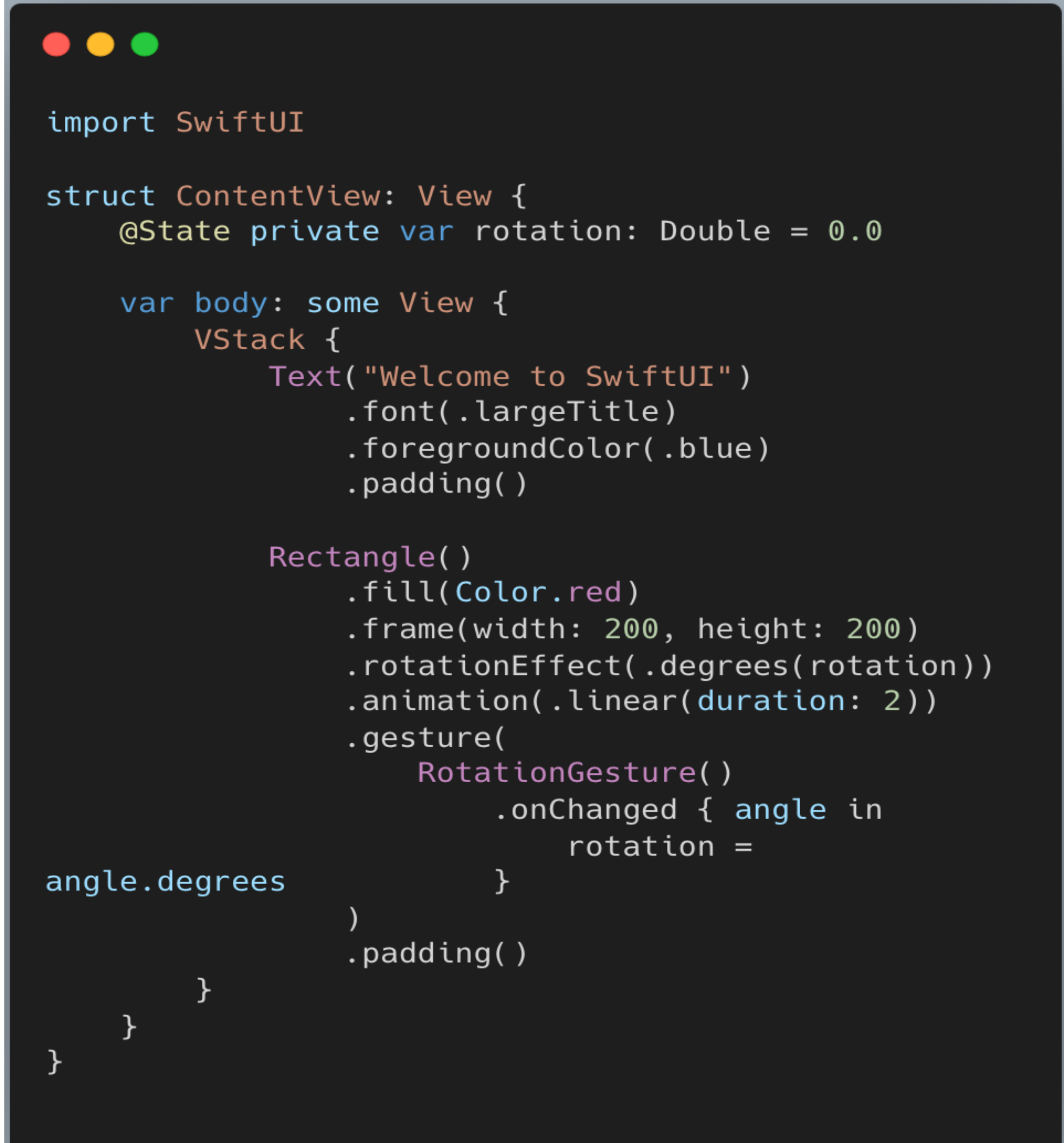
Environment variables can be accessed by any view within the same environment scope.

## Managing State and Data Flow

For simple, localized state, use **@State** within a view.

For shared state across multiple views, consider using **@EnvironmentObject** or passing data through views using property bindings.

