13 Tips to Write Code Like a Swiftie

🦄 Mikaela Caron | Deep Dish Swift 2023

Mikaela Caron 🦄

- iOS Engineer at Lickability
- Freelancer
- Organizer of iOSDevHappyHour
- Content Creator

Lickability

Coding Like a Swiftie

Coding Like a Swiftie

Coding Like a Swiftie

- This is about Swift, not specific to SwiftUI or UIKit

Disclaimer

Tip #1: Proper Casing

- Properly capitalizing classes, structs, functions
- **UpperCamelCase**: protocols
- **lowerCamelCase**: variables,
- Do **NOT**, use snake_case, or

Tip #1: Proper Casing

- Properly capitalizing classes, structs, functions
- **UpperCamelCase**: protocols, enums, structs, classes
- **lowerCamelCase**: variables, parameters
- Do **NOT**, use snake_case, or kebab-case

Tip #1: Proper Casing

```
protocol Artist {
    var name: String { get set }
}
```

Tip #1: Proper Casing

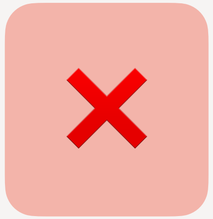
```
protocol Artist {
    var name: String { get set }
}

class PopArtist: Artist {
    var name: String
    init(name: String) {
        self.name = name
    }
}
```

Tip #1: Proper Casing

```
protocol Artist {
    var name: String { get set }
}

class PopArtist: Artist {
    var name: String
    init(name: String) {
        self.name = name
    }
}

let artist = PopArtist(name: "Taylor Swift")
```

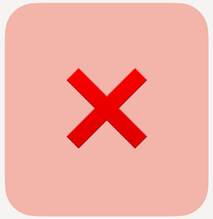
Tip #1: Proper Casing

```
protocol artist {
    var Name: String { get set }
}

class popartist: artist {
    var Name: String
    init(name: String) {
        self.Name = name
    }
}

let TaylorSwift = popartist(name: "Taylor Swift")
```



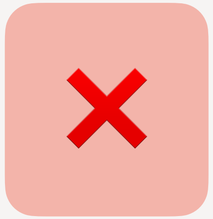
Tip #1: Proper Casing

```
protocol artist {
    var Name: String { get set }
}

class popartist: artist {
    var Name: String
    init(name: String) {
        self.Name = name
    }
}

let TaylorSwift = popartist(name: "Taylor Swift")
```



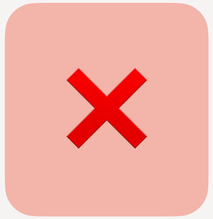
Tip #1: Proper Casing

```
protocol artist {
    var Name: String { get set }
}

class popartist: artist {
    var Name: String
    init(name: String) {
        self.Name = name
    }
}

let TaylorSwift = popartist(name: "Taylor Swift")
```



Tip #1: Proper Casing

```
protocol artist {
    var Name: String { get set }
}

class popartist: artist {
    var Name: String
    init(name: String) {
        self.Name = name
    }
}

let TaylorSwift = popartist(name: "Taylor Swift")
```



Tip #2: Naming

- Use US English spelling
- **Functions** prefer names that make the call site read like natural language
- **Booleans** should read as assertions about the receiver
- **Protocols** that describe capability should be named with suffixes `-able`, `ible`, or `ing`
- All else should be read as nouns

Tip #2: Naming

- Use US English

```
enum AlbumColor {
    case red, blue, black
}

func makeFavorite(album: String) -> String {
    return "⭐ \(album)"
}
```

Tip #2: Naming

- Functions: Prefer names that make the call site read like natural language

```
func composeSong(lyrics: String) -> String {
    return "🎵 \(lyrics) 🎵"
}
```

Tip #2: Naming

- Booleans should read as assertions about the receiver

```
let areYouReady: Bool

let areWeGettingBackTogether: Bool

func isEndGame() -> Bool
```

