I’m a Coder

Liz Marley

cat sketches by @Adana

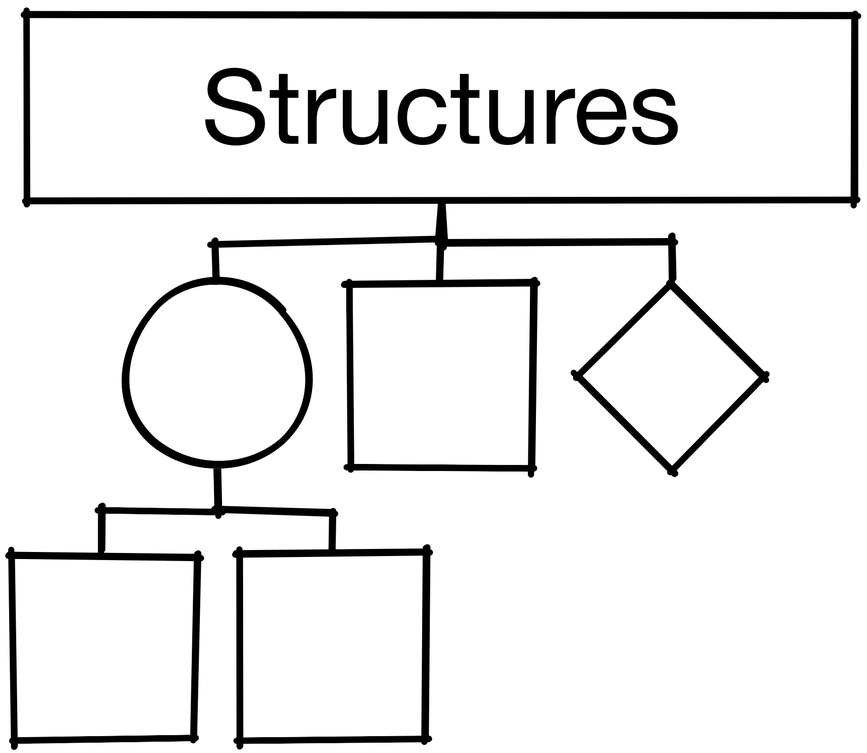
If you can't read this line, please choose a closer seat.

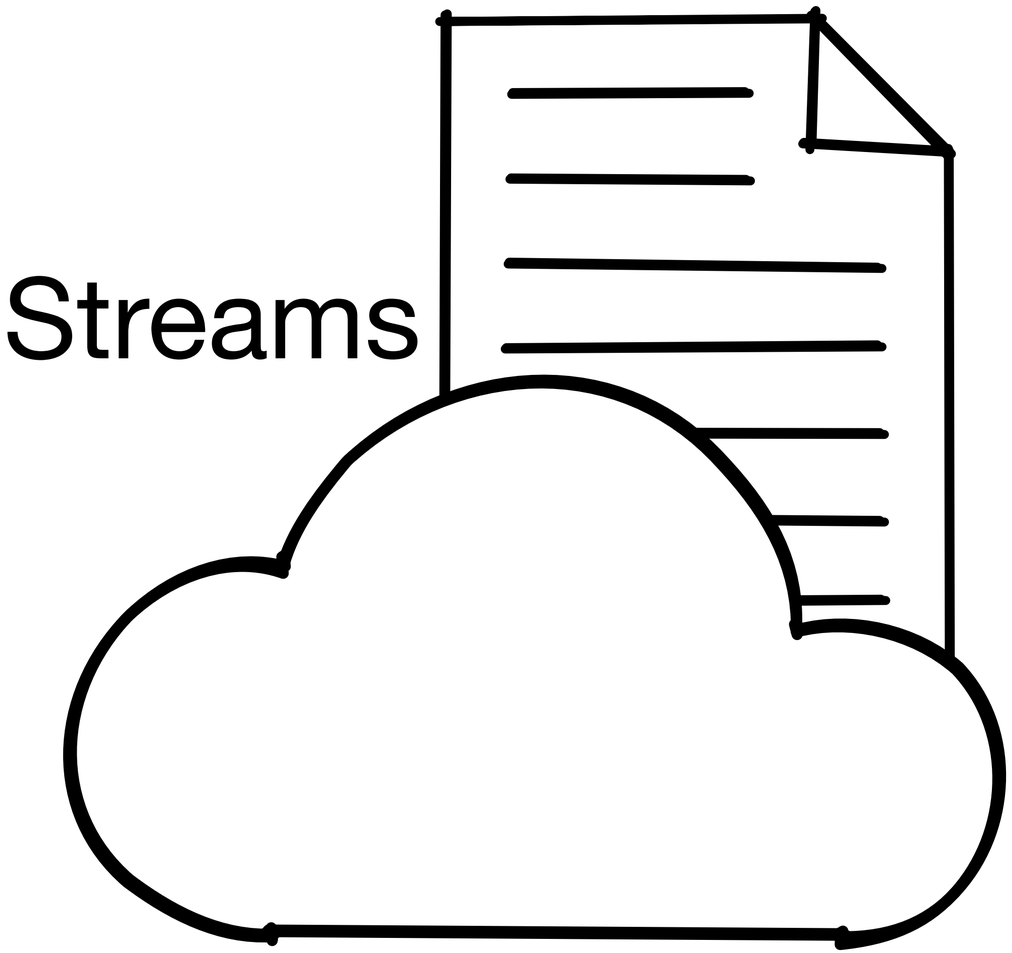
Codable

& NSCoding

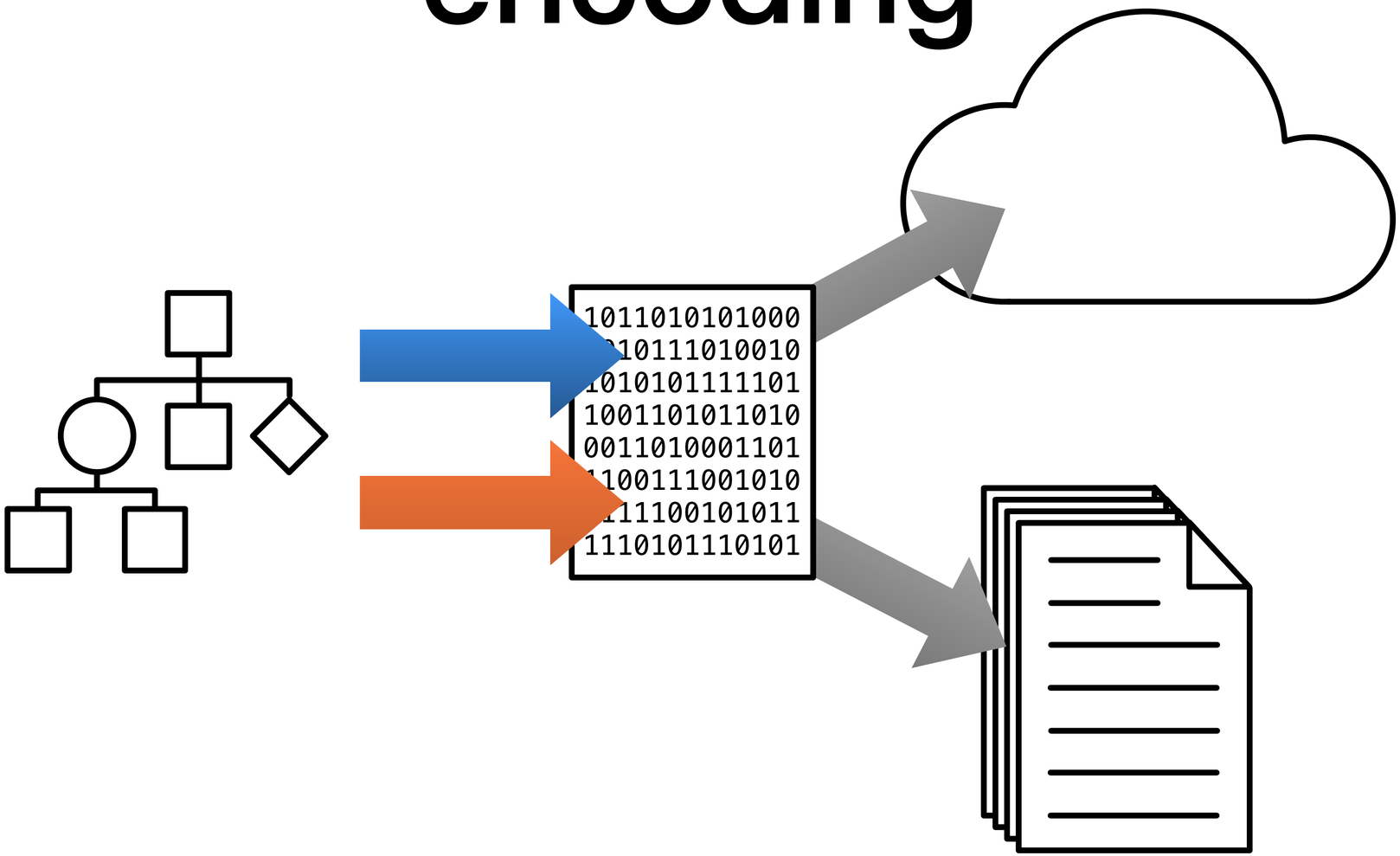
If you can't read this line, please choose a closer seat.

