

Functional
Reactive
Intuition

Functional
Reactive = streams of
values over
time

streams of
values over
time

=

pipes!
signals!
streams!

pipes!
signals!
streams!

=

map, filter,
reduce on
your callbacks

map, filter,
reduce on
your callbacks

=

no more

```
var isValidEmailAddress = false  
var isValidPassword = false  
var isButtonPressed = false  
var isNetworkOperationInProgress = false
```

The brief

“When the user starts simultaneously panning and rotating an object, start a countdown from 3. Stop the timer either when the countdown ends or when the user stops both gestures.”

Let me take you
to imperative-land

when the user touches the screen

check **if** a user is panning an object, store that information

check **if** both gestures are running simultaneously, **if** yes, start a timer, from 3, counting down.

check **if** the user is rotating an object, store that information

check **if** both gestures are running simultaneously, **if** yes, start a timer, from 3, counting down.

check **if** the user stopped panning

stop the timer **if** needed

check **if** the user stopped rotating

stop the timer **if** needed

when the timer ticks, decrease the number of seconds left

if the number of seconds left is zero, stop the timer

```
var panPresent = false
var pinchPresent = false
var gestureTimer: NSTimer?
var secondsLeft = 3

func handlePan(panGesture: UIPanGestureRecognizer) {
    if panGesture.state == .Began && self.panPresent == false {
        self.panPresent = true
        self.checkIfBothGesturesPresent()
    } else if panGesture.state == .Ended {
        self.panPresent = false
        self.stopTimerIfNeeded()
    }
}

func handlePinch(pinchGesture: UIPinchGestureRecognizer) {
    if pinchGesture.state == .Began && self.pinchPresent == false {
        self.pinchPresent = true
        self.checkIfBothGesturesPresent()
    } else if pinchGesture.state == .Ended {
        self.pinchPresent = false
        self.stopTimerIfNeeded()
    }
}

func checkIfBothGesturesPresent() {
    if self.pinchPresent == true && self.panPresent == true && self.gestureTimer == nil {
        self.secondsLeft = 3
        self.gestureTimer = NSTimer.scheduledTimerWithTimeInterval(1, target: self, selector: "tick:"...
        print("started")
    }
}

func stopTimerIfNeeded() {
    if let gestureTimer = gestureTimer {
        gestureTimer.invalidate()
        self.gestureTimer = nil
        print("completed")
    }
}

func tick(timer: NSTimer) {
    if self.secondsLeft <= 0 {
        self.stopTimerIfNeeded()
        return
    }
    self.secondsLeft--
    print("tick")
}
```

What would this look
like if you replaced all
“**if**” instances with
“**when**”?

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    let pan = UIPanGestureRecognizer()  
    let pinch = UIPinchGestureRecognizer()  
  
    let panStarted = pan.rx_event.filter { gesture in gesture.state == .Began }  
    let panEnded = pan.rx_event.filter { gesture in gesture.state == .Ended }  
  
}
```

```
override func viewDidLoad() {
    super.viewDidLoad()

    let pan = UIPanGestureRecognizer()
    let pinch = UIPinchGestureRecognizer()

    let panStarted = pan.rx_event.filter { gesture in gesture.state == .Began }
    let panEnded = pan.rx_event.filter { gesture in gesture.state == .Ended }

    let pinchStarted = pinch.rx_event.filter { gesture in gesture.state == .Began }
    let pinchEnded = pinch.rx_event.filter { gesture in gesture.state == .Ended }

}
```

```
override func viewDidLoad() {
    super.viewDidLoad()

    let pan = UIPanGestureRecognizer()
    let pinch = UIPinchGestureRecognizer()

    let panStarted = pan.rx_event.filter { gesture in gesture.state == .Began }
    let panEnded = pan.rx_event.filter { gesture in gesture.state == .Ended }

    let pinchStarted = pinch.rx_event.filter { gesture in gesture.state == .Began }
    let pinchEnded = pinch.rx_event.filter { gesture in gesture.state == .Ended }

    // condition: when both pan and pinch has begun
    let bothGesturesStarted = Observable.of(panStarted, pinchStarted).merge(maxConcurrent: 1)
    // condition: when both pan and pinch ended
    let bothGesturesEnded = Observable.of(panEnded, pinchEnded).merge()
}
```

```
override func viewDidLoad() {
    super.viewDidLoad()

    let pan = UIPanGestureRecognizer()
    let pinch = UIPinchGestureRecognizer()

    let panStarted = pan.rx_event.filter { gesture in gesture.state == .Began }
    let panEnded = pan.rx_event.filter { gesture in gesture.state == .Ended }

    let pinchStarted = pinch.rx_event.filter { gesture in gesture.state == .Began }
    let pinchEnded = pinch.rx_event.filter { gesture in gesture.state == .Ended }