Tip #2: Naming

- Protocols that describe capability should be named with suffixes `able`, `ible`, or `ing`

```
protocol Calmable { }

protocol Lovable { }

protocol Equatable { }
```

Tip #2: Naming

- All other types, properties, variables, constants should be read as nouns

```
let artist = PopArtist(name: "Taylor Swift”)

let lyric = "You belong with me"
```

Tip #3: Argument Labels

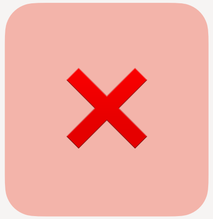
- Syntax that adds a word to make the function easier to read

Tip #3: Argument Labels

```
func isThereAnI(in word: String) -> Bool {
    word.contains("*i*")
}

isThereAnI(in: “team")
```

Tip #3: Argument Labels

```
func isThereAnI(word: String) -> Bool {
    word.contains("*i*")
}

isThereAnI(word: "team")
```



Tip #4: Type Alias

- Introduces a named alias of an existing type.

Tip #4: Type Alias

- Introduces a named alias of an existing type.

Tip #4: Type Alias

```
typealias Discography = OrderedDictionary<String, [Song]>

func makeTaylorDiscography(_ discography: Discography) {
}
```

Tip #5: Documentation

- Write a documentation comment for every declaration

```
/// Creates a string with surrounding music notes.
/// - Parameter lyric: The song lyric.
/// - Returns: The lyric with surrounding music notes.
func makeLyric(_ lyric: String) -> String {
    return "🎵 \(lyric) 🎵"
}
```

Tip #5: Documentation

- Quick help

Summary

Creates a string with surrounding music notes.

Declaration

```
func makeLyric(_ lyric: String) -> String
```

Parameters

`lyric` The song lyric.

Returns

The lyric with surrounding music notes.

Declared In

Examples.playground

Tip #6: Access Control

- What objects can access other objects
- Default to `private`, and get-only

Tip #6: Access Control

```
class TaylorSwift {

    private(set) var lyrics: String

    init(lyrics: String) {
        self.lyrics = lyrics
    }
}
```

Tip #6: Access Control

```
let taylor = TaylorSwift(lyrics: "I just wanna stay in that
lavender haze”)

print(taylor.lyrics)
```

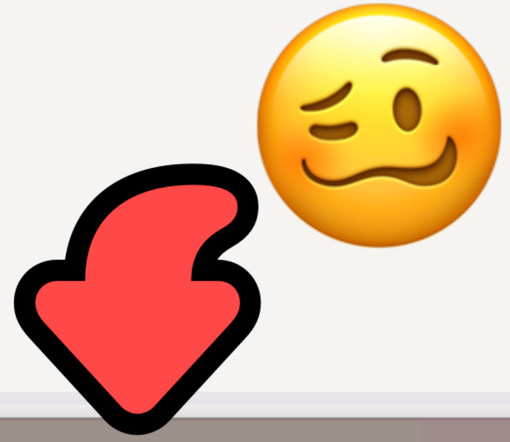
```
I just wanna stay in that lavender haze
```

Tip #6: Access Control

```
let taylor = TaylorSwift(lyrics: "I just wanna stay in that lavender haze”)

print(taylor.lyrics)
```

