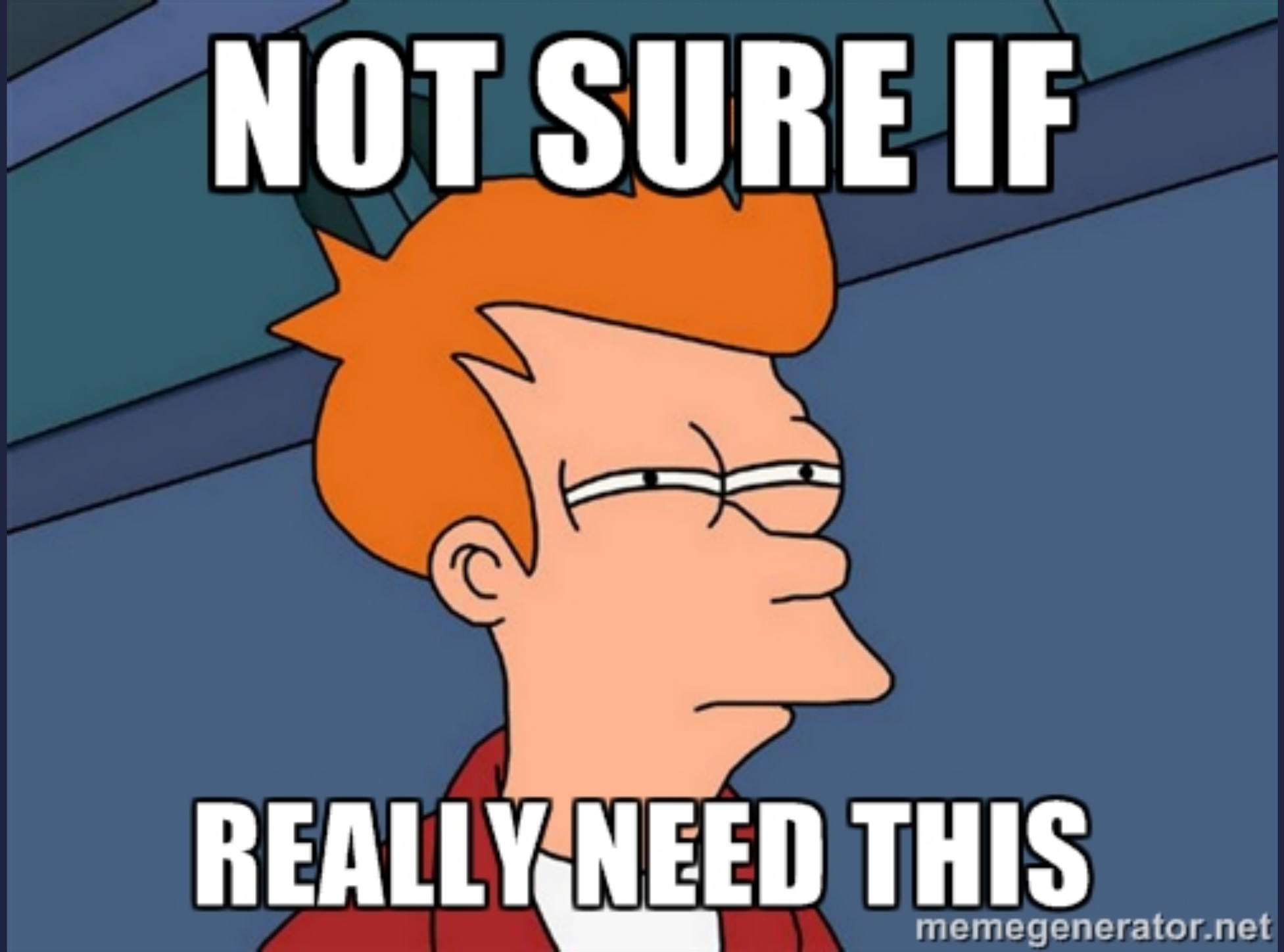
REACTIVECOCOA GOODNESS

PART I OF II

3

PHASES OF LEARNING REACTIVECOCOA

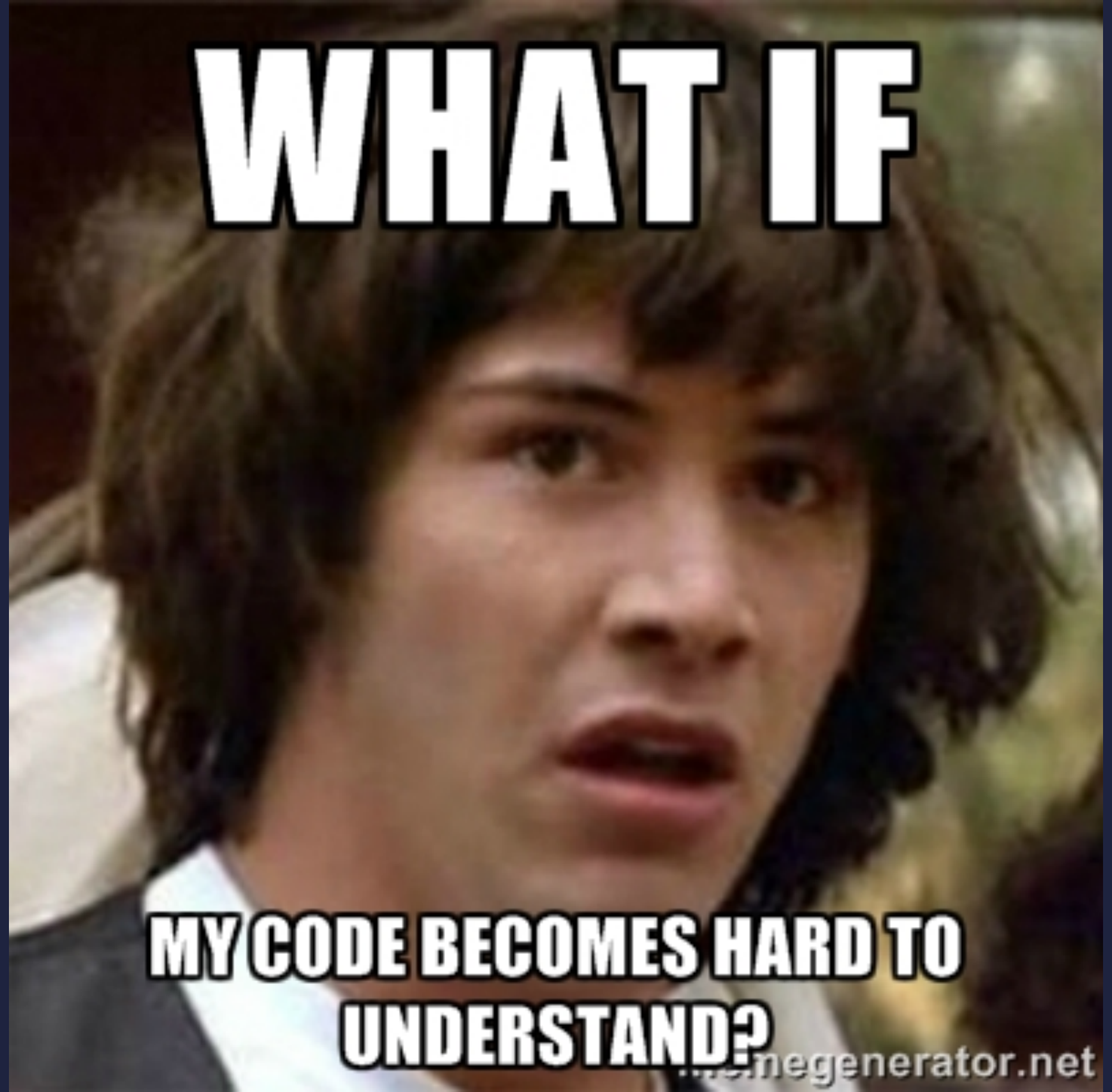
1


NOT SURE IF
REALLY NEED THIS
memegenerator.net

2


REACTIFY
ALL THE THINGS
memegenerator.net

3


WHAT IF
MY CODE BECOMES HARD TO UNDERSTAND?
negenerator.net

WHAT IS REACTIVECOCOA?

