PREMISE: PROMISES

HI. I'M ANNE.

WHY PROMISES?

BRACE YOURSELF. I'M GOING TO SAY SOMETHING NICE ABOUT JAVASCRIPT

FIRST STOP IS ALWAYS WIKIPEDIA

- FIRST USED IN 1976-77

BARBARA LISKOV IS KINDA RAD

- FIRST USED IN 1976-77
- PIPELINING/CHAINING INVENTED IN 1988

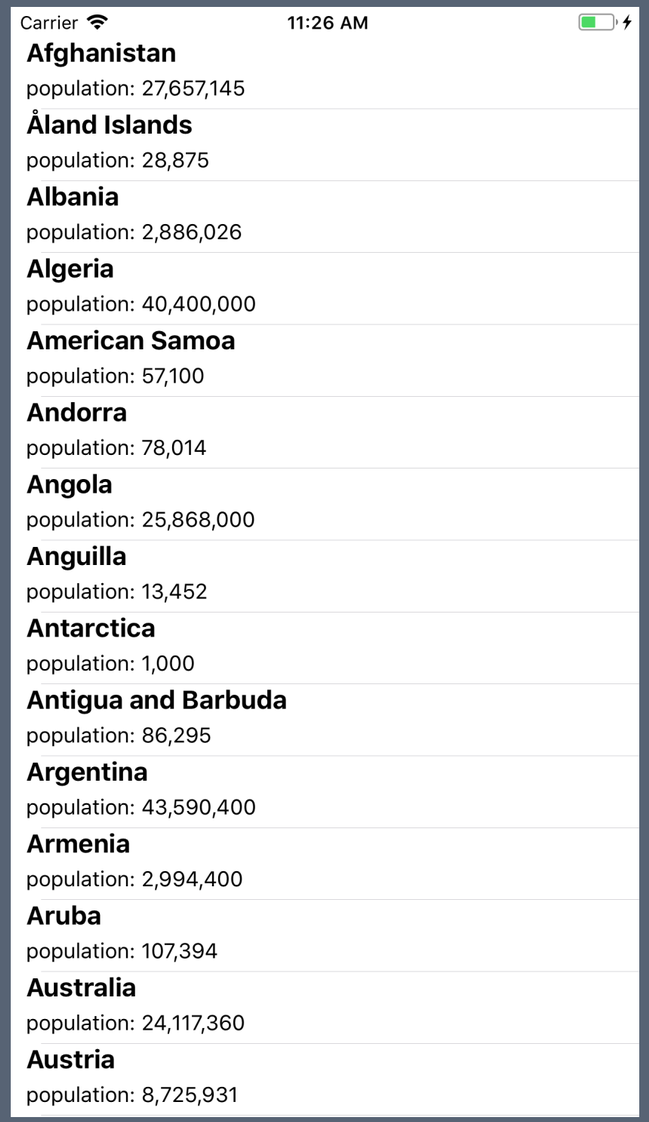
Y2K REVIVAL

- FIRST USED IN 1976-77
- PIPELINING/CHAINING INVENTED IN 1988
- RESURGENCE OF INTEREST IN 2000

WHAT IS A PROMISE AND WHY WOULD I WANT ONE?