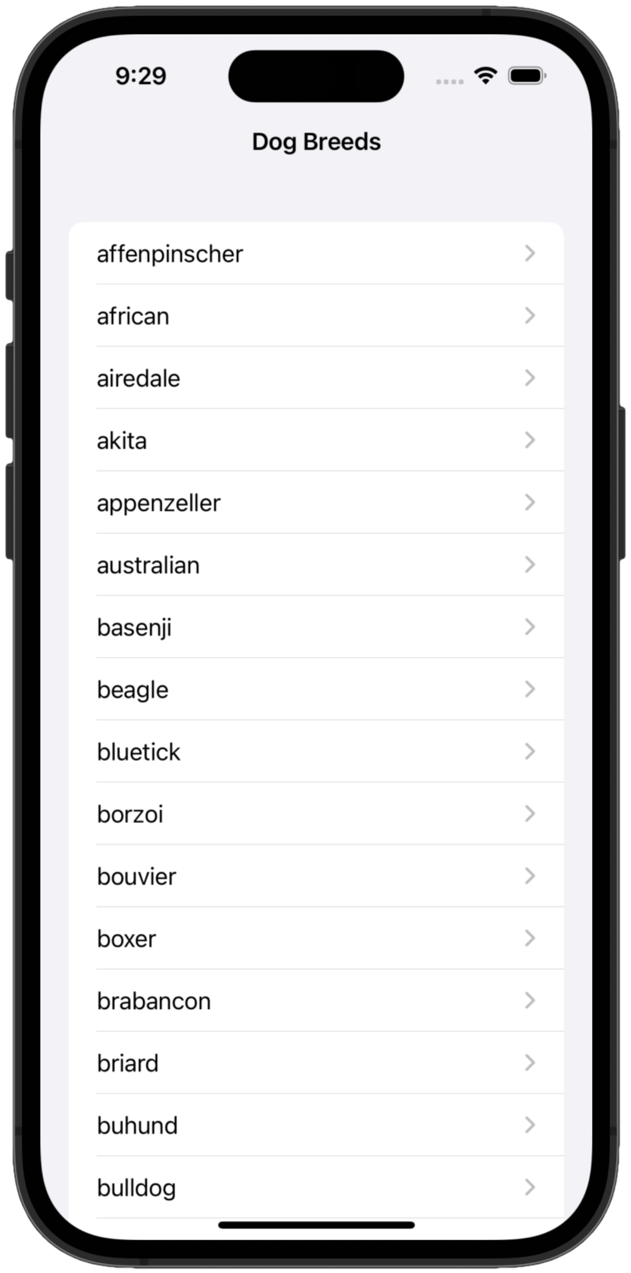
## Code sample



```
import SwiftUI

struct ContentView: View {
    @State private var rotation: Double = 0.0

    var body: some View {
        VStack {
            Text("Welcome to SwiftUI")
                .font(.largeTitle)
                .foregroundColor(.blue)
                .padding()

            Rectangle()
                .fill(Color.red)
                .frame(width: 200, height: 200)
                .rotationEffect(.degrees(rotation))
                .animation(.linear(duration: 2))
                .gesture(
                    RotationGesture()
                        .onChanged { angle in
                            rotation =
angle.degrees            }
                )
                .padding()
        }
    }
}
```

# Demo

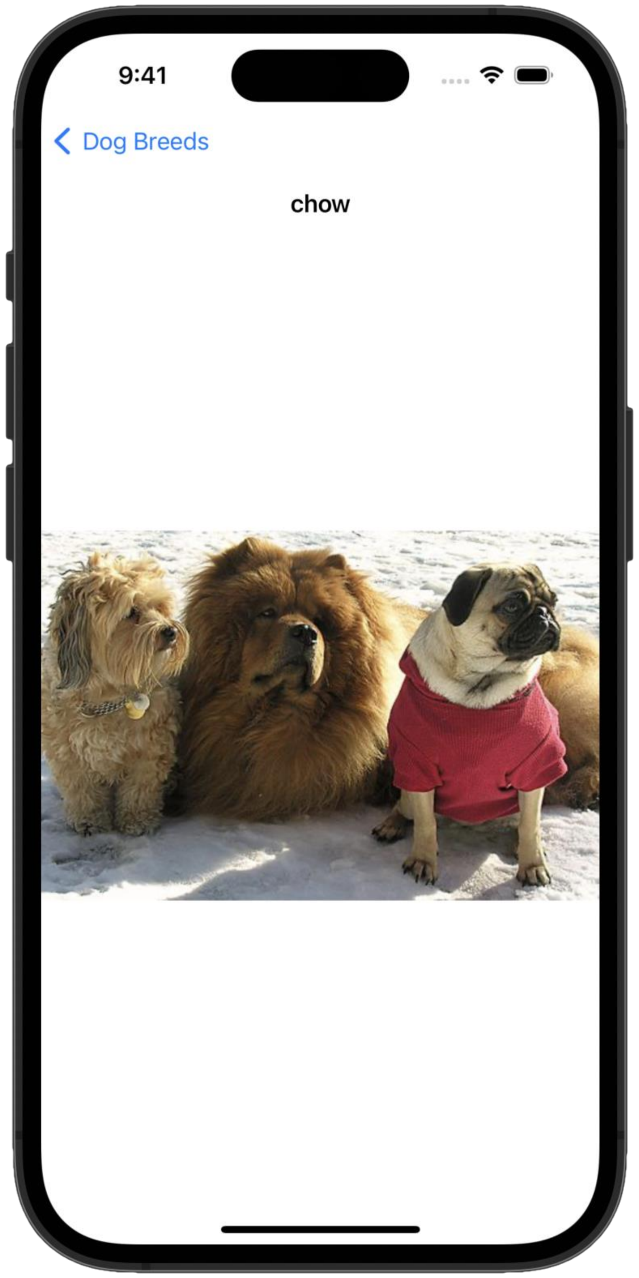
- Dog Breeds App
  - Data source: REST APIs
  - Design Pattern: MVVM
  - Framework: SwiftUI



9:29

Dog Breeds

affenpinscher
african
airedale
akita
appenzeller
australian
basenji
beagle
bluetick
borzoi
bouvier
boxer
brabancon
briard
buhund
bulldog



9:41

Dog Breeds

chow