AUTHORS (GITHUB FOLKS) SAY:

ReactiveCocoa (RAC) is an Objective-C framework inspired by Functional Reactive Programming. It provides APIs for composing and transforming streams of values.

THE IDEA TO CREATE REACTIVECOCOA SEEMS TO HAVE COME FROM RX (MICROSOFT .NET), WHICH IS SIMILAR.

RAC ALLOWS US AN EVENT-BASED PROGRAMMING MODEL WHICH MAKES STATE TRACKING OBSOLETE - *in theory.*

ONE THING IS CERTAIN:
IT MAKES STATE EASIER TO MAINTAIN.

SOME PEOPLE THINK RAC IS ALL ABOUT THIS...

```
// Keeps someObject.property synced with
// otherObject.other.property
// -> one-way binding
RAC(someObject, property) =
  RACObserve(otherObject, other.property);
```

...OR THIS...

```
// Keeps someObject.property synced with
// otherObject.other.property
// and vice versa -> two-way binding
RACChannelTo(someObject, property) =
  RACChannelTo(otherObject, other.property);
```

... BUT RAC IS SO MUCH MORE!

RAC = SIGNALS + SUBSCRIBERS

A SIGNAL IS A STREAM OF VALUES.

SIGNALS CAN BE TRANSFORMED, COMBINED, ETC.

A SUBSCRIBER SUBSCRIBES TO A SIGNAL.

RAC LETS BLOCKS, OBJECTS, AND PROPERTIES SUBSCRIBE TO SIGNALS.

USING SIGNALS AND SUBSCRIBERS, YOU CAN MODEL ANY REACTION TO ANY EVENT THAT HAPPENS IN YOUR APPLICATION.

MOST OF UI-CENTERED CODE ACTUALLY REACTS TO SOMETHING!

DIVING INTO SIGNALS

MOST TRIVIAL SIGNAL:

RACEmptySignal

```
// Sends no value. Ever. #yolo
RACSignal *sendNothing =
  [RACEmptySignal empty];
```

MOST TRIVIAL SIGNAL THAT SENDS VALUES:

RACReturnSignal

```
// All subscribers will immediately receive an
// empty array as 'next' value on subscription.
RACSignal *returnEmptyArrayOnSubscription =
  [RACReturnSignal return:@[]];
```

BRIDGE OOP/FRP WORLD:

RACSubject

```
- (id)init {
    if (self = [super init]) {
        self.loadMoreSubject = [RACSubject subject];
    }

    return self;
}

- (void)loadMore {
    [self.loadMoreSubject sendNext:nil];
}
```

OTHER SIGNALS:

RACSignal (DUH!),
RACReplaySubject,
RACChannelTerminal,
RACErrorSignal,…

UIKIT BATTERIES INCLUDED:

```
UITextfield.rac_textSignal
NSObject.rac_willDeallocSignal
UITableviewCell.rac_prepareForReuseSignal
UIButton.rac_command
UIRefreshControl.rac_command
...
```

SIGNAL LIBRARIES ON COCOAPODS

```
pod search RACExtensions
pod search reactivecocoa
```

YOU BUILD APP LOGIC BY TRANSFORMING VALUES SENT BY SIGNALS

MAP

```
[usersSignal map:^NSString *(MMUser *user) {
    return user.email;
}];
```

FILTER

```
[usersSignal filter:^BOOL(MMUser *user) {
    return user.isEnabled;
}];
```

ZIP

```
// If signalOne and signalTwo have both
// sent a value, signalOne's value is passed on.
[RACSignal zip:@[signalOne, signalTwo]
    reduce:^id(RACTuple *t) {
        return t.first;
}];
```

MERGE SIGNALS

```
// No matter if the user or a notification
// provokes a value send, the value is passed
// on in both cases.
[RACSignal merge:@[userTriggerSignal,
    notificationCenterTriggerSignal]]
```

FLATTEN OR MAP/FLATTEN SIGNALS

```
// First maps all freshly sent requestParams to a new
// signal that will eventually send a server response.
// Then applies flatten such that we get all those
// server responses directly as values.
RACSignal *currentUsers = [requestParamsSignal
    flattenMap:^RACStream *(NSDictionary *requestParams) {
        return [MMAPI.sharedInstance
            allUsersWithParams:requestParams];
    }
];
```

(YEAH, YOU CAN BUILD SIGNALS OF SIGNALS OF SIGNALS.)