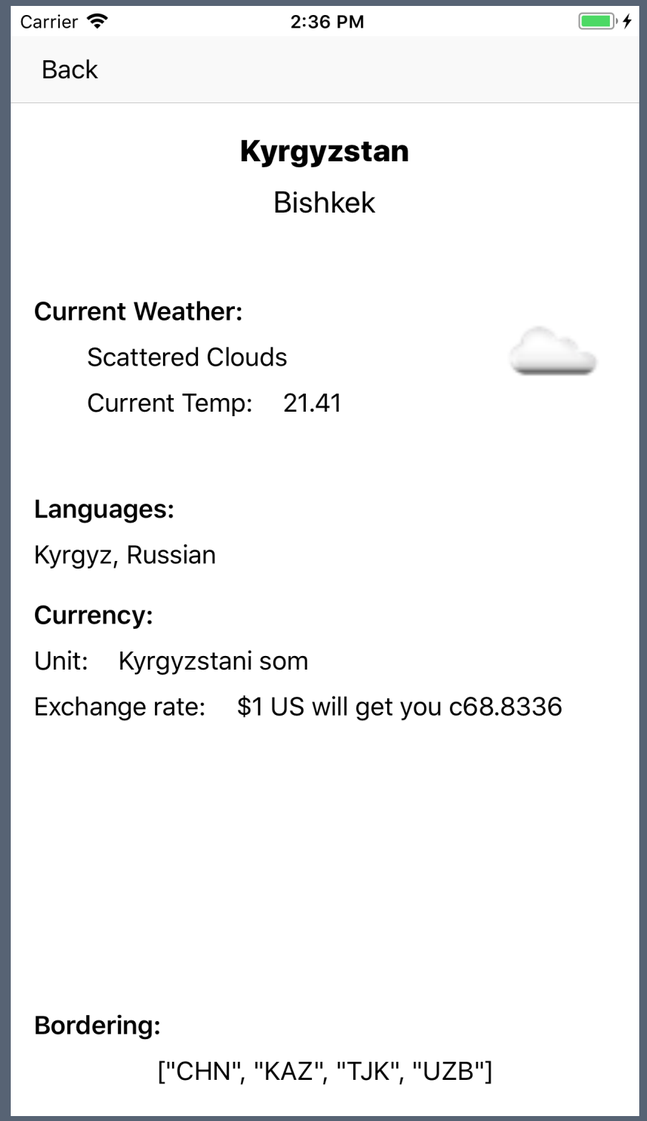
- REPRESENTS THE THE EVENTUAL RESULT OF AN ASYNCHRONOUS TASK
- OR THE ERROR IF THAT TASK FAILS
- A WAY OF FORMALIZING COMPLETION HANDLERS TO CHAIN ASYNCHRONOUS TASKS

SO I MADE AN APP



I HIT UP THE REST COUNTRIES API TO GET A LIST OF ALL THE COUNTRIES IN THE WORLD...

THEN I FOUND A WEATHER API AND A CURRENCY EXCHANGE API TO PLAY WITH



COUNTRYLIST VIEW CONTROLLER

```swift
func fetchAllCountries(handler:  @escaping ([Country]?) -> ()) {
    guard let url = URL(string: allCountriesURLString) else { return }

    let urlRequest = URLRequest(url: url)
    let session = URLSession.shared
    let task = session.dataTask(with: urlRequest, completionHandler: { data, response, error in
        guard error == nil else {
            print("🏞 request error")
            return
        }

        guard let responseData = data else {
            print("🏞 data response error")
            return
        }

        let countryArray: [Country] = self.decodeAllCountries(countryData: responseData)
        handler(countryArray)
    })

    task.resume()
}
```

```
override func viewDidLoad() {
        super.viewDidLoad()
        let networker = Networker()

        networker.fetchAllCountries { countries in
            guard let list = countries else { return }
            self.countryList = list
            DispatchQueue.main.async {
                self.tableView.reloadData()
            }
        }
    }
```

SELECTED COUNTRY VIEW CONTROLLER

```swift
func fetchCurrentExchangeRate(currencyCode: String, handler:  @escaping (ExchangeRate?) -> ()) {
        guard let currencyURL = URL(string: currencyConversionBaseURLString + currencyAccessKey + "&currencies=\(currencyCode)&format=1") else {
            print("💱 currency url error")
            return
        }

        let urlRequest = URLRequest(url: currencyURL)
        let session = URLSession.shared
        let task = session.dataTask(with: urlRequest, completionHandler: { data, response, error in

            guard error == nil else {
                print("💱 request error: \(String(describing: error))")
                return
            }

            guard let responseData = data else {
                print("💱 data response error")
                return
            }

            guard let exchangeRate: ExchangeRate = self.decodeExchangeRateData(currencyData: responseData) else {
                print("💱 decoding error")
                return
            }
            handler(exchangeRate)

        })

        task.resume()
    }
```

SIDE TRIP!

THIS IS WHY YOU GO TO MEETUPS!

FUN WITH CODABLE

BOTSWANA PULA

"USDBWP":9.591896

DANISH KRONE

"USDDKK":6.04103

POLISH ZŁOTY

"USDPLN":3.388799

FUN WITH CODABLE

```
struct Quote: Codable {
    var conversion: String = ""
    var rate: Float = 0.0
}
```

FUN WITH CODABLE

```
extension Quote {

    struct QuoteKeys: CodingKey {
        var stringValue: String
        var intValue: Int?

        init?(stringValue: String) {
            self.stringValue = stringValue
        }

        init?(intValue: Int) {
            return nil
        }
    }

    public init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: QuoteKeys.self)

        for key in container.allKeys {
            self.conversion = key.stringValue
            self.rate = try container.decode(Float.self, forKey: key)
        }
    }
}
```

