State

```
@property (nonatomic, assign) BOOL loading;

- (void)fetchBananas {
  // Fetch some bananas from the network.
}
```

```
@property (nonatomic, assign) BOOL loading;

- (void)fetchBananas {
  self.loading = @YES;
  [[APIClient sharedClient] getBananasWithCompletion:^{
    self.loading = @NO;
  }];
}
```

```
@property (nonatomic, assign) BOOL loading;

- (void)fetchBananas {
  self.loading = @YES;
  [[APIClient sharedClient] getBananasWithCompletion:^{
    self.loading = @NO;
  }];
}

- (void)fetchBushels {
  self.loading = @YES;
  [[APIClient sharedClient] getBushelsWithCompletion:^{
    self.loading = @NO;
  }];
}
```

```objc
@property (nonatomic, assign) BOOL loading;

- (void)fetchBananas {
  self.loading = @YES;
  [[APIClient sharedClient] getBananasWithCompletion:^{
    self.loading = @NO;
  }];
}

- (void)fetchBushels {
  self.loading = @YES;
  [[APIClient sharedClient] getBushelsWithCompletion:^{
    self.loading = @NO;
  }];
}
```

self.loading = ???

```
@property (nonatomic, assign) BOOL bananasLoading;
@property (nonatomic, assign) BOOL bushelsLoading;

- (void)fetchBananas {
  self.bananasLoading = @YES;
  [[APIClient sharedClient] getBananasWithCompletion:^{
    self.bananasLoading = @NO;
  }];
}

- (void)fetchBushels {
  self.bushelsLoading = @YES;
  [[APIClient sharedClient] getBushelsWithCompletion:^{
    self.bushelsLoading = @NO;
  }];
}
```

```
@property (nonatomic, assign) BOOL bananasLoading;
@property (nonatomic, assign) BOOL bushelsLoading;
@property (nonatomic, assign) BOOL monkeysLoading;

- (void)fetchBananas {
  ...
}

- (void)fetchBushels {
  ...
}

- (void)fetchMonkeys {
  ...
}
```

As we get more complex, we add more and more state in

State is exponential

1 boolean = 2 states
2 booleans = 4 states
3 booleans = 8 states
4 booleans = 16 states
??? booleans = ?!?! states

State

State is never simple.

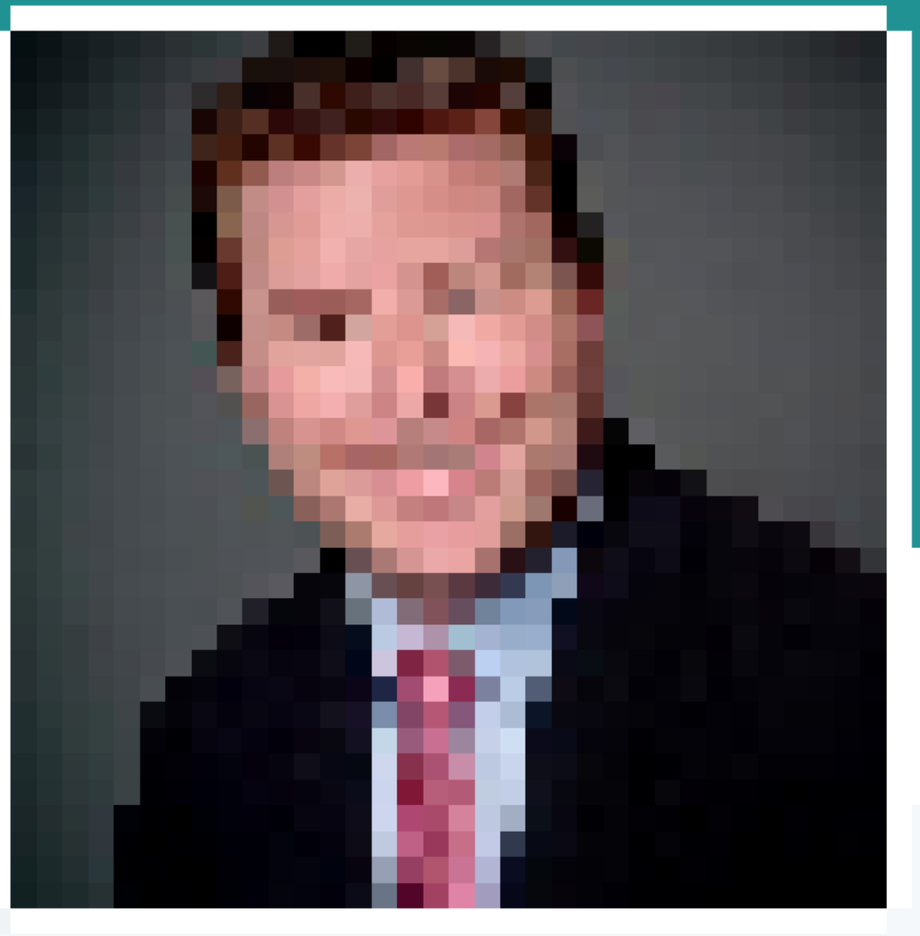
– Rich Hickey

Staters Gonna

State

Eli Perkins

@eliperkins



[redacted]

