Functional Reactive Programming

Oscar Swanros

MobileCoder.mx

@Swanros

http://swanros.com

I got bored.

Until...

WWDC '14

I'm running away from OOP and the MVC pattern.

And also a little bit of…

```
if ([self requestingApplicationIsValid]) {
    NSLog(@"NAPPSMANAGER :: REQUESTING APP IS VALID.");
    if ([OLSessionsManager sharedManager].isSessionActive) {
        NSLog(@"NAPPSMANAGER :: THERE IS A SESSION ACTIVE.");
        [self verifyCurrentDeviceIsEnrolled:^(BOOL isEnrolled) {
            if (isEnrolled) {
                NSLog(@"NAPPSMANAGER :: DEVICE IS ENROLLED.");
                [self verifyRequestingAppIsInAppInfoListWithBlock:^(BOOL isInAppInfoList, OLApp *app) {
                    if (isInAppInfoList) {
                        NSLog(@"NAPPSMANAGER :: REQUESTING APP IS IN APP INFOLIST.");
                        [self requestSecondaryTokenForApp:app success:^(NSString *token) {
                            if (automaticExecution) {
                                [self sendTokenToRequestingApp:token errorCode:OLNAPPSErrorNone];
                            }else{
                                if (successBlock) {
                                    executeOnMainThread(^{
                                        successBlock(token);
                                    });
                                }
                            }
                        } failure:^(OLNAPPSError error) {
                            if (automaticExecution) {
                                [self sendTokenToRequestingApp:nil errorCode:error];
                            }else{
                                if (failureBlock) {
                                    executeOnMainThread(^{
                                        failureBlock(error);
                                    });
                                }
                            }
                        }];
                    }else{
                        if (automaticExecution) {
                            [self sendTokenToRequestingApp:nil errorCode:OLNAPPSErrorRequestingAppDoesNotHavePermission];
                        }else{
                            if (failureBlock) {
                                executeOnMainThread(^{
                                    failureBlock(OLNAPPSErrorRequestingAppDoesNotHavePermission);
                                });
                            }
                        }
                    }
                }];
            }
        }];
    }else{
        if (failureBlock) {
            executeOnMainThread(^{
                failureBlock(OLNAPPSErrorNoSessionActive);
            });
        }
    }
}
```

```
typedef (^myBlock)(BOOL isThisCool) = ^void() {
   //
};
```

Callback hell, anyone?

Because it gets boring quickly...

Enter **FRP.**

Functional Reactive Programming

=

Functional Programming + Reactive Programming

Functional Programming

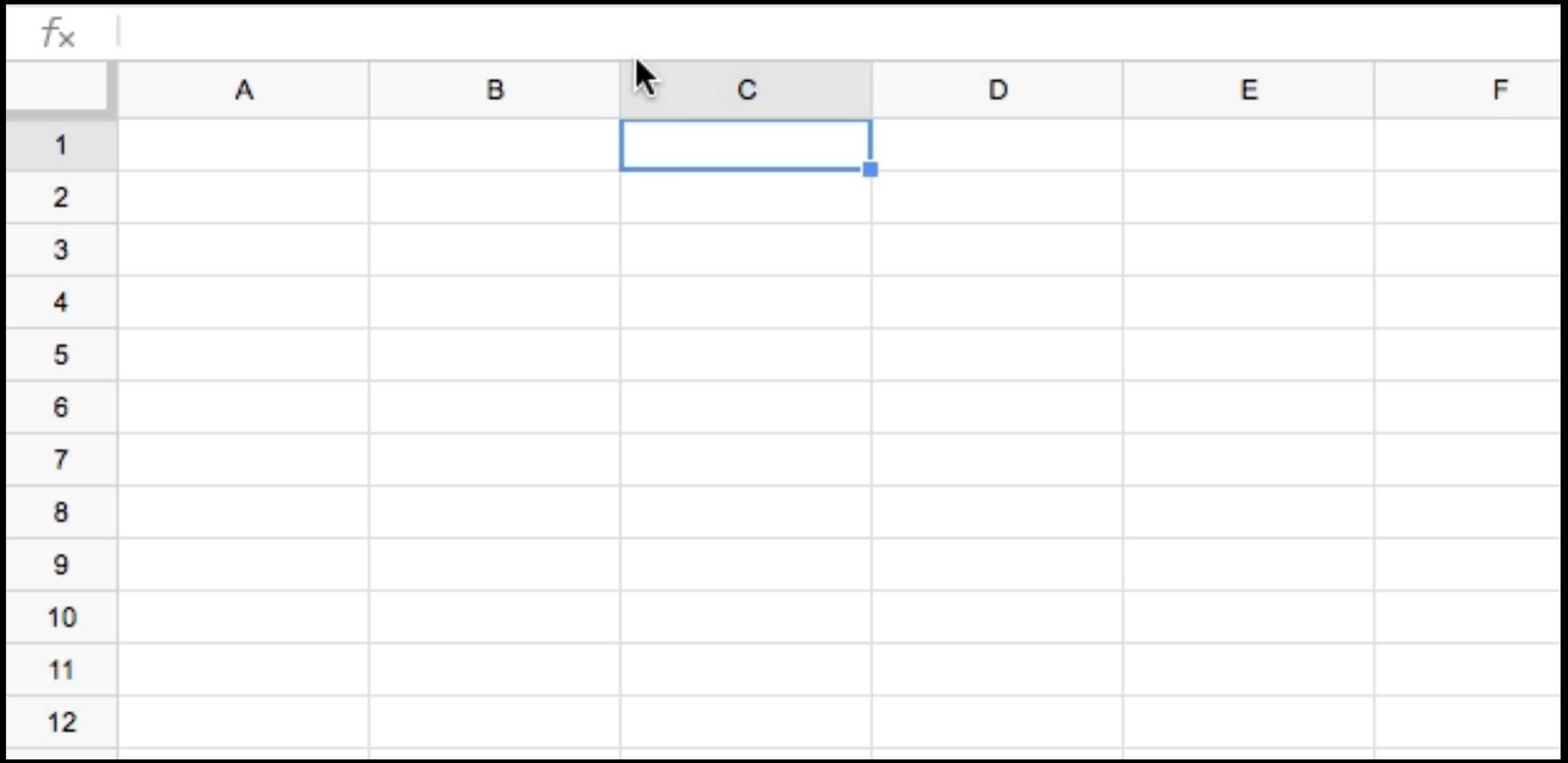
In a *purely functional* language, `f(x)` will return the same value for the same `x`.

Always.

Reactive Programming

In a *reactive* language, `y=f(x)`

y will always stay up to date when x changes.


fx
A
B
C
D
E
F
1
2
3
4
5
6
7
8
9
10
11
12

FRP is about datatypes that represent a *value* over *time*.

At the core of FRP are Signals.

A *Signal* is a **value** that changes **over time**.