SELECTED COUNTRY VIEW CONTROLLER

```swift
private func vanillaNetworkingGetTheStuff() {
        // guard some optional business

        networker.fetchCurrentExchangeRate(currencyCode: currencyCode) { rate in
            self.exchangeRate = rate
            DispatchQueue.main.async {
                self.setupExchangeRateUI()
            }
        }

        networker.fetchCapitalCityWeather(country: country) { weather in
            self.weather = weather
            DispatchQueue.main.async {
                self.setupWeatherUI()
            }

            guard let iconCode = self.weather?.conditions.first?.iconCode else {
                print("🌈 error unwrapping icon code")
                return
            }
            self.networker.fetchWeatherIcon(iconCode: iconCode) { weatherImage in
                DispatchQueue.main.async {
                    self.weatherIconImageView.image = weatherImage
                }

            }
        }
    }
```

🤔 SOME CONCERNS

- TWO SEPARATE NETWORK CALLS
- THAT COULD END AT TWO DIFFERENT TIMES
- A THIRD NETWORK CALL THAT DEPENDS ON ONE OF THE OTHERS

LET`S TRY PROMISES

RETURN A PROMISE OF A TYPE

```
func fetchAllCountries(handler:  @escaping ([Country]?) -> ())
```

VS

```
func promiseFetchAllCountries() -> Promise<[Country]>
```

FULFILL OR REJECT

```
func promiseFetchAllCountries() -> Promise<[Country]> {
        // some url and session business

        return Promise { seal in
            let task = session.dataTask(with: urlRequest) { data, _, error in
                if let responseData = data {
                    let allCountries = self.decodeAllCountries(countryData: responseData)
                    seal.fulfill(allCountries)
                } else if let requestError = error {
                    seal.reject(requestError)
                }
            }

            task.resume()
        }
    }
```

SIDE TRIP!

THE MARCH OF PROGRESS VS. THE INTERNET IS FOREVER

PROMISEKIT 6.0 INCLUDED A MAJOR CHANGE IN THE PROMISE INITIALIZER

FROM

```
Promise { fulfill, reject in
    //…
}
```

TO

```
Promise { seal in
    // ...
}
```

COUNTRYLIST VIEW CONTROLLER

```
override func viewDidLoad() {
        super.viewDidLoad()
        let networker = Networker()

        firstly {
            networker.promiseFetchAllCountries()
        }.done { countryArray in
            self.countryList = countryArray
            self.tableView.reloadData()
        }.catch { error in
            print("📝 some kind of error listing all countries -> \(error)")
        }
    }
```

ENTER THE CIRCLE OF SHARING

handler

IS GARBAGE

I MEAN, LOOK AT THIS:

```
func fetchAllCountries(handler:  @escaping ([Country]?) -> ())
```

LOOK AT THIS MESS:

```
func fetchAllCountries(handler:  @escaping ([Country]?) -> ())
```

› IS THIS CLEAN CODE?

AND WHAT ABOUT THIS:

```
func duckBusiness() {
    doAThing {
        quackLikeADuck()
    }
}

func doAThing(handler: ()-> ()) {
    doSomeStuff()
    doAnotherThing {
        handler()
    }
}

func doAnotherThing(handler: ()->()) {
    doSomeMoreThings()
    handler()
}
```

UGH. IT'S THE WORST:

```
func fetchAllCountries(handler:  @escaping ([Country]?) -> ())
```

- IS THIS CLEAN CODE?
- CHAINING IS IMPOSSIBLE
- FUCKINGBLOCKSYNTAX.COM
  AND
  FUCKINGCLOSURESYNTAX.COM

THANK YOU FOR SHARING

PROMISES HANDLE

handler

CHECK THIS OUT

```
firstly {
        networker.promiseFetchAllCountries()
    }.done { countryArray in
        self.countryList = countryArray
        self.tableView.reloadData()
    }.catch { error in
        print("📝 some kind of error listing all countries -> \(error)")
    }
```

THAT WAS EASY MODE. WHAT ABOUT THE HARD STUFF?

```
func promiseFetchCurrentExchangeRate(currencyCode: String) -> Promise<ExchangeRate> {
    ... }

func promiseFetchCapitalCityWeather(country: Country) -> Promise<Weather> {
    ... }

func promiseFetchWeatherIcon(iconCode: String) -> Promise<UIImage> {
    ... }
```

```
firstly {
    when(fulfilled:
        networker.promiseFetchCurrentExchangeRate(currencyCode: currencyCode),
        networker.promiseFetchCapitalCityWeather(country: country))
    }.done { exchangeRate, weather in
        self.exchangeRate = exchangeRate
        self.weather = weather

        guard let iconCode = weather.conditions.first?.iconCode else { return }
        self.networker.promiseFetchWeatherIcon(iconCode: iconCode).done { weatherImage in
            self.weatherIconImageView.image = weatherImage
        }
    }.catch { error in
        print("🏞 error in getting the data for \(String(describing: country.name)) -> \(error)")
    }.finally {
        self.setupExchangeRateUI()
        self.setupWeatherUI()
        self.activityIndicator.stopAnimating()
    }
```