@eliperkins

TWEETS 47 | FOLLOWING 69 | FOLLOWERS 37

Tweets | Tweets & replies

[redacted] @eliperkins · Mar 4

@eliperkins

most places on the internet

@_eliperkins

on Twitter

venmo

State

Simple

vs.

Easy

Simple

Easy

```
[[[[self tableView] frame] size] height]
```

```
sum' :: (Num a) => [a] -> a
sum' xs = foldl (\acc x -> acc + x) 0 xs
```

```
(defn inc-each [coll] (into (empty coll) (map inc coll)))
```

Easy, but not Simple

Easy, but not Simple

```
class HelperClass {
    class func decorateMonkey(
      monkey: Monkey,
      shouldTellServer: Bool,
      bananasToEatAfter: [Banana]
      ) -> Void {
        ...
      }
}
```

Simple

vs.

Easy

Immutability
Source of Truth
Value vs. Reference types
Getting `funcy`

Immutability

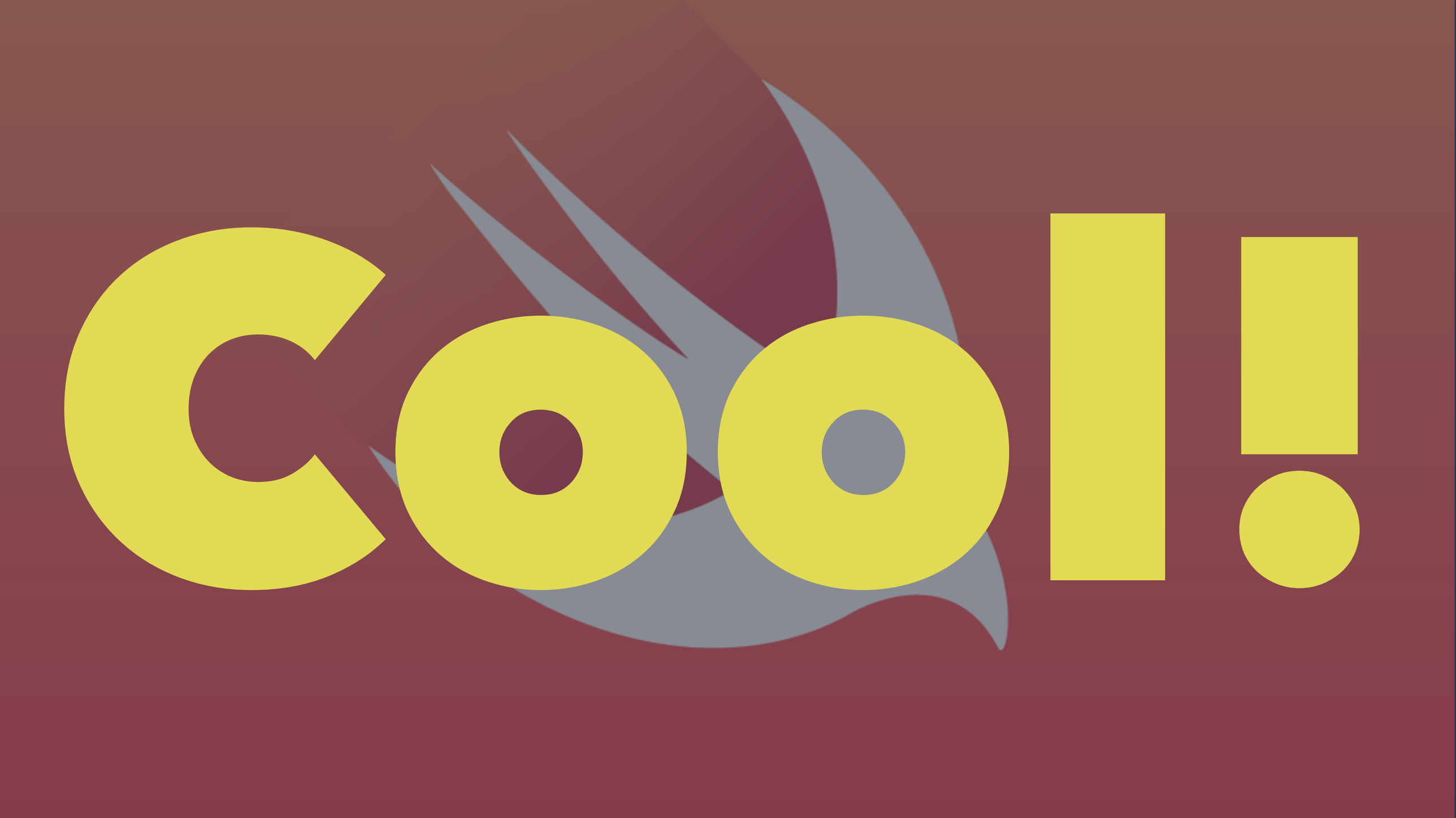
Source of Truth

Value vs. Reference types

Getting `funcy`

"Mutable objects complect time and values."

– Lee Byron


Cool!

```
let bananas = [Banana(peeled: true), Banana(peeled: false)]
let numberOfBananas = bananas.count
numberOfBananas = 4  // cannot assign to 'let' value
```

Immutabilty helps us reason about code

```
struct HTTPClient {
    let accessToken: AccessToken

    init(accessToken: AccessToken) {
        self.accessToken = accessToken
    }
}
```

"What is the impact of the change I want to make?"

If you can't reason about the software, you make it harder to make these decisions.

– Rich Hickey

Immutability

Source of Truth

Value vs. Reference types

Getting `funcy`


A
B
C
D

$$
\begin{aligned}
&\longrightarrow A \longrightarrow B \longrightarrow C \longrightarrow D \\
&\longrightarrow A \longrightarrow C \\
&\longrightarrow C \longrightarrow B \\
&\longrightarrow D \longrightarrow B
\end{aligned}
$$

$$\longrightarrow\ [Artist] \longrightarrow\ Artist :: [Album] \longrightarrow\ Album :: [Song] \longrightarrow\ Song$$