https://gist.github.com/emarley

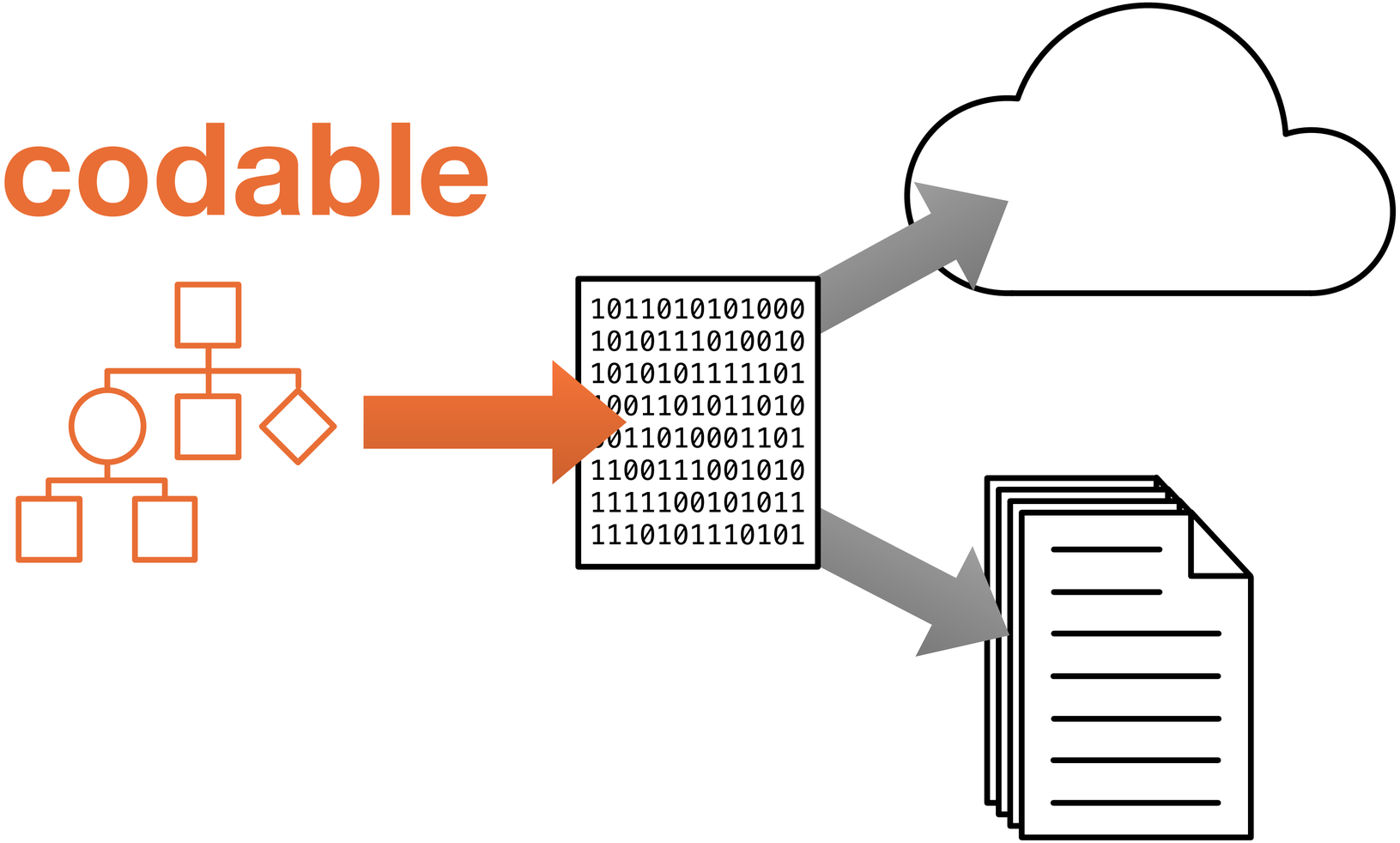

Structures


Streams

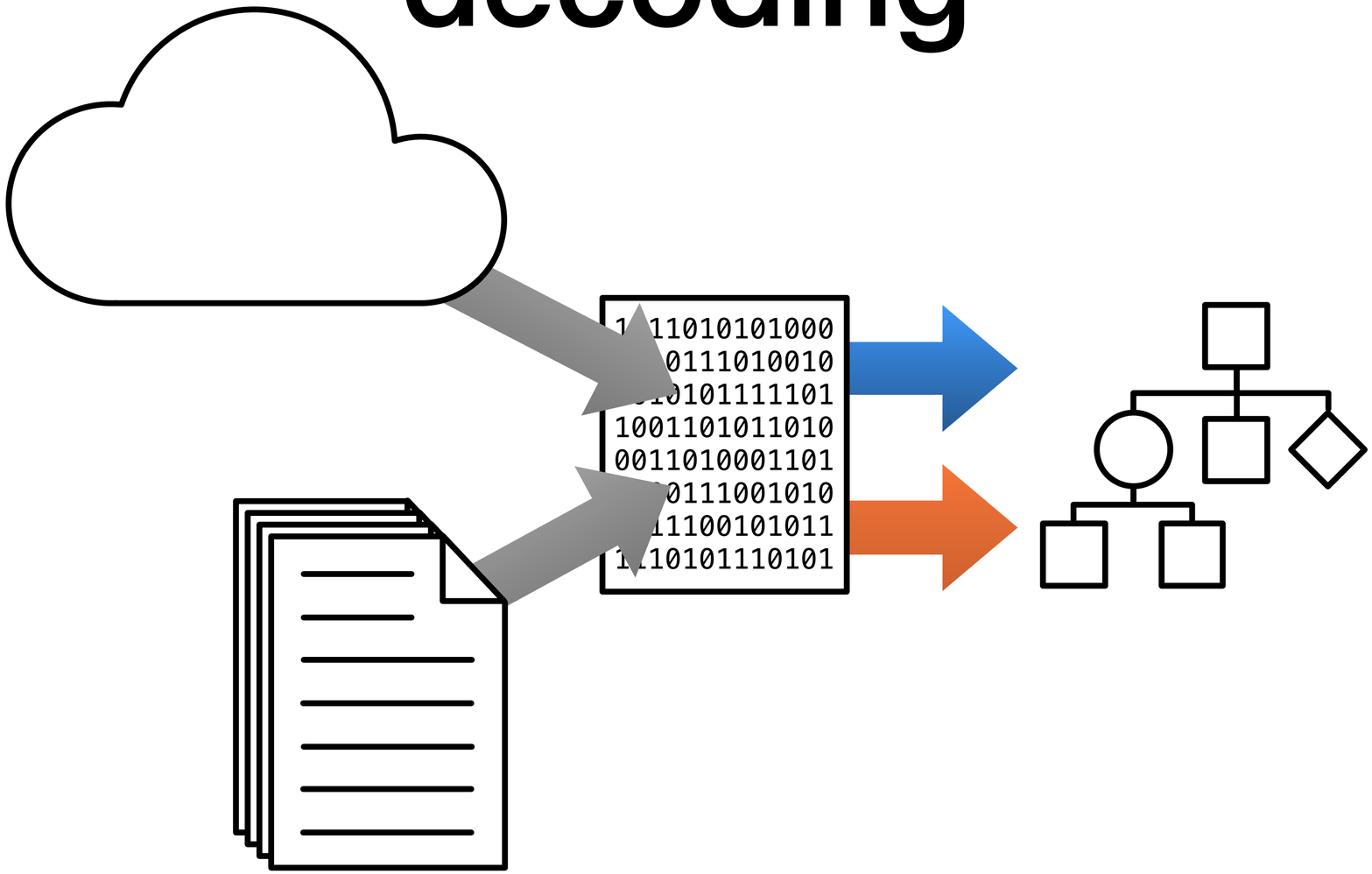
encoding


1011010101000
0110111010010
1010101111101
1001101011010
0011010001101
1100111001010
1111100101011
1110101110101

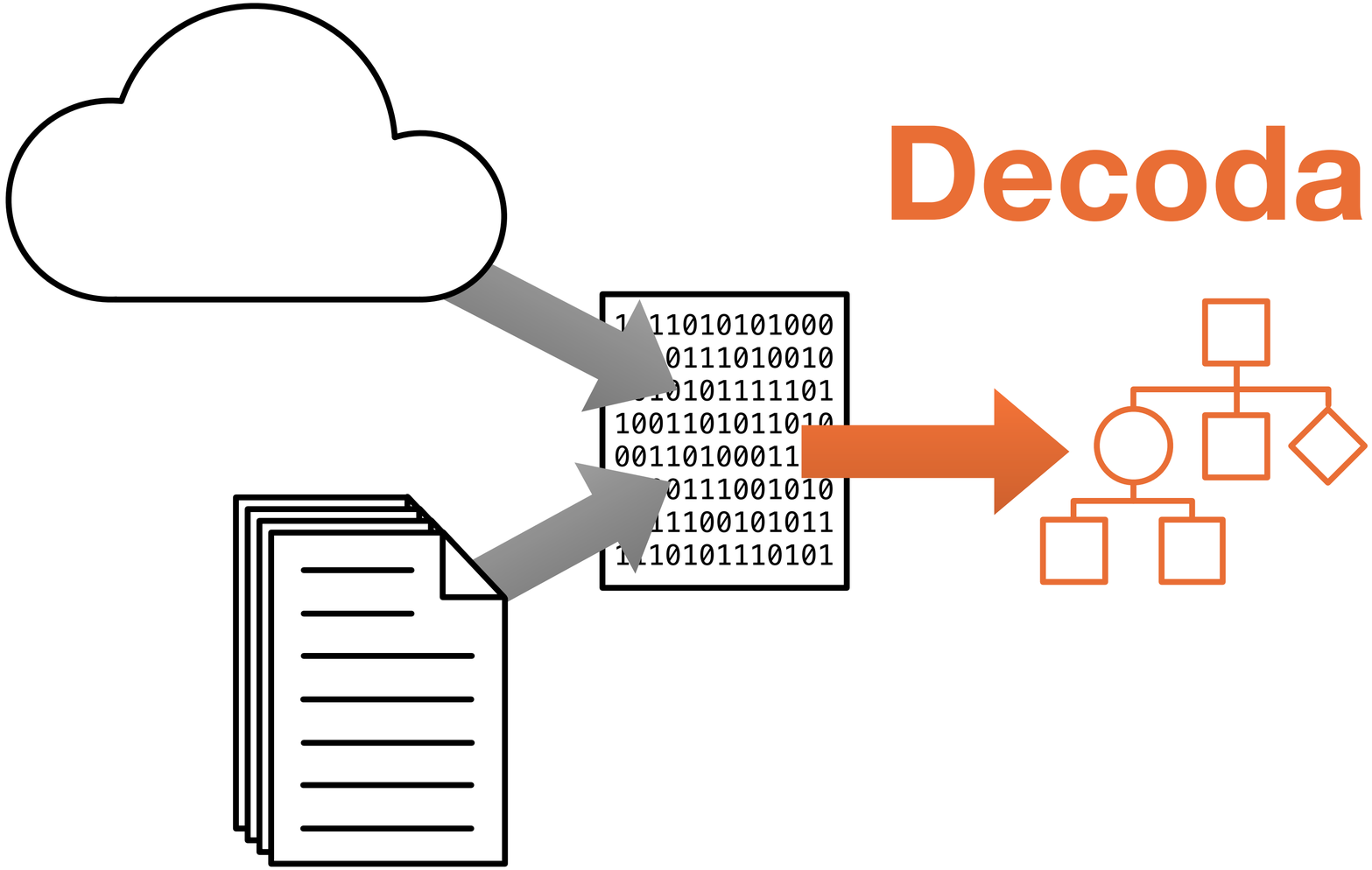
Encodable


1011010101000
1010111010010
1010101111101