✅

```
taylor.lyrics = "I'm the problem, it's me"
```



Cannot assign to property: 'lyrics' setter is inaccessible

Tip #7: Code Structure

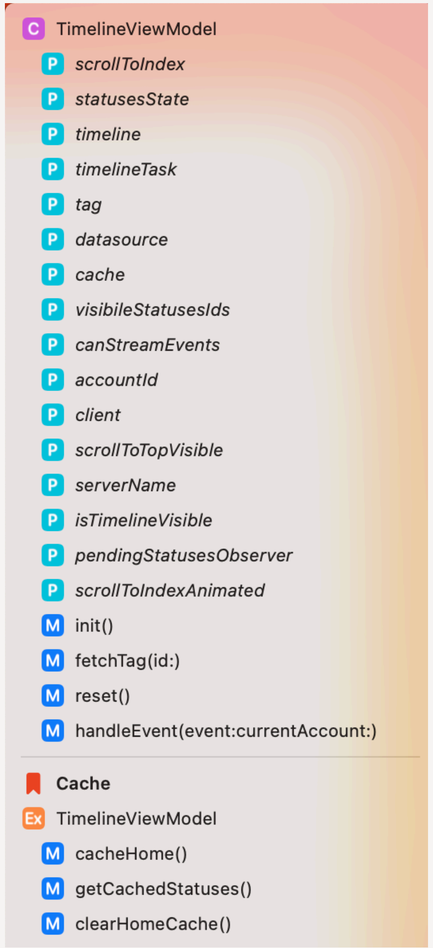
- How a file is organized

Tip #7: Code Structure

- Import statements
- Protocols
- Main type declaration of the file
- Properties
- Methods
- Extension protocol conformances