$$\longrightarrow\ [Artist] \longrightarrow\ Artist :: [Album] \longrightarrow\ Album :: [Song] \longrightarrow\ Song$$

Truth as State

setState()

View Models

```
struct SongViewModel {
  let song: Song
}

struct AlbumViewModel {
  let songs: [SongViewModel]
  let imageURL: NSURL

  var image: UIImage {
    // Load our image from the network or cache
  }
}
```

```
struct SongListViewModel {
  let songs: [SongCellViewModel]
}

struct SongCellViewModel {
  let songViewModel: SongViewModel
  let albumViewModel: AlbumViewModel

  var image: UIImage {
    return albumViewModel.image
  }
}
```

$$
\begin{array}{lcl}
\longrightarrow & [ArtistViewModel] & \longrightarrow \quad ArtistViewModel :: [AlbumViewModel] \\
\longrightarrow & ArtistViewModel :: [AlbumViewModel] & \longrightarrow \quad AlbumViewModel :: [SongCellViewModel] \\
\longrightarrow & SongCellViewModel :: (AlbumViewModel, SongViewModel) & \longrightarrow \quad SongViewModel
\end{array}
$$

Immutability

Source of Truth

Value vs. Reference types

Getting funcy

Value Types

Values

```
struct Banana {
    var peeled: Bool = false
}

var banana = Banana(peeled: true) // {peeled true}
var bananaReference = banana      // {peeled true}
bananaReference.peeled = false    // {peeled false}
banana                            // {peeled true}
bananaReference                   // {peeled false}
```

Values as data

Objects are not data

Data is data

Values are data

"Let data be data."

– Rich Hickey


"Let data be data."
– Rich Hickey (again)
Clojure
Mission Bay

Values Reduce State

Pistachio[1]

```
struct Origin {
  var city: String

  init(city: String = "") {
    self.city = city
  }
}

struct Person {
  var name: String
  var origin: Origin

  init(name: String = "", origin: Origin = Origin()) {
    self.name = name
    self.origin = origin
  }
}
```

---

[1] from https://github.com/felixjendrusch/Pistachio && https://github.com/robb/Monocle