1001101011010
1011010001101
1100111001010
1111100101011
1110101110101

decoding


1011010101000
0111010010
0101111101
1001101011010
0011010001101
0111001010
1100101011
1010101110101

Decodable


1011010101000
0111010010
0101111101
1001101011010
00110100011
0111001010
1100101011
1010101110101

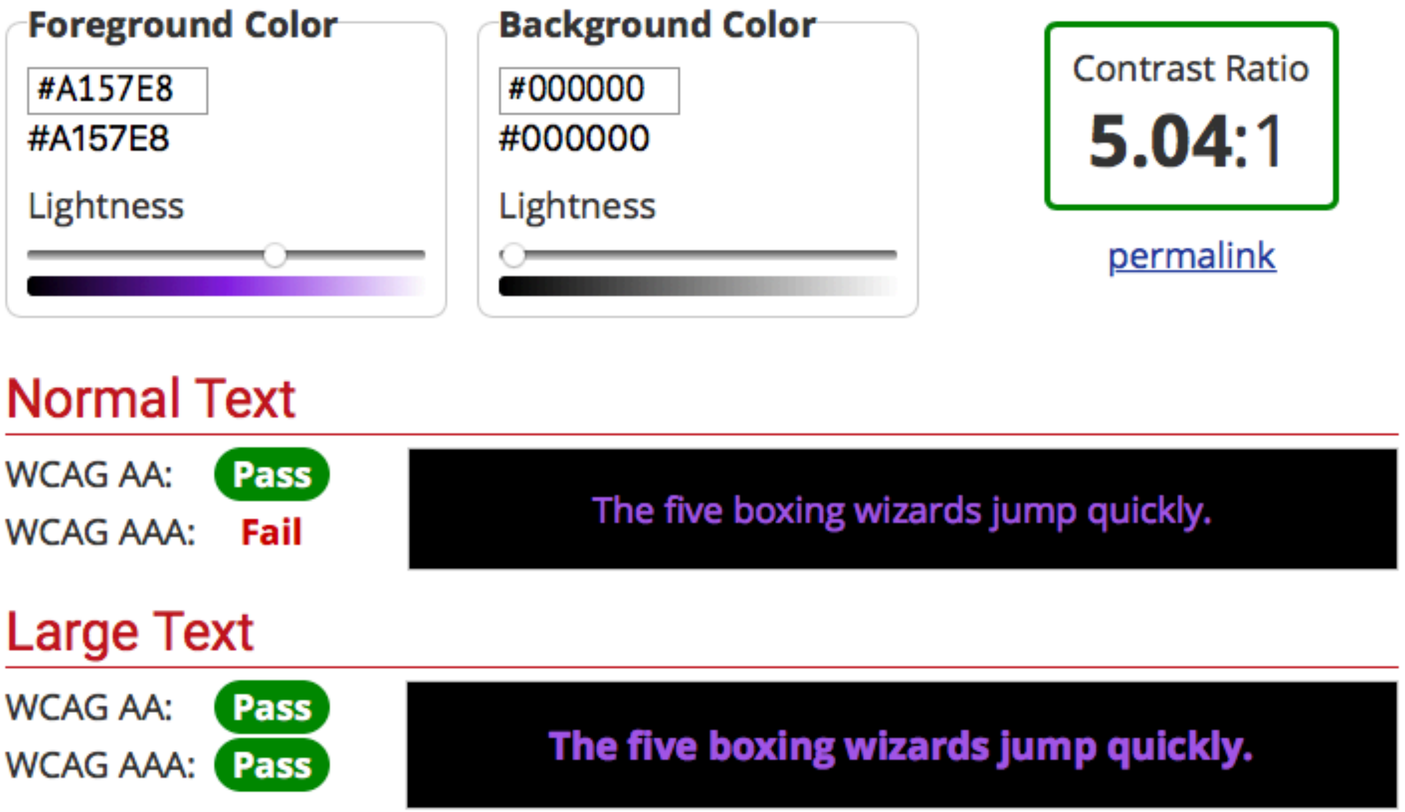
Is this enough contrast?

