

State Driven Development

The Beauty of Enums in Swift



@conradstoll



State Driven Development

enum

Bool

```
var instanceVariable : Bool
```

How often exactly?

```
// Regex for boolean state  
// Match Bool typed vars that aren't computed properties  
var.*:.*Bool(\n|.*=.*)  
var.*=\s?(true|false)
```



Xcode

File

Edit

View

Find

Navigate



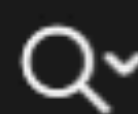
Grocery >



iPhone 8 Plus



Find > Text > Containing



Text



In Grocery.xcodeproj

Ignoring Case

```
var active : Bool
var contentLoaded : Bool
var dataLoaded : Bool
var editing : Bool
var enabled : Bool
var expanded : Bool
var handlingTouches : Bool
var hasTakenUserAction : Bool
var isCompleted : Bool
var isRunning : Bool
var lastTouchedButtonSide : Bool
var lastTouchedRightSide : Bool
var notified : Bool
var previewEnded : Bool
var saved : Bool
var started : Bool
```

Example



Starting a Workout

```
func startWorkout() {  
    // ...  
}
```

One variable

```
var workoutInProgress : Bool = false

func startWorkout() {
    guard workoutInProgress == false else {
        return
    }

    workoutInProgress = true

    // ...
}
```

Two variables

```
var hasLoggedIn : Bool = false
var workoutInProgress : Bool = false

func startWorkout() {
    guard hasLoggedIn && workoutInProgress == false else {
        return
    }

    workoutInProgress = true

    // ...
}
```

The tangled web of state

```
var hasLoggedIn : Bool = false
var workoutInProgress : Bool = false
var savingWorkout : Bool = false

func stopWorkout() {
    workoutInProgress = false
    savingWorkout = true

    saveWorkout { (success) in
        self.savingWorkout = false
    }
}
```

The tangled web of state

```
var hasLoggedIn : Bool = false
var workoutInProgress : Bool = false
var savingWorkout : Bool = false

func startWorkout() {
    guard hasLoggedIn && workoutInProgress == false && !savingWorkout else {
        return
    }

    workoutInProgress = true

    // ...
}
```

All of the Possible
Boolean States

Valid boolean states

User Not Logged In, No Workout, Not Saving

User Logged In, No Workout, Not Saving

User Logged In, Workout In Progress, Workout Not Saving

User Logged In, No Workout, Workout Is Saving

```
var hasLoggedIn : Bool
```

```
var workoutInProgress : Bool
```

```
var savingWorkout : Bool
```

Invalid boolean states

User Not Logged In, Workout In Progress, Not Saving

User Not Logged In, Workout In Progress, Workout Is Saving

User Not Logged In, No Workout, Workout Is Saving

User Logged In, Workout In Progress, Workout Is Saving

```
var hasLoggedIn : Bool
```

```
var workoutInProgress : Bool
```

```
var savingWorkout : Bool
```

Perfect case for an enum

Four valid states

- Hasn't logged into the app yet
- In the middle of a workout
- Saving the last workout
- Idle

Four enum cases

```
enum WorkoutApplicationLifecycle {  
    case notLoggedIn  
    case workoutInProgress  
    case savingLastWorkout  
    case idle  
}
```

Enums make invalid states
impossible to represent!

Enums help programmers
ask *specific* questions!

Can I start a workout?



The *vague* question with BOOLs

```
var hasLoggedIn : Bool = false
var workoutInProgress : Bool = false
var savingWorkout : Bool = false

func startWorkout() {
    guard hasLoggedIn && workoutInProgress == false && !savingWorkout else {
        return
    }

    workoutInProgress = true

    // ...
}
```

The *vague* question with enums

```
var lifecycle : WorkoutApplicationLifecycle = .notLoggedIn

func startWorkout() {
    guard lifecycle == .idle else {
        return
    }

    workoutInProgress = .workoutInProgress

    // ...
}
```

ActivityType

```
enum ActivityType {  
    case running  
    case walking  
    case yoga  
}
```

Is this a running activity?

```
if activityType == .running {  
    gpsProvider.start()  
}
```

```
if activityType == .running {  
    gpsProvider.stop()  
}
```

Is this a running or walking activity?

```
if activityType == .running ||  
    activityType == .walking {  
    gpsProvider.start()  
}  
  
if activityType == .running {  
    gpsProvider.stop()  
}
```

Extend enums with
variables!