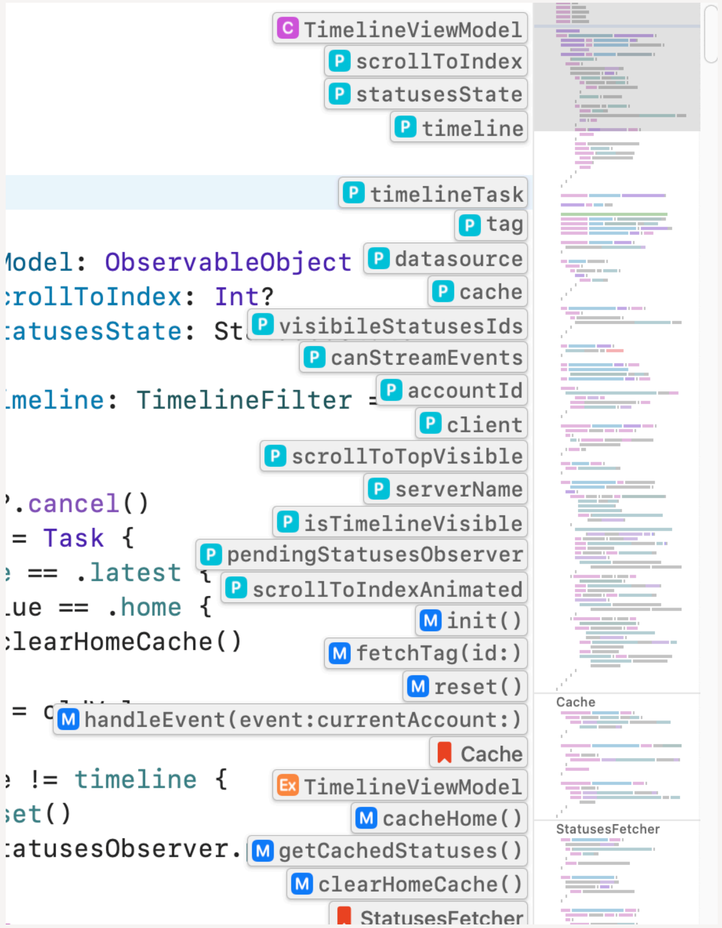
Tip #7: Code Structure

Jump Bar

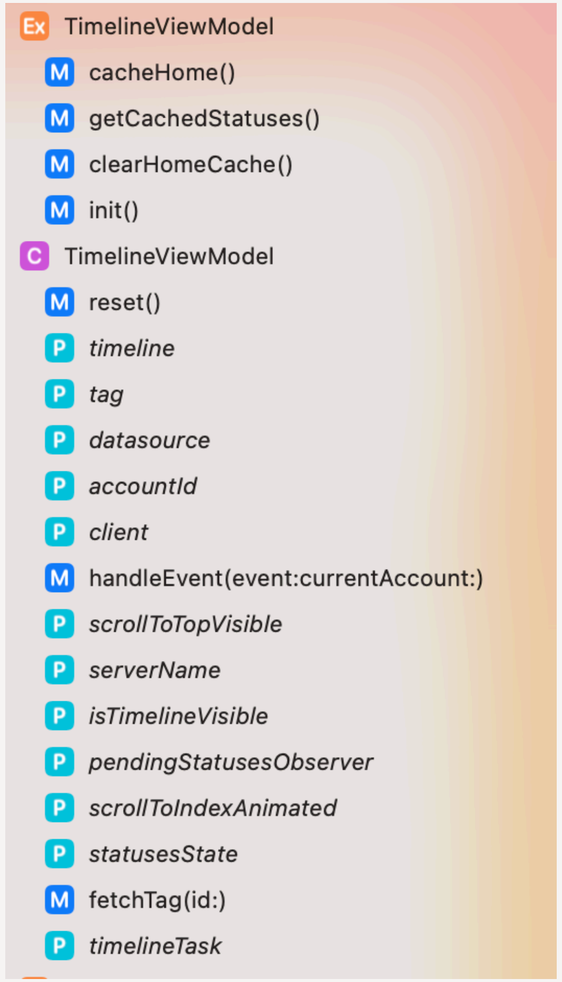


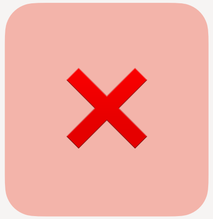
Minimap

Minimap + Cmd



Tip #7: Code Structure





Tip #8: Enumerations (Enums)

- Defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code.

Tip #8: Enumerations (Enums)

- Defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code.

```
enum Album {
    case reputation
    case midnights
    case evermore
}
```

Tip #8: Enumerations (Enums)

- Raw values and associated values

Tip #8: Enumerations (Enums)

- Raw values

```
enum Album: Int {
    case reputation = 2017
    case midnights = 2022
    case evermore = 2020
}
```

Tip #8: Enumerations (Enums)

- Associated values

```
enum Album {
    case reputation(views: Int)
    case midnights(String)
    case evermore
}
```

Tip #8: Enumerations (Enums)

```
enum SongRequest {
    case mySongs
    case requestSong(id: String)

    var baseURL: URL {
        return API.url
    }
}
```

Tip #8: Enumerations (Enums)

```
enum SongRequest {
    // ...
    var path: String {
        switch self {
        case .mySongs:
            return "/mysongs"
        case let .requestSong(id: id)
            return "/request/\(id)"
        }
    }
}
```

Tip #8: Enumerations (Enums)

```
enum SongRequest {
    //...
    var httpMethod: HTTPMethod {
        switch self {
        case .mySongs:
            return .get
        case .requestSong:
            return .post
        }
    }
}
```

Tip #8: Enumerations (Enums)

```
networkController.send(request: SongRequest.requestSong(id:
song.id))
```

Tip #9: Formatting Large Numbers

- Improve readability with underscores

Tip #9: Formatting Large Numbers

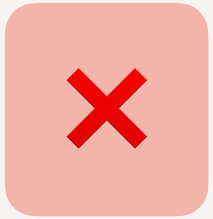
```
struct Statistics {
    let views: Int
}

let shakeItOffStats = Statistics(views: 3_278_311_534)

let blankSpaceStats = Statistics(views: 3_144_540_109)

let badBloodStats = Statistics(views: 1_547_508_835)
```

Tip #10: Blank Space

- Vertical and horizontal spacing

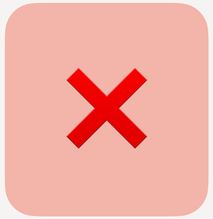
Tip #10: Blank Space

```
struct Lyric {


    var intro: String = ""
    var verse: String = ""
    var chorus: String


    var bridge: String



    func makeSong() {
    }
}
```