Is this enough contrast?

Is this enough contrast?

Is this enough contrast?

https://webaim.org/resources/contrastchecker/?fcolor=A157E8&bcolor=000000



https://webaim.org/resources/contrastchecker/?fcolor=A157E8&bcolor=000000&api


```
{
    "ratio": 5.04,
    "AA": "pass",
    "AALarge": "pass",
    "AAA": "fail",
    "AAALarge": "pass"
}
```

https://webaim.org/resources/contrastchecker/?fcolor=A157E8&bcolor=000000&api

```
struct WebColorContrastResponse {
  let ratio: CGFloat
  let AA: String
  let AALarge: String
  let AAA: String
  let AAALarge: String
}
```

https://webaim.org/resources/contrastchecker/?fcolor=A157E8&bcolor=000000&api

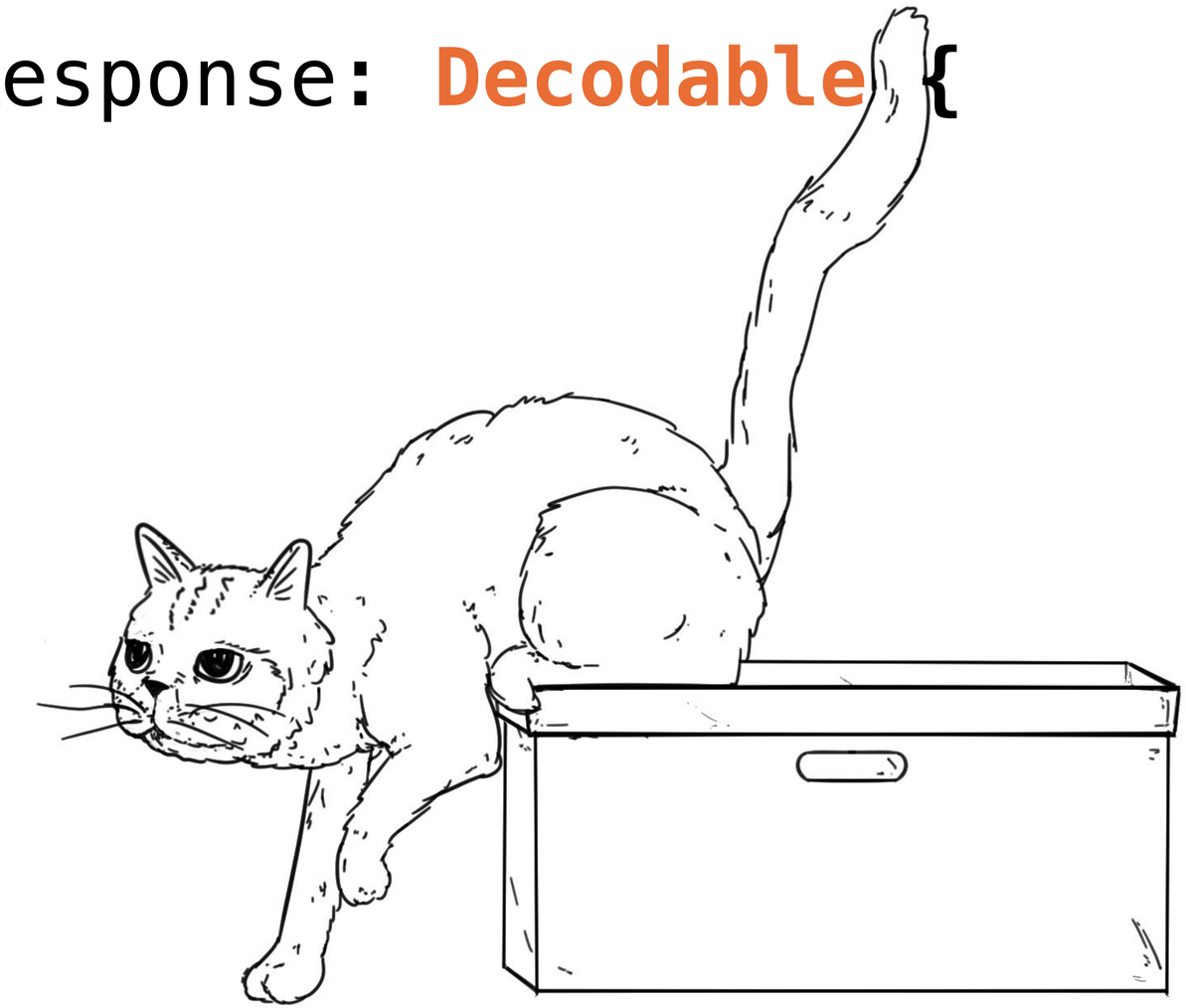
```
struct WebColorContrastResponse: Decodable {
  let ratio: CGFloat
  let AA: String
  let AALarge: String
  let AAA: String
  let AAALarge: String
}
```

```swift
struct WebColorContrastResponse: Decodable {
  init(from decoder: Decoder) throws {
    let values = try decoder.container(keyedBy: CodingKeys.self)
    ratio = try values.decode(CGFloat.self, forKey: .ratio)
    AA = try values.decode(String.self, forKey: .AA)
    AALarge = try values.decode(String.self, forKey: .AALarge)
    AAA = try values.decode(String.self, forKey: .AAA)
    AAALarge = try values.decode(String.self, forKey: .AAALarge)
  }

  enum CodingKeys: String, CodingKey {
    case ratio = "ratio"
    case AA = "AA"
    case AALarge = "AALarge"
    case AAA = "AAALarge"
    case AAALarge = "AAALarge"
  }

  let ratio: CGFloat
  let AA: String
  let AALarge: String
  let AAA: String
  let AAALarge: String
}
```

```
struct WebColorContrastResponse: Decodable {
  init(from decoder: Decoder) throws {
    let values = try decoder.container(keyedBy: CodingKeys.self)
    ratio = try values.decode(CGFloat.self, forKey: .ratio)
    AA = try values.decode(String.self, forKey: .AA)
    AALarge = try values.decode(String.self, forKey: .AALarge)
    AAA = try values.decode(String.self, forKey: .AAA)
    AAALarge = try values.decode(String.self, forKey: .AAALarge)
  }

  enum CodingKeys: String, CodingKey {
    case ratio = "ratio"
    case AA = "AA"
    case AALarge = "AALarge"
    case AAA = "AAALarge"
    case AAALarge = "AAALarge"
  }

  let ratio: CGFloat
  let AA: String
  let AALarge: String
  let AAA: String
  let AAALarge: String
}
```