That's the evilest thing I can imagine.

CONCAT SIGNALS

```
// Catches any error in signal, waits 1 second,
// then passes on the error, and then immediately
// retries (= resubscribes). Forever.
[[signal catch:^(NSError *error) {
    return [[[RACSignal
              empty]
              delay:1.0]
              concat:[RACSignal error:error]];
}] retry];
```

SCAN/REDUCE

```
// Starts off with a mutable array, and adds all
// events that are ever sent by the 'events' signal.
// E.g. for infinite scrolling.
RACSignal *combinedEventsInAllPages = [events
    scanWithStart:NSMutableArray.new
    reduce:^id(NSMutableArray *running,
      NSArray *eventsInPage) {
        [running addObject:eventsInPage];
        return running;
    }];
```

SKIP VALUES

```
// RACObserve immediately sends the current
// property value. Skip:1 'swallows' that one.
[[RACObserve(self, someProperty)] skip: 1];
```

TAKE X VALUES AND STOP

```
// Only sends the first 3 text values.
// Then the signal is disposed of.
[textField.rac_textSignal take: 3];
```

DISTINCT UNTIL CHANGED

```
// Does only send distinct (= first and new) values.
[RACObserve(self, isUserLoggedIn) distinctUntilChanged];
```

THROTTLE/DELAY/REPEAT...
... THERE IS A TRANSFORM FOR EVERYTHING.

SIGNALS CAN SWITCH THREADS

```
RAC(self, results) = [[[[RACReturnSignal
    return:@[@(M_PI), @(M_E), @(M_SQRT1_2)]]
    deliverOn:RACScheduler.scheduler]
    // All that follows happens on a background thread
    map:^NSNumber *(NSNumber *specialNumber) {
        // Do expensive calculations
        // NSNumber *result = ...
        return result;
    }]
    // All that follows (RAC assignment) happens
    // on the main thread
    deliverOn:RACScheduler.mainThreadScheduler];
```

RACCOMMAND = ONE-OFF SIGNAL WRAPPER

```
self.registerCommand = [[RACCommand alloc] initWithEnabled:
    [self.emailField.rac_textSignal map:^id(NSString *text) {
        return @(text.length > 0);
    }]
    signalBlock:^RACSignal *(NSDictionary *params) {
        return [MMAPI.sharedInstance registerWithParams:params];
    }];

[self.registerCommand.executionSignals.flatten subscribeNext:^(id result) {
    NSLog(@"Successfully registered!");
}];

// ...

[self.registerCommand execute:@{MMAPIRegisterEmail: self.emailField.text,
  MMAPIRegisterPassword: self.pwField.text}];
```

SIGNALS CAN BE HOT OR COLD

A hot signal is a signal that sends values (and presumably does work) regardless of whether it has any subscribers.

A cold signal is a signal that defers its work and the sending of any values until it has a subscriber.

Hot signals often don't send subscribers all values of all time, but only values after their subscription time.

SO WE HAVE SIGNALS.
NOW WE NEED TO SUBSCRIBE.

SUBSCRIBE WITH A BLOCK

```
// Prints current date & time every second
RACSignal *dates = [RACSignal interval:1.0
    onScheduler:RACScheduler.mainThreadScheduler];
[dates subscribeNext:^(NSDate *date) {
    NSLog([NSDateFormatter
        localizedStringFromDate:[NSDate date]
        dateStyle:NSDateFormatterShortStyle
        timeStyle:NSDateFormatterFullStyle]);
}];
```

SUBSCRIBE WITH A PROPERTY (KEYPATH)

```
// RAC() is a clever macro that abuses
// subscripting to let subscription look
// like a left-hand assignment
RAC(self.proxyObject.awesomeArray) =
   [RACReturnSignal return:@[]];
```