Does this activity support GPS?

```
enum ActivityType {  
    case running  
    case walking  
    case yoga  
  
    var supportsGPS : Bool {  
        switch self {  
            case .running, .walking:  
                return true  
            case .yoga:  
                return false  
        }  
    }  
}  
  
if activityType.supportsGPS {  
    gpsProvider.start()  
}
```

Use *enum extensions* to
describe custom behavior

Answers to specific questions

```
enum WorkoutApplicationLifecycle {  
    case notLoggedIn  
    case workoutInProgress  
    case savingLastWorkout  
    case idle  
  
    var canStartWorkout : Bool {  
        get {  
            switch self {  
                case .notLoggedIn, .workoutInProgress, .savingLastWorkout:  
                    return false  
                case .idle:  
                    return true  
            }  
        }  
    }  
}
```

The temptation of BOOL

```
var workoutInProgress : Bool = false

func startWorkout() {
    guard workoutInProgress == false else {
        return
    }

    // ...
}
```



var trap : Bool

IT'S A
TRAP

Use *enums* to describe
what the program is doing

Asking the specific question

```
var lifecycle : WorkoutApplicationLifecycle = .notLoggedIn

func startWorkout() {
    guard lifecycle.canStartWorkout else {
        return
    }

    lifecycle = .workoutInProgress

    // ...
}
```

Describing program state

```
var lifecycle : WorkoutApplicationLifecycle = .notLoggedIn

func stopWorkout() {
    lifecycle = .savingLastWorkout

    saveWorkout { (success) in
        self.lifecycle = .idle
    }
}
```

Enums describing custom behavior

```
enum WorkoutApplicationLifecycle {  
    case notLoggedIn  
    case workoutInProgress  
    case savingLastWorkout  
    case idle  
  
    var canStartWorkout : Bool  
    var actionTitle : String  
    var backgroundColor : UIColor  
    var barButtonItem : [UIBarButtonItem]  
}
```

Action title for buttons

```
enum WorkoutApplicationLifecycle {  
    var actionTitle : String {  
        get {  
            switch self {  
                case .notLoggedIn:  
                    return "Sign In"  
                case .workoutInProgress:  
                    return "End Workout"  
                case .idle:  
                    return "Start Workout"  
                default:  
                    return ""  
            }  
        }  
    }  
}
```

Background color for buttons

```
enum WorkoutApplicationLifecycle {  
    var backgroundColor : UIColor {  
        get {  
            switch self {  
            case .notLoggedIn:  
                return UIColor.blue  
            case .workoutInProgress: fallthrough  
            case .savingLastWorkout:  
                return UIColor.red  
            case .idle:  
                return UIColor.green  
            }  
        }  
    }  
}
```

Extend enums with
functions!

Actions for buttons

```
enum WorkoutApplicationLifecycle {  
  
    // ...  
  
    func actionTaken(from viewController : WorkoutViewController) {  
        switch self {  
        case .notLoggedIn:  
            viewController.login()  
        case .workoutInProgress:  
            viewController.stopWorkout()  
            viewController.saveWorkout()  
        case .savingLastWorkout:  
            break  
        case .idle:  
            viewController.startWorkout()  
        }  
    }  
}
```

Simple component interfaces

```
@IBAction func didTapActionButton() {  
    lifecycle.actionTaken(from: self)  
}
```

Extend enums with
new cases

Adding new states

```
enum WorkoutApplicationLifecycle {  
    case restoringWorkout  
}
```

- Can start workout? - false
- Action title? - default
- Background color? - UIColor.red
- Action taken? - break



Associated values

Defining associated values

```
enum WorkoutApplicationLifecycle {  
    // ...  
    case workoutInProgress(current : Workout)  
}
```

Accessing associated values

```
if case .workoutInProgress(let workout) = lifecycle {  
    // ...  
}
```

Pattern matching

```
if case .workoutInProgress = lifecycle { }
```

```
if case let .workoutInProgress(_) = lifecycle { }
```

```
if case let .workoutInProgress(current: workout) = lifecycle { }
```

```
if case let .workoutInProgress(workout) = lifecycle { }
```

```
if case .workoutInProgress(let workout) = lifecycle { }
```

Switch statement

```
switch self {  
    case .workoutInProgress:  
        // ...  
    case let .workoutInProgress(_):  
        // ...  
    case let .workoutInProgress(current: workout):  
        // ...  
    case let .workoutInProgress(workout):  
        // ...  
    case .workoutInProgress(let workout):  
}
```

Recommended syntax

```
if case let .workoutInProgress(current: workout) = lifecycle {  
    // ...  
}
```

```
switch self {  
    case let .workoutInProgress(current: workout):  
        // ...  
}
```

Setting associated values

```
var lifecycle : WorkoutApplicationLifecycle = .notLoggedIn

func startWorkout() {
    guard lifecycle.canStartWorkout else {
        return
    }

    var newWorkout : Workout = Workout()

    // ...

    lifecycle = .workoutInProgress(current: newWorkout)
}
```