NOPE!

```
struct WebColorContrastResponse: Decodable {
  let ratio: CGFloat
  let AA: String
  let AALarge: String
  let AAA: String
  let AAALarge: String
}
```



```swift
struct WebColorContrastResponse: Decodable {
  let ratio: CGFloat
  let AA: String
  let AALarge: String
  let AAA: String
  let AAALarge: String
}
```

```
struct WebColorContrastResponse: Decodable {
  let ratio: CGFloat
  let AA: String
  let AALarge: String
  let AAA: String
  let AAALarge: String
}
```

What if the key names don't match?

```
struct WebColorContrastResponse: Decodable {
  let ratio: CGFloat
  let smallDoubleA: String
  let largeDoubleA: String
  let smallTripleA: String
  let largeTripleA: String
}
```

What if the key names don't match?

```
struct WebColorContrastResponse: Decodable {

  let ratio: CGFloat
  let smallDoubleA: String
  let largeDoubleA: String
  let smallTripleA: String
  let largeTripleA: String
}
```

```
struct WebColorContrastResponse: Decodable {
  enum CodingKeys: String, CodingKey {
    case ratio
    case smallDoubleA = "AA"
    case largeDoubleA = "AALarge"
    case smallTripleA = "AAA"
    case largeTripleA = "AAALarge"
  }

  let ratio: CGFloat
  let smallDoubleA: String
  let largeDoubleA: String
  let smallTripleA: String
  let largeTripleA: String
}
```

```
struct WebColorContrastResponse: Decodable {
  enum CodingKeys: String, CodingKey {
    case ratio
    case smallDoubleA = "AA"
    case largeDoubleA = "AALarge"
    case smallTripleA = "AAA"
    case largeTripleA = "AAALarge"
  }

  let ratio: CGFloat
  let smallDoubleA: String
  let largeDoubleA: String
  let smallTripleA: String
  let largeTripleA: String
}
```

What if the types don't match?

```
struct WebColorContrastResponse: Decodable {

  enum CodingKeys: String, CodingKey {
    case ratio
    case tripleA = "AAA"
  }

  let ratio: CGFloat
  let tripleA: String
}
```

What if the types don't match?

```
struct WebColorContrastResponse: Decodable {
  init(from decoder: Decoder) throws {
    let values = try decoder.container(keyedBy: CodingKeys.self)
    let str = try values.decode(String.self, forKey: .ratio)
    guard let ratioAsDouble = Double(str) else {
      throw DecodingError.typeMismatch(…)
    }
    ratio = CGFloat(ratioAsDouble)
    tripleA = try values.decode(String.self, forKey: .tripleA)
  }

  enum CodingKeys: String, CodingKey {
    case ratio
    case tripleA = "AAA"
  }

  let ratio: CGFloat
  let tripleA: String
}
```

What if the types don't match?

Real World

```
let decoder = JSONDecoder()
decoder.keyDecodingStrategy = .convertFromSnakeCase

decoder.dateDecodingStrategy = .iso8601
```

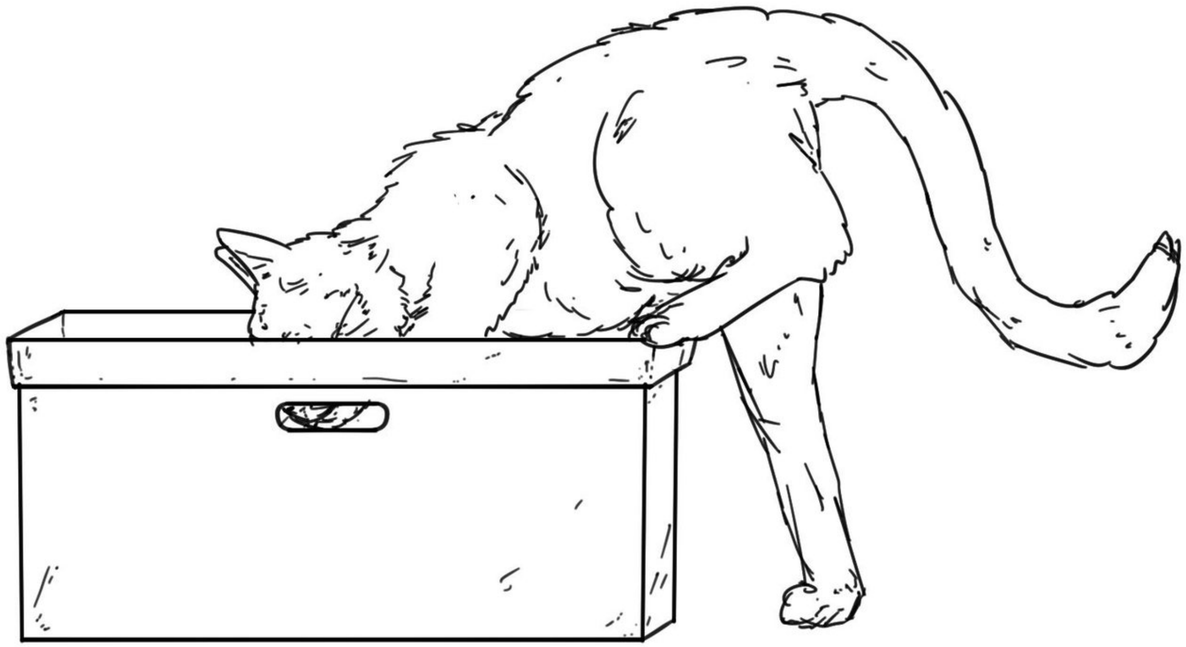
```swift
struct WebColorContrastResponse {
  let ratio: CGFloat
  let tripleA: String
}
```

```swift
extension WebColorContrastResponse: Decodable {
  init(from decoder: Decoder) throws {
    let values = try decoder.container(keyedBy: CodingKeys.self)
    let str = try values.decode(String.self, forKey: .ratio)
    guard let ratioAsDouble = Double(str) else {
      throw DecodingError.typeMismatch(…)
    }
    ratio = CGFloat(ratioAsDouble)
    tripleA = try values.decode(String.self, forKey: .tripleA)
  }

  enum CodingKeys: String, CodingKey {
    case ratio
    case tripleA = "AAA"
  }
}
```

```swift
struct ColorContrast {
  let ratio: CGFloat
  let smallTextRating: Rating
  let largeTextRating: Rating
}

extension ColorContrast {
  init?(webResponse: WebColorContrastResponse) {
    guard let ratioAsDouble = Double(webResponse.ratio) else { return nil }
    ratio = CGFloat(ratioAsDouble)
    smallTextRating = Rating(webResponse.AA, webResponse.AAA)
    largeTextRating = Rating(webResponse.AALarge, webResponse.AAALarge)
  }
}

enum Rating {
  case doubleAPass
  case tripleAPass
  case fail

  init(_ doubleA: String, _ tripleA: String) {
    switch(doubleA, tripleA) {
      case ("pass", "pass"): return .tripleAPass
      case ("pass", "fail"): return .doubleAPass
      default: return .fail
    }
  }
}
```

```swift
struct WebContrastResponse: Decodable {
  let ratio: String
  let AA: String
  let AALarge: String
  let AAA: String
  let AAALarge: String
}
```