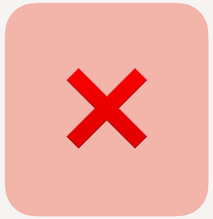
Tip #10: Blank Space

```
struct Lyric{
  var  intro:  String=""
    var verse:  String  =""
 var chorus:String  =  ""
    var bridge: String
    var   isComplete:  Bool

    func makeSong ()    {
    }
}
```



Tip #10: Blank Space

```
struct Lyric{


    var  intro:  String   =""
    var verse:  String  =  ""



    var chorus: String =""
    var bridge: String  =""
  var isComplete:  Bool

        func makeSong ()    {
        }
}
```



Tip #10: Blank Space

```
struct Lyric {
    var intro: String = ""
    var verse: String = ""
    var chorus: String = ""
    var bridge: String
    var isComplete: Bool

    func makeSong() {
    }
}
```

Tip #10: Blank Space

```
struct Lyric {
    var intro: String = ""
    var verse: String = ""
    var chorus: String = ""
    var bridge: String
    var isComplete: Bool

    func makeSong() {
    }
}
```

Tip #10: Blank Space

```
struct Lyric {
    var intro: String = ""
    var verse: String = ""
    var chorus: String = ""
    var bridge: String
    var isComplete: Bool

    func makeSong() {
    }
}
```

Tip #10: Blank Space

```
struct Lyric {
    var intro: String = ""
    var verse: String = ""
    var chorus: String = ""
    var bridge: String
    var isComplete: Bool

    func makeSong() {
    }
}
```

Tip #10: Blank Space

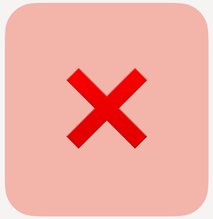
```
struct Lyric {
    var intro: String = ""
    var verse: String = ""
    var chorus: String = ""
    var bridge: String
    var isComplete: Bool

    func makeSong() {
    }
}
```

Tip #11: Don’t Use Prefixes

- Avoid general prefixes before types

Tip #11: Don’t Use Prefixes

- Avoid general prefixes before types



```
struct TSLyric {
    var intro: String
    var verse: String
    var chorus: String
    var bridge: String
}
```

Tip #11: Don’t Use Prefixes

- Avoid general prefixes before types

```
struct Lyric {
    var intro: String
    var verse: String
    var chorus: String
    var bridge: String
}
```

Tip #12: Namespacing

- Group related items

Tip #12: Namespacing

```
enum Design {
    struct Grid {
        static let tiny: CGFloat = 4
        static let small: CGFloat = 8
        static let medium: CGFloat = 16
        static let large: CGFloat = 24
    }
}
```

Tip #12: Namespacing

```
enum Design {
    struct Grid {
        static let tiny: CGFloat = 4
        static let small: CGFloat = 8
        static let medium: CGFloat = 16
        static let large: CGFloat = 24
    }
}

Design.Grid.medium
```


BAR

Tip #13: SwiftLint

- A tool to enforce Swift style and conventions
- Open-source project by Realm

Tip #13: SwiftLint

```
16  struct Sign {
17      let startTime: Date
18
19
20  _
21      let endtime: Date
22  }
23
```

⚠ Vertical Whitespace Violation: Limit vertical whitespace to a single empty line; currently 3 (vertical_whitespace)

13 Tips to Write Code Like a Swiftie

1. Proper Casing
2. Naming
3. Argument Labels
4. Type Alias
5. Documentation
6. Access Control
7. Code Structure
8. Enumerations (Enums)
9. Formatting Large Numbers
10. Blank Space
11. Don’t Use Prefixes
12. Namespacing
13. SwiftLint

13 Tips to Write Code Like a Swiftie

Questions?

@mikaela__caron

@mikaelacaron

@mikaelacaron@swiftdevs.space