WHEN

```
when(fulfilled:
    networker.promiseFetchCurrentExchangeRate(currencyCode: currencyCode),
    networker.promiseFetchCapitalCityWeather(country: country)
)
```

DONE

```
.done { exchangeRate, weather in
        self.exchangeRate = exchangeRate
        self.weather = weather

        guard let iconCode = weather.conditions.first?.iconCode else { return }
        self.networker.promiseFetchWeatherIcon(iconCode: iconCode).done { weatherImage in
            self.weatherIconImageView.image = weatherImage
      }
```

CATCH

```
.catch { error in
            print("🗺️ some kind of error in getting the data for \(String(describing: country.name)) -> \(error)")
        }
```

FINALLY

```
.finally {
            self.setupExchangeRateUI()
            self.setupWeatherUI()
            self.activityIndicator.stopAnimating()
        }
```

```
firstly {
    when(fulfilled:
        networker.promiseFetchCurrentExchangeRate(currencyCode: currencyCode),
        networker.promiseFetchCapitalCityWeather(country: country))
    }.done { exchangeRate, weather in
        self.exchangeRate = exchangeRate
        self.weather = weather

        guard let iconCode = weather.conditions.first?.iconCode else { return }
        self.networker.promiseFetchWeatherIcon(iconCode: iconCode).done { weatherImage in
            self.weatherIconImageView.image = weatherImage
        }
    }.catch { error in
        print("🏞 error in getting the data for \(String(describing: country.name)) -> \(error)")
    }.finally {
        self.setupExchangeRateUI()
        self.setupWeatherUI()
        self.activityIndicator.stopAnimating()
    }
```

WAIT A SECOND...

```
self.networker.promiseFetchWeatherIcon(iconCode: iconCode)
    .done { weatherImage in
        self.weatherIconImageView.image = weatherImage
    }
```

```
firstly {
    when(fulfilled:
        networker.promiseFetchCurrentExchangeRate(currencyCode: currencyCode),
        networker.promiseFetchCapitalCityWeather(country: country))
    }.then { exchangeRate, weather in
        self.exchangeRate = exchangeRate
        self.weather = weather

        guard let iconCode = weather.conditions.first?.iconcode else { return }
        networker.promiseFetchWeatherIcon(iconCode: iconCode)
    }.done { weatherImage in
        self.weatherIconImageView.image = weatherImage
    }.catch { error in
        print("🏞 error in getting the data for \(String(describing: country.name)) -> \(error)")
    }.finally {
        self.setupExchangeRateUI()
        self.setupWeatherUI()
        self.activityIndicator.stopAnimating()
    }
```

❗ Ambigious reference to member 'firstly(execute:)'

CHANGE YOUR TOOLS.
CHANGE YOUR MIND

PROS & CONS

SEEMS LIKE A LOT OF OVERHEAD FOR A SMALL PROJECT

SEEMS LIKE A LOT OF OVERHEAD FOR A SMALL PROJECT

SYNTAX IS CLEARLY MORE READABLE

SEEMS LIKE A LOT OF OVERHEAD FOR A SMALL PROJECT

SYNTAX IS CLEARLY MORE READABLE

NEW(ISH) IDEA AROUND AN OLD PROBLEM

SEEMS LIKE A LOT OF OVERHEAD FOR A SMALL PROJECT

SYNTAX IS CLEARLY MORE READABLE

NEW(ISH) IDEA AROUND AN OLD PROBLEM

PROMISEKIT IS...OKAY

THE OTHER OPTIONS

- GOOGLE PROMISES

THE OTHER OPTIONS

- GOOGLE PROMISES
- BRIGHTFUTURES

THE OTHER OPTIONS

- GOOGLE PROMISES
- BRIGHTFUTURES
- HYDRA

IF I HAD TO DO IT ALL OVER AGAIN...

QUESTIONS 

THANKS!

 @NORTHOFNORMAL

 NORTHOFNORMAL

 GITHUB.COM/NORTHOFNORMAL/PROMISESPROMISES