Encodable



NSArray
NSDictionary
NSData

NSArray
NSDictionary
NSData

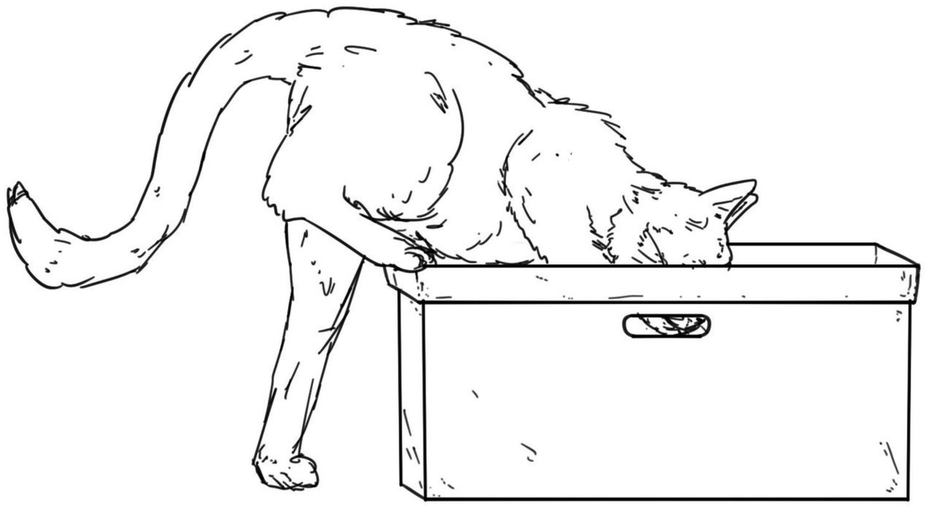
NSCoding & Codable

NSArray
NSDictionary
NSData

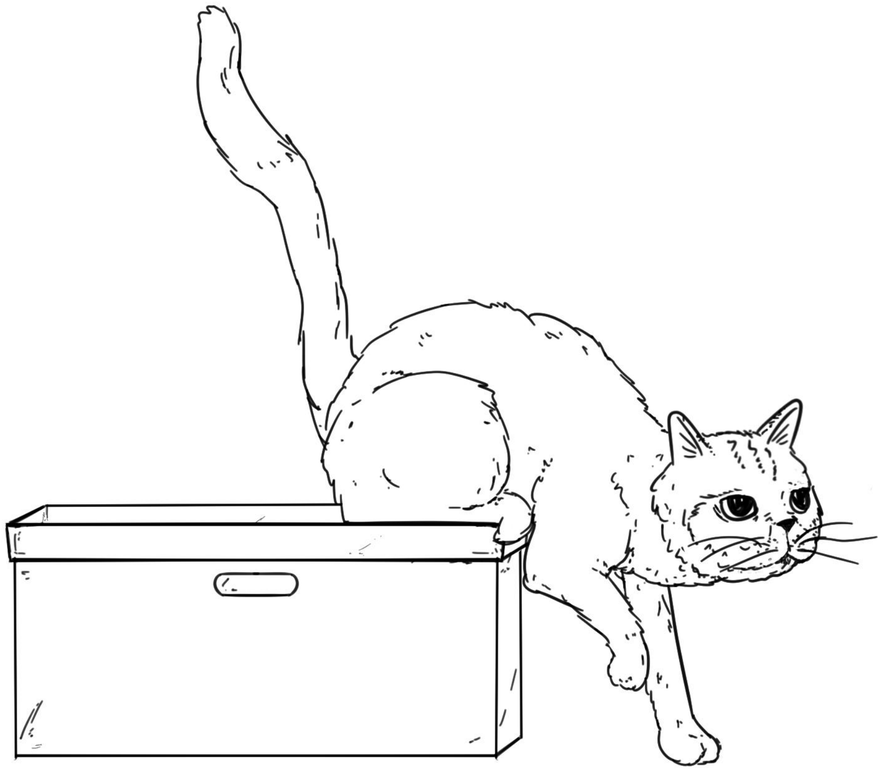
NSCodingable 🧐

NSCoding	Codable
iOS 2.0, macOS 10.0*	Xcode 9.0, Swift 4.0 (2017)
protocol	typealias for 2 protocols
Objective-C and Swift	Swift
NSObject subclasses	**structs, enums,** classes
manual conformance	automatic conformance*

Cannot conform to **NSCoding** or **Decodable** in a **class** extension.

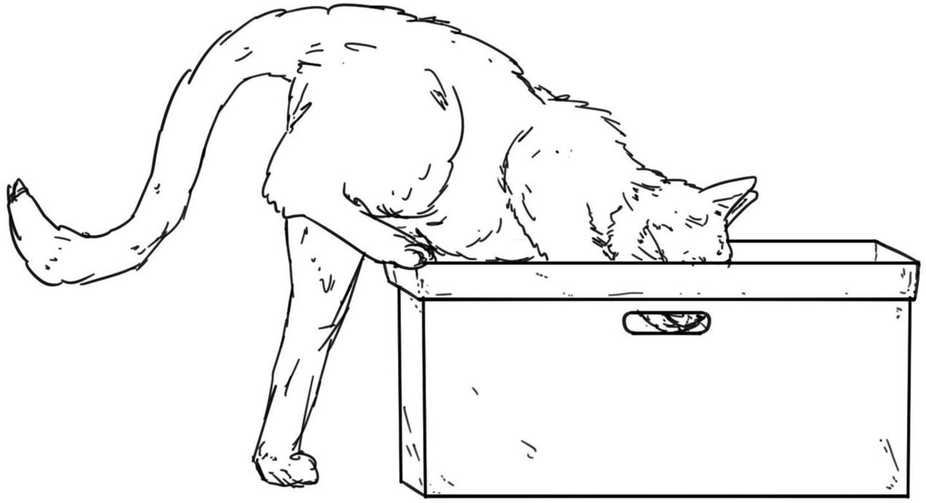


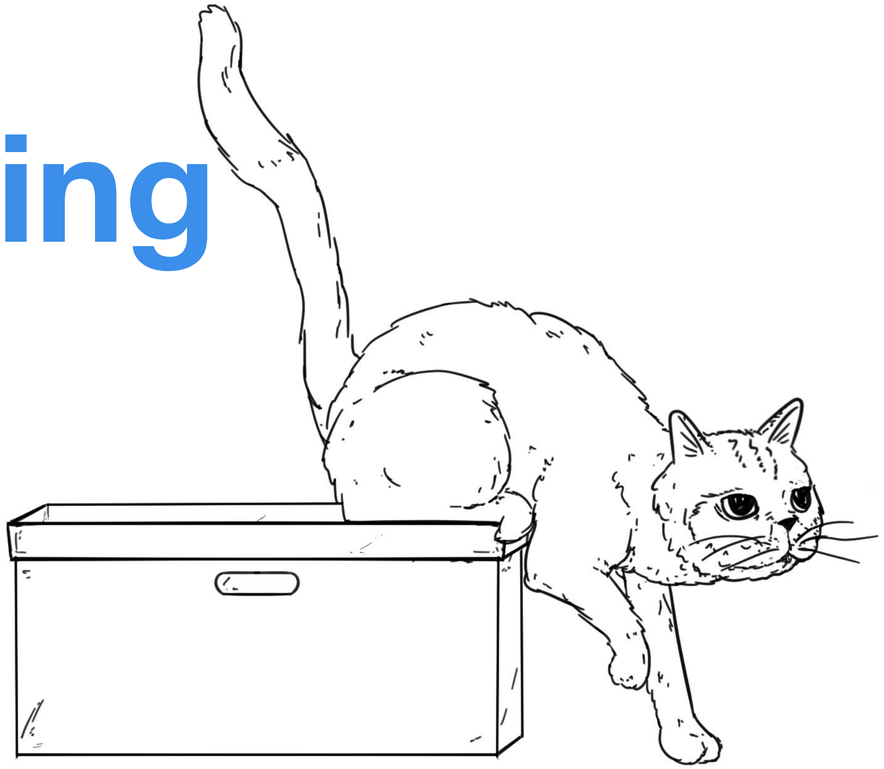
Encodable



Decodable

NSCoding





Encodable **Decodable**

Swift Playgrounds

(putting NSCoding stuff in a Coder)