Answering more questions

```
enum WorkoutApplicationLifecycle {  
    // ...  
    case workoutInProgress(current: Workout)  
  
    var navigationTitle : String {  
        switch self {  
            case let .workoutInProgress(current: workout):  
                return workout.type.activityName  
            }  
        }  
    }  
}
```

Highly relevant values

```
enum WorkoutApplicationLifecycle {  
    // ...  
    case workoutInProgress(current: Workout)  
  
    func actionTaken(from viewController : WorkoutViewController) {  
        switch self {  
        case let .workoutInProgress(current: workout):  
            viewController.stopWorkout()  
            viewController.saveWorkout(current: workout)  
        }  
    }  
}
```

Result¹

```
enum Result<Value> {  
    case success(Value)  
    case failure(Error)  
}
```

```
@frozen enum Result<Success, Failure> where Failure : Error {  
    case success(Success)  
    case failure(Failure)  
}
```

¹ <https://www.swiftbysundell.com/posts/the-power-of-result-types-in-swift>

Enum cases can provide context

```
protocol TableViewCellContent {  
    var titleText : String?  
    var descriptionText : String?  
    var image : UIImage?  
    var accessoryType : UITableViewCellStyleAccessoryType  
}
```

```
// Supported Configuration Options  
* Simple cell with one line of text  
* Detailed cell with two lines of text  
* Image Cell  
* Accessory Image Cell
```

Enums conform to protocols!

```
protocol TableViewCellContent {  
    var titleText : String?  
    var descriptionText : String?  
    var image : UIImage?  
    var accessoryType : UITableViewCellStyleAccessoryType  
}  
  
enum CellConfiguration : TableViewCellContent {  
    case simpleCell(title : String)  
    case detailedCell(title : String, description : String)  
    case imageCell(image : UIImage)  
    case accessoryCell(title : String,  
                       image : UIImage,  
                       accessory : UITableViewCellStyleAccessoryType)  
}
```

```
extension Workout {  
    var cellConfiguration : CellConfiguration {  
        return .detailedCell(title: workoutTitle,  
                               description: activityType.description)  
    }  
}  
  
extension User {  
    var cellConfiguration: CellConfiguration {  
        return .imageCell(image: profileImage)  
    }  
}
```

```
func tableView(_ tableView: UITableView,  
    cellForItemAt indexPath: IndexPath) -> UITableViewCell {  
    let cell = tableView.dequeueReusableCell  
        (withReuseIdentifier: reuseIdentifier,  
         for: indexPath) as! CustomTableViewCell  
  
    let workout = workouts[indexPath.row]  
  
    cell.setup(with: workout.cellConfiguration)  
  
    return cell  
}
```

```
class CustomTableViewCell : UITableViewCell {  
  
    @IBOutlet weak var titleLabel : UILabel?  
    @IBOutlet weak var descriptionLabel : UILabel?  
    @IBOutlet weak var imageView : UIImageView?  
  
    func setup(with content : TableViewCellContent) {  
        titleLabel?.text = content.titleText  
        descriptionLabel?.text = content.descriptionText  
        imageView?.image = content.image  
        accessoryType = content.accessoryType  
    }  
}
```

Enums can also be the
single source of truth

View controller customizations

```
protocol ViewControllerAppearance {  
    var title : String  
    var leftBarButtonItems : [UIBarButtonItem]  
    var rightBarButtonItems : [UIBarButtonItem]  
    var overlayAlpha : CGFloat  
    var activityIndicatorActive : Bool  
}  
  
enum ViewControllerState : ViewControllerAppearance {  
    case .editing(Workout)  
    case .saving(Workout)  
    case .viewing(Workout)  
}
```

Appearance as a function of state

```
func setupView(for state: ViewControllerState) {  
    navigationItem.title = state.title  
    navigationItem.leftBarButtonItems = state.leftBarButtonItems  
    navigationItem.rightBarButtonItems = state.rightBarButtonItems  
  
    UIView.animate(withDuration: 0.35) {  
        self.overlayView.alpha = state.overlayAlpha  
    }  
  
    if state.activityIndicatorActive {  
        activityIndicator.startAnimating()  
    } else {  
        activityIndicator.stopAnimating()  
    }  
}
```

Behavior as a function of state

```
protocol ViewControllerBehavior {  
    func didSelectCell(at indexPath: IndexPath)  
    func highlightCell(at indexPath: IndexPath)  
    func handleDragOnCell(at indexPath: IndexPath)  
    func numberOfRows(in section: Int) -> Int  
  
    var numberOfSections: Int  
    var supportsDragAndDropInteraction: Bool  
}  
  
extension ViewControllerState : ViewControllerBehavior {}
```