Pistachio[1]

```
struct OriginLenses {
  static let city = Lens(get: { $0.city }, set: { (inout origin: Origin, city) in
    origin.city = city
  })
}

struct PersonLenses {
  static let name = Lens(get: { $0.name }, set: { (inout person: Person, name) in
    person.name = name
  })

  static let origin = Lens(get: { $0.origin }, set: { (inout person: Person, origin) in
    person.origin = origin
  })
}

let felix = Person(name: "Felix", origin: Origin(city: "Berlin"))
let robb = set(PersonLenses.name, person, "Robb")
get(PersonLenses.name, robb) // == "Robb"
felix.name // == "Felix"
```

---

[1] from https://github.com/felixjendrusch/Pistachio && https://github.com/robb/Monocle

Immutability
Source of Truth
Value vs. Reference types

Getting `funcy`

Procedural

```
// Need a class here. With a struct:
// `immutable value of type '[Banana]' only has mutating members named 'append'`
class Monkey {
    var stomach: [Banana] = []
}
```

Procedural

```
// Need a class here. With a struct:
// `immutable value of type '[Banana]' only has mutating members named 'append'`
class Monkey {
    var stomach: [Banana] = []
}

let monkey = Monkey()
func eat() {
  monkey.stomach.append(Banana(peeled: true))
}
eat()
monkey.stomach    // == [{peeled: true}]
```

Object-Oriented

```
class Monkey {
    private var stomach: [Banana] = []
    func eat(banana: Banana) {
        stomach.append(banana)
    }
}

let monkey = Monkey()
monkey.eat(Banana(peeled: true))
```

Object-Oriented

```
class Monkey {
    var stomach: [Banana] = []
    func eat(banana: Banana) {
        stomach.append(banana)
    }
}

let monkey = Monkey()
monkey.eat(Banana(peeled: true))  // Happy 🐵
```

- Procedural: Mutable + Separate Data & Code
- Object-Oriented: Mutable + Combined Data & Code

Functional

```
struct Monkey {
    let stomach: [Banana]
}

func eat(monkey: Monkey, banana: Banana) -> Monkey {
    let stomach = monkey.stomach + [banana]
    return Monkey(stomach: stomach) // New 🐒, new stomach
}
```

Functional

```
struct Monkey {
    let stomach: [Banana]
}

func eat(monkey: Monkey, banana: Banana) -> Monkey {
    let stomach = monkey.stomach + [banana]
    return Monkey(stomach: stomach) // New 🐒, new stomach
}

let monkey = Monkey(stomach: [])
let fedMonkey = eat(monkey: monkey, banana: Banana(peeled: true))

monkey == fedMonkey // == false

monkey.stomach // == []
fedMonkey.stomach // == [{peeled: true}]
```

```
let troop = [
    Monkey(stomach: []),
    Monkey(stomach: []),
    Monkey(stomach: [])
]
```

```
let troop = [
    Monkey(stomach: []),
    Monkey(stomach: []),
    Monkey(stomach: [])
]

let fedTroop = troop.map {
    eat($0, Banana(peeled: true))
}

troop          // == Array of unfed 🙊
fedTroop       // == Array of monkeys with one banana in their stomach
```

```
let troop = [
    Monkey(stomach: []),
    Monkey(stomach: []),
    Monkey(stomach: [])
]

let fedTroop = troop.map {
    eat($0, Banana(peeled: true))
}

let bananaCount = fedTroop.map { return $0.stomach.count }.reduce(0, combine: +)
bananaCount // == 3

// `map` a second time
let superfedTroop = troop.map {
    eat(eat($0, Banana(peeled: true)), Banana(peeled: true))
}

let bananaRecount = fedTroop.map { return $0.stomach.count }.reduce(0, combine: +)
bananaRecount // == 3

let superfedCount = superfedTroop.map { return $0.stomach.count }.reduce(0, combine: +)
superfedCount // == 6
```

- Procedural: Mutable + Separate Data & Code
- Object-Oriented: Mutable + Combined Data & Code
- Functional: Immutable + Separate Data & Code

- Procedural: Mutable + Separate Data & Code
- Object-Oriented: Mutable + Combined Data & Code
- Functional: Immutable + Separate Data & Code
- FauxO: Immutable + Combined Data & Code

Functional

```
struct Monkey {
    let stomach: [Banana]
}

func eat(monkey: Monkey, banana: Banana) -> Monkey {
    let stomach = monkey.stomach + [banana]
    return Monkey(stomach: stomach) // New 🐒, new stomach
}
```