Playground Page

```
let page =
 PlaygroundPage.current
let slide =
 PlaygroundSlide(text: "Clever
 Slide Title", textColor: ■,
 backgroundColor: ■)

let encoder = JSONEncoder()
let data = try
 encoder.encode(slide)
let proxy = page.liveView as!
 PlaygroundRemoteLiveViewProxy
proxy.send(.data(data))
```

Clever Slide Title

Stop

The Problem

```
struct Slide {
  let title: String
  let textColor: UIColor
  let backColor: UIColor
}
```

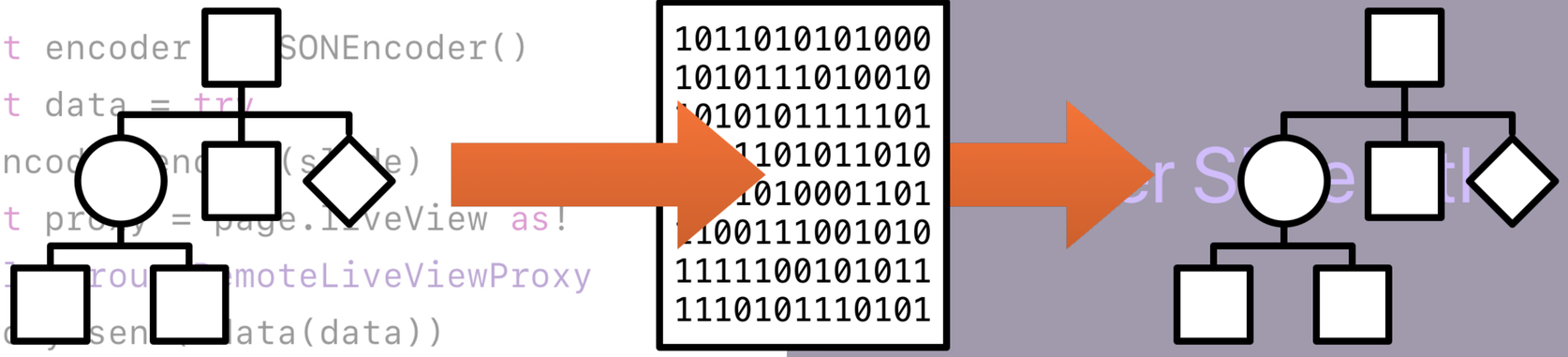
```
// UIViewController
func receive(_ message: PlaygroundValue) {…}
```

PlaygroundValue

```
(1) enum PlaygroundValue {
(2)     case boolean(Bool)
(3)     case floatingPoint(Double)
(4)     case integer(Int)
(5)     case string(String)
(6)     case array([PlaygroundValue])
(7)     case dictionary([String:PlaygroundValue])
(8)     case data(Data)
(9) }
```

Playground Page

```
let page =
 PlaygroundPage.current
let slide =
 PlaygroundSlide(text: "Clever
 Slide Title", textColor: ,
 backgroundColor: )

let encoder = JSONEncoder()
let data = try
 encoder.encode(slide)
let proxy = page.liveView as!
 PlaygroundRemoteLiveViewProxy
proxy.send(data(data))
```



Stop

Apple Inc.

Developer | Discover | Design | Develop | Distribute | Support | Account

Documentation › UIKit › Drawing › UIColor

Language: Swift | API Changes: Show

Class

UIColor

An object that stores color data and sometimes opacity (alpha value).

SDKs

iOS 2.0+

tvOS 9.0+

Relationships

Inherits From

NSObject

Conforms To

CVarArg
Equatable
Hashable
NSCopying
NSItemProviderReading
NSItemProviderWriting
NSSecureCoding

NSKeyed(Un)Archiver
methods to call

```
(1) // NSKeyedArchiver
(2) func encode(_ object: Any?, forKey key: String)


(4) // NSKeyedUnarchiver
(5) func decodeObject(forKey key: String) -> Any?

(7) // NSKeyedUnarchiver: NSSecureCoding
(8) func decodeObject<T>(of cls: T.Type, forKey
    key: String) -> T?
(9)     where T : NSObject, T : NSCoding
```

CodingKeys to use

```
(1) private enum CodingKeys: String, CodingKey {
(2)   case text
(3)   case textColor
(4)   case backColor
(5)     }
```

PlaygroundSlide: Encodable

```
(1) public func encode(to encoder: Encoder) throws {
(2)   var c = encoder.container(keyedBy: CodingKeys.self)
(3)   try c.encode(self.text, forKey: .text)
(4)   try c.encode(data(from: self.backColor),
                        forKey: .backgroundColor)
(5)   try c.encode(data(from: self.textColor),
                        forKey: .textColor)
(6) }
```

UIColor -> Data with NSKeyedArchiver

```
(1) private func data(from color: UIColor) -> Data {
(2)   let archiver = NSKeyedArchiver(forWritingWith:
                                     NSMutableData())
(3)   archiver.encode(color,
                forKey: PlaygroundSlide.nestedColorKey)
(4)   archiver.finishEncoding()
(5)   return archiver.encodedData
(6) }
```

PlaygroundSlide: Decodable

```
(1) public init(from decoder: Decoder) throws {
(2)   let container = try decoder.container(keyedBy:
    CodingKeys.self)
(3)   self.text = try container.decode(String.self,
    forKey: .text)

(5)   let backData = try container.decode(Data.self,
                                          forKey: .backColor)
(6)   self.backColor = PlaygroundSlide.color(from: backData)!

(8)   let textData = try container.decode(Data.self,
                                          forKey: .textColor)
(9)   self.textColor = PlaygroundSlide.color(from: textData)!
(10)}
```

Data -> UIColor with NSKeyedUnarchiver

```
(1) private static func color(from data: Data) ->
    UIColor? {
(2)   let unarchiver =
    NSKeyedUnarchiver(forReadingWith: data)
(3)   return unarchiver.decodeObject(of:
    UIColor.self, forKey:
    PlaygroundSlide.nestedColorKey)
(4)     }
(5)
```

Recap

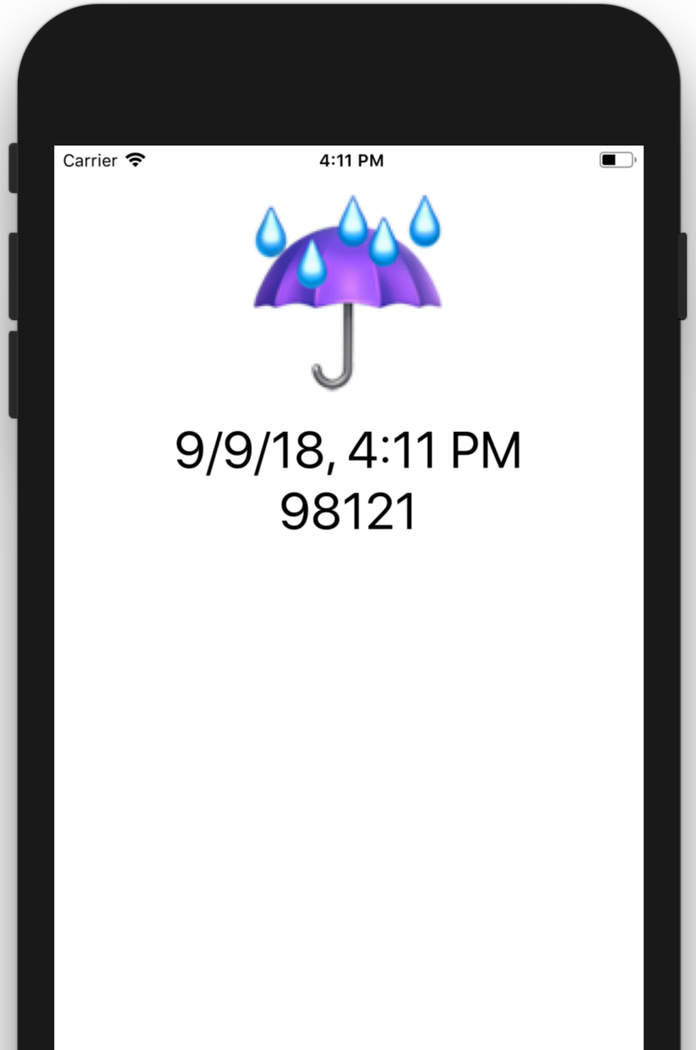
- The two halves of an iPad Playground are separate processes. Data can flow between them, but not much else.
- We can use **Codable** to get our struct back and forth.
- UIColor does not conform to **Codable**.
- We can't add **Codable** conformance to a class in an extension.
- UIColor conforms to **NSCoding**, so we can still **encode** some Data on the left and **decode** it on the right.