CaseIterable²

```
enum WorkoutApplicationLifecycle : CaseIterable {  
    case notLoggedIn  
    case workoutInProgress  
    case savingLastWorkout  
    case idle  
}
```

`WorkoutApplicationLifecycle.allCases`

`[notLoggedIn, workoutInProgress, savingLastWorkout, idle]`

² <https://oleb.net/blog/2018/06/enumerating-enum-cases/>

Custom CaseIterable

```
enum WorkoutApplicationLifecycle {  
    case notLoggedIn  
    case workoutInProgress(current : Workout)  
    case savingLastWorkout  
    case idle  
}  
  
extension WorkoutApplicationLifecycle : CaseIterable {  
    static var allCases: [WorkoutApplicationLifecycle] {  
        return [.notLoggedIn, .savingLastWorkout, .idle]  
            + WorkoutActivityType.allCases.map(.workoutInProgress)  
    }  
}
```

Enum raw values

```
enum DefaultKeys : String {  
    case loggedInUserId  
    case selectedActivityType  
}  
  
print(DefaultKeys.selectedActivityType.rawValue)  
  
selectedActivityType
```

Specific string raw values

```
enum DefaultKeys : String {  
    case loggedInUserId  
    case selectedActivityType = "SelectedActivityTypeDefaultKey"  
}  
  
print(DefaultKeys.selectedActivityType.rawValue)  
  
SelectedActivityTypeDefaultKey
```

Customize with RawRepresentable

```
enum ApplicationTheme {  
    case defaultLight  
    case darkMode  
    case blueberry  
    case eggplant  
    case mustard  
    case broccoli  
}  
  
struct Theme : Equatable {  
    let name : String  
    let background : UIColor  
    let cell : UIColor  
    let tint : UIColor  
    let foreground : UIColor  
    let statusBar : UIStatusBarStyle  
}
```

RawRepresentable

```
enum ApplicationTheme : RawRepresentable {  
    typealias RawValue = Theme  
  
    init?(rawValue: Theme) {  
        // ...  
    }  
  
    var rawValue: Theme {  
        // ...  
    }  
}  
  
print(ApplicationTheme.broccoli.rawValue.name)  
  
Broccoli
```

Associated theme values

```
enum ApplicationTheme {  
    case defaultLight(theme : Theme)  
    case darkMode(theme : Theme)  
    case blueberry(theme : Theme)  
    case eggplant(theme : Theme)  
    case mustard(theme : Theme)  
    case broccoli(theme : Theme)  
}
```

All the Themes

```
enum ApplicationTheme : CaseIterable {  
    case defaultLight  
    case darkMode  
    case blueberry  
    case eggplant  
    case mustard  
    case broccoli  
}
```

```
ApplicationTheme.allCases
```

```
[.defaultLight, .darkMode, .blueberry, .eggplant, .mustard, .broccoli]
```

Something Fun!



Confetti Moments

Colors, Images, 

ConfettiMoment

```
public enum ConfettiMoment {  
    // Color Confetti  
    case confettiMMF  
    case confettiMFP  
    case confetti🌈  
  
    // Emoji Confetti  
    case 🌮  
  
    // Custom Confetti  
    case customColors(colors : [UIColor], name : String)  
    case customImages(images : [UIImage], name : String)  
  
    func effectScene() -> SKScene  
}
```

4:03



8.01mi Run

Thursday, Jul 12 at 3:35 pm

Activity: Run



8.01

DISTANCE
(MI)



1:10:00

DURATION



8:44

AVG PACE
(MIN/MI)