    // condition: when both pan and pinch has begun
    let bothGesturesStarted = Observable.of(panStarted, pinchStarted).merge(maxConcurrent: 1)
    // condition: when both pan and pinch ended
    let bothGesturesEnded = Observable.of(panEnded, pinchEnded).merge()

    // when bothGesturesStarted, do this:
    bothGesturesStarted.subscribeNext { _ in

        print("started")
    }
}
```

```
override func viewDidLoad() {
    super.viewDidLoad()

    let pan = UIPanGestureRecognizer()
    let pinch = UIPinchGestureRecognizer()

    let panStarted = pan.rx_event.filter { gesture in gesture.state == .Began }
    let panEnded = pan.rx_event.filter { gesture in gesture.state == .Ended }

    let pinchStarted = pinch.rx_event.filter { gesture in gesture.state == .Began }
    let pinchEnded = pinch.rx_event.filter { gesture in gesture.state == .Ended }

    // condition: when both pan and pinch has begun
    let bothGesturesStarted = Observable.of(panStarted, pinchStarted).merge(maxConcurrent: 1)
    // condition: when both pan and pinch ended
    let bothGesturesEnded = Observable.of(panEnded, pinchEnded).merge()

    // when bothGesturesStarted, do this:
    bothGesturesStarted.subscribeNext { _ in

        print("started")
        // create a timer that ticks every second
        let timer = Observable<Int>.timer(repeatEvery: 1)
        let timerThatTicksThree = timer.take(3)
        let timerThatTicksThreeAndStops = timerThatTicksThree.takeUntil(bothGesturesEnded)

    }
}
```



```
override func viewDidLoad() {
    super.viewDidLoad()

    let pan = UIPanGestureRecognizer()
    let pinch = UIPinchGestureRecognizer()

    let panStarted = pan.rx_event.filter { gesture in gesture.state == .Began }
    let panEnded = pan.rx_event.filter { gesture in gesture.state == .Ended }

    let pinchStarted = pinch.rx_event.filter { gesture in gesture.state == .Began }
    let pinchEnded = pinch.rx_event.filter { gesture in gesture.state == .Ended }

    // condition: when both pan and pinch has begun
    let bothGesturesStarted = Observable.of(panStarted, pinchStarted).merge(maxConcurrent: 1)
    // condition: when both pan and pinch ended
    let bothGesturesEnded = Observable.of(panEnded, pinchEnded).merge()

    // when bothGesturesStarted, do this:
    bothGesturesStarted.subscribeNext { _ in

        print("started")
        // create a timer that ticks every second
        let timer = Observable<Int>.timer(repeatEvery: 1)
        let timerThatTicksThree = timer.take(3)
        let timerThatTicksThreeAndStops = timerThatTicksThree.takeUntil(bothGesturesEnded)

        timerThatTicksThreeAndStops.subscribe(onNext: { count in
            print("tick: \(count)")
        }, onCompleted: {
            print("completed")
        })
    }
}
```

or simply just...

```
override func viewDidLoad() {
    super.viewDidLoad()

    let pan = UIPanGestureRecognizer()
    let pinch = UIPinchGestureRecognizer()

    let panStarted = pan.rx_event.filter { $0.state == .Began }
    let panEnded = pan.rx_event.filter { $0.state == .Ended }

    let pinchStarted = pinch.rx_event.filter { $0.state == .Began }
    let pinchEnded = pinch.rx_event.filter { $0.state == .Ended }

    // condition: when both pan and pinch ended
    let bothGesturesEnded = Observable.of(panEnded, pinchEnded).merge()

    // when both pan and pinch has begun, do this:
    Observable.of(panStarted, pinchStarted).merge(maxConcurrent: 1)
        .subscribeNext { _ in

            print("started")
            let timer = Observable<Int>.timer(repeatEvery: 1)
                                .take(3)
                                .takeUntil(bothGesturesEnded)

            timer.subscribe(onNext: { count in
                print("tick: \(count)")
            }, onCompleted: {
                print("completed")
            })
        }
    }
}
```

define **what** “simultaneously panning and rotating” means

define **what** “start a countdown from 3” means

define **what** “when the user stops the gestures” means

define **what** a timer is

now do this:

“When the user starts simultaneously panning and rotating an object, start a countdown from 3.

Stop the timer either when the countdown ends or when the user stops both gestures.”

github.com/
itchingpixels/talks

@itchingpixels