State Restoration

(putting Codable stuff in an NSCoder)

State Restoration

```
struct Forecast {
  let zipCode: Int
  let date: Date
  let description: Weather
}

enum Weather: String {
  case sunny = "😎"
  case drizzling = "🌧"
  case raining = "☔"
}
```



UIViewController methods to override

```
func encodeRestorableState(with coder: NSCoder)
func decodeRestorableState(with coder: NSCoder)
```

NSKeyed(Un)Archiver
methods to call

```
// NSKeyedArchiver
func encode(_ object: Any?, forKey key: String)

// NSKeyedUnarchiver
func decodeObject(forKey key: String) -> Any?

// NSKeyedUnarchiver: NSSecureCoding
func decodeObject<T>(of cls: T.Type,
                     forKey key: String) -> T?
    where T : NSObject, T : NSCoding
```

NSKeyed(Un)Archiver
methods to call

```
// NSKeyedArchiver
func encode(_ object: Any?, forKey key: String)

// NSKeyedUnarchiver
func decodeObject(forKey key: String) -> Any?

// NSKeyedUnarchiver: NSSecureCoding
func decodeObject<T>(of cls: T.Type,
                     forKey key: String) -> T?
    where T : NSObject, T : NSCoding
```

NSCoding

```
private enum Keys: String {
    case zipCode
    case date
    case description
}
override func encodeRestorableState(with coder: NSCoder) {
    super.encodeRestorableState(with: coder)
    guard let forecast = lastKnownForecast else { return }

    coder.encode(forecast.zipCode, forKey: Keys.zipCode.rawValue)
    coder.encode(forecast.date, forKey: Keys.date.rawValue)
    coder.encode(forecast.description, forKey: Keys.description.rawValue)
}
```

```
override func decodeRestorableState(with coder: NSCoder) {
   super.decodeRestorableState(with: coder)

   let zipCode = coder.decodeInteger(forKey: Keys.zipCode.rawValue)
   guard zipCode > 0 else { return }

   let objcDate = coder.decodeObject(of: NSDate.self, forKey: Keys.date.rawValue)
   guard let date = objcDate as Date? else { return }

   let objcWeather = coder.decodeObject(of: NSString.self,
                                        forKey: Keys.description.rawValue)
   guard let rawWeather = objcWeather as String? else { return }
   guard let description = Weather(rawValue: rawWeather) else { return }

   lastKnownForecast = Forecast(zipCode: zipCode,
                                date: date,
                                description: description)
}
```

What about Codable?

```
struct Forecast {
    let zipCode: Int
    let date: Date
    let description: Weather
}

enum Weather: String {
    case sunny = "😎"
    case drizzling = "🌧"
    case raining = "☔"
}
```

What about Codable?

```
struct Forecast: Codable {
    let zipCode: Int
    let date: Date
    let description: Weather
}

enum Weather: String, Codable {
    case sunny = "😎"
    case drizzling = "🌧"
    case raining = "☔"
}
```

NSKeyed(Un)Archiver
methods to call

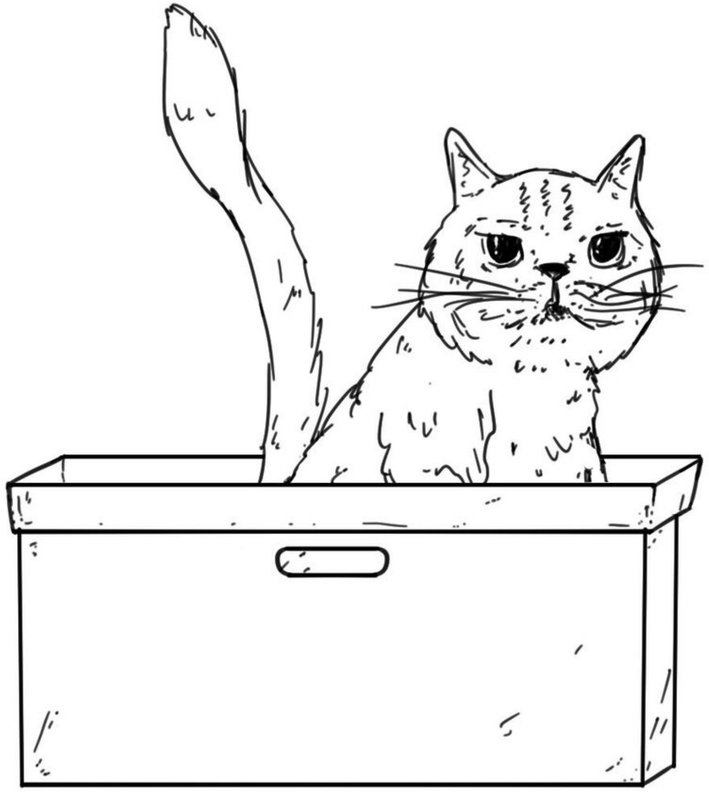
```
(1) // NSKeyedArchiver
(2) @nonobjc func encodeEncodable<T>(
        _ value: T, forKey key: String) throws
        where T : Encodable

(4) // NSKeyedUnarchiver
(5) @nonobjc func decodeDecodable<T>(
        _ type: T.Type, forKey key: String) -> T?
        where T : Decodable
```

NSCoding & Codable

```
private enum Keys: String { case forecast }

override func encodeRestorableState(with coder:
NSCoder) {
    super.encodeRestorableState(with: coder)
    guard let forecast = lastKnownForecast,
          let coder = coder as? NSKeyedArchiver else { return }
    try? coder.encodeEncodable(forecast, forKey: Keys.forecast.rawValue)
}

override func decodeRestorableState(with coder:
NSCoder) {
    super.decodeRestorableState(with: coder)
    guard let coder = coder as? NSKeyedUnarchiver else { return }
    lastKnownForecast = coder.decodeDecodable(Forecast.self,
                                              forKey: Keys.forecast.rawValue)
}
```

Recap

- State Restoration requires **NSCoding**.
- Simple structs are often useful for State Restoration.
- Simple structs can often auto-conform to **Codable**.
- **NSCoding** and **Codable** can be used together with **encodeEncodable** and **decodeDecodable**.



Linked List

`liz.micro.blog`

NSCoding
- NeXTstep docs
- General Info
- NSSecureCoding

Codable
- Introduction at WWDC 2017
- Swift Evolution Proposal

State Restoration
- Apple Documentation
- Sample Code

Playgrounds
- PlaygroundLiveViewMessageHandler
- Debugging iPad Playground View Controllers in Xcode
- Sample Code

Coders?

NSKeyedUnarchiver
NSKeyedArchiver
NSUnarchiver
NSArchiver

JSONDecoder
JSONEncoder
PropertyListDecoder
PropertyListEncoder