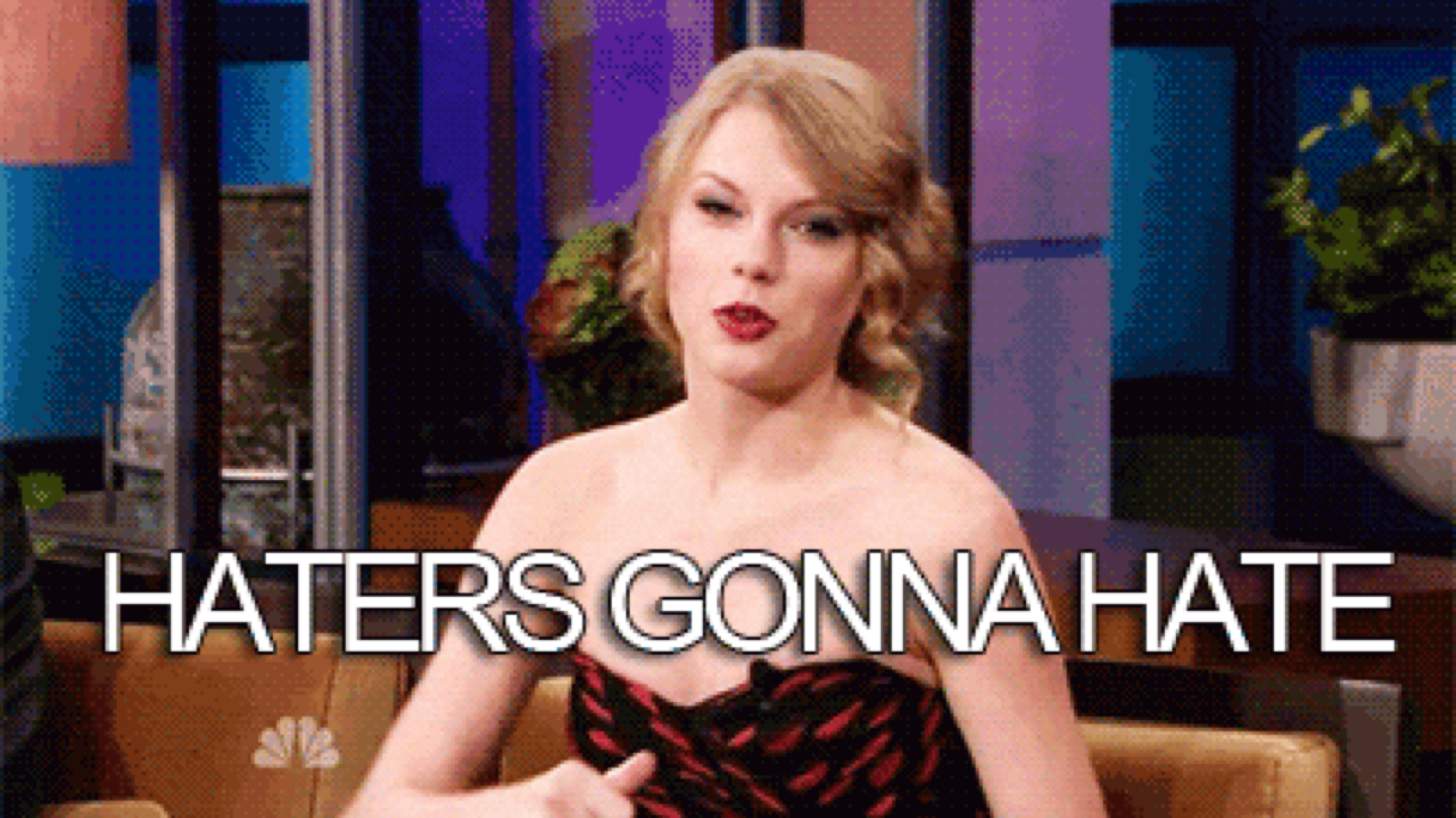
FauxO

```
struct Monkey {
    let stomach: [Banana]

    func eat(banana: Banana) -> Monkey {
        return Monkey(stomach: stomach + [banana]) // New 🐒, new stomach
    }
}
```

- Procedural: Mutable + Separate Data & Code
- Object-Oriented: Mutable + Combined Data & Code
- Functional: Immutable + Separate Data & Code
- FauxO: Immutable + Combined Data & Code

Immutability
Source of Truth
Value vs. Reference types
Getting `funcy`


HATERS GONNA HATE

- Rich Hickey - "Simple Made Easy"
- Gary Bernhardt - "Boundaries"
- Lee Byron - "Immutable Data & React"
- Learn You as Haskell
- Justin Spahr-Summers - "Enemy of the State"
- Andy Matuschak - WWDC 2014: Session 229 [2]

---

[2] ...or pretty much anything Andy says.

Thanks

@_eliperkins