INJECT BLOCK-BASED SIDE-EFFECTS

```
// doNext: is 'woven' into the signal chain,
// the side-effect becomes part of the signal.
// Dangerous, but sometimes necessary!
parametersOnRefreshOrParameterChanged =
    [parametersOnRefreshOrParameterChanged
        doNext:^(NSDictionary *parameters) {
            @strongify(self)
            self.isRefreshing = YES;
    }];
```

SUBSCRIBE WITH A METHOD

```
// Calls updateAuthorizationWithAuthToken: when
// RACObserve() sends a value. RACObserve()
// will btw immediately send the first value,
// synchronously.
[self rac_liftSelector:
     @selector(updateAuthorizationWithAuthToken:)
     withSignals: RACObserve(MMSession.sharedSession,
          currentCredentials.authToken), nil];
```

WHAT RAC BUYS YOU

CAN BIND UI COMPONENT VALUES TO MODEL VALUES, INCLUDING TRANSFORMATIONS ...

... WHICH LEADS TO:
A REAL MVVM PARADIGM

CAN CALL METHODS WITH MULTIPLE PARAMETERS WHEN SIGNALS FIRE IN A CERTAIN CONFIGURATION

ELIMINATES THOSE ERROR-PRONE BOOL FLAGS YOU USE TO KEEP TRACK OF COMPLEX STATE

CAN WAIT FOR MULTIPLE SIGNALS UNTIL IT DOES SOMETHING
(TEDIOUS AND DANGEROUS TO DO WITH BOOL FLAGS)

MAKES THREAD SWITCHES EASY AS 

FORCES YOU TO THINK TWICE BEFORE INTRODUCING A SIDE-EFFECT.

ENCOURAGES IMMUTABLE OBJECTS
(BUT DOES NOT ENFORCE THEM).

MAKES CODE EVEN MORE REUSABLE (E.G.NETWORK RESPONSE ERROR FILTER)

NICE INTERFACE TO KVO AND NSNotificationCenter

HAS NICE COLLECTION UTILS INDEPENDENT OF SIGNALS (MIGHT GET REMOVED SOMEDAY):

RACSequence

LIMITATIONS & PITFALLS

BLOCKS CAN EASILY CREATE RETAIN CYCLES

 ALWAYS USE @WEAKIFY AND @STRONGIFY

IF YOU OVERDO IT, PERFORMANCE SUFFERS

 WATCH BINDINGS IN UITABLEVIEWCELLS!

NO WAY FOR DISTINCTUNTILCHANGED TO NOT USE ISEQUAL 😢

NO SPECIAL TREATMENT FOR MUTABLE ARRAYS

[[[[[[[[[CODE BECOMES] HARD] TO] UNDERSTAND] IF YOU] NEST] SIGNALS] TOO MUCH]

 USE INTERMEDIATE PROPERTIES!

HOW DOES RAC DO ALL THIS?

BLOCKS, KVO, METHOD SWIZZLING, MACROS, GCD, LOCKS,...

(maybe every runtime feature there is)

MEMORY MANAGEMENT *just works*™.
BUT IT'S EASY TO CREATE SIGNALS THAT LIVE FOREVER!

RAC'S CODE IS REALLY ADVANCED STUFF,
BUT HAVING A LOOK AT IT IS WORTH IT.

A WORD ABOUT SWIFT:

RAC can be used with Swift (didn't test it yet),
but a pure Swift implementation will take a while.
Also, KVO will not work without NSObject anyway.

Proof of concept is currently being merged into RAC:

`https://github.com/jspahrsummers/RxSwift`

IF YOU WANT TO LEARN RAC:

1. First have a look at how subscriptions work
2. Read through RAC's Github issues (mostly RAC Q/A)
3. Start slow, create a lot of intermediate properties
4. Go wild!

IF YOU ARE STUCK,
PING ME AT
@MANUELMALY
NO REALLY. DO IT!

THANKS FOR LISTENING!

COMING SOON: PART II INCLUDING MVVM PATTERN AND MORE CODE

@MANUELMALY

CREATIVEPRAGMATICS.COM