1.5K

CALORIES

LIKES AND COMMENTS

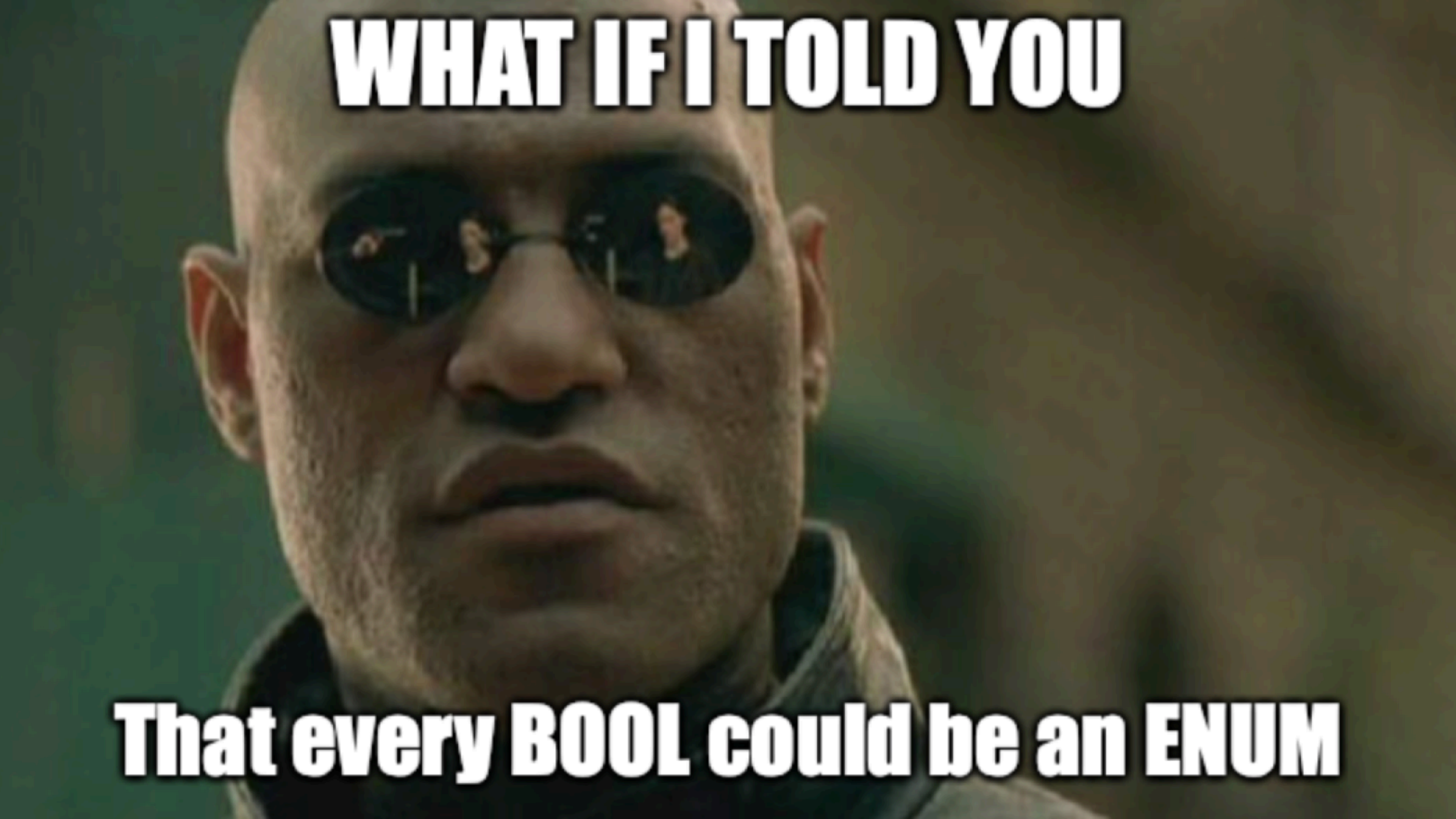
Show some love, write a comment, or share it!

Awesome run! [Congrats!!!](#)

POST

1 2 3 4 5 6 7 8 9 0
- / : ; () \$ & @ "
#+= . , ? ! ' ⌫
ABC space return



A close-up of Morpheus from the movie The Matrix, wearing his signature black sunglasses and a serious expression. The background is a blurred, dimly lit interior.

WHAT IF I TOLD YOU

That every BOOL could be an ENUM

Additional resources

- Enum Driven Table Views³⁴
- Pattern Matching in Swift⁵
- Maintaining State⁶

³ <https://www.natashatherobot.com/swift-enums-tableviews/>

⁴ <https://www.raywenderlich.com/5542-enum-driven-tableview-development>

⁵ <http://alisoftware.github.io/swift/pattern-matching/2016/03/27/pattern-matching-1/>

⁶ <https://developer.apple.com/documentation/swift/maintainingstateinyourapps>

Additional resources

- Enums and Optionals⁷
- Writing Self Documenting Swift Code⁸
- Code Encapsulation in Swift⁹
- Enumerations¹⁰

⁷ <http://khanlou.com/2018/04/enums-and-optionals/>

⁸ <https://www.swiftbysundell.com/posts/writing-self-documenting-swift-code>

⁹ <https://www.swiftbysundell.com/posts/code-encapsulation-in-swift>

¹⁰ <https://ericasadun.com/2015/05/26/swift-the-hall-of-the-dwarven-enumeration-king/#more-1529>

Thanks!

—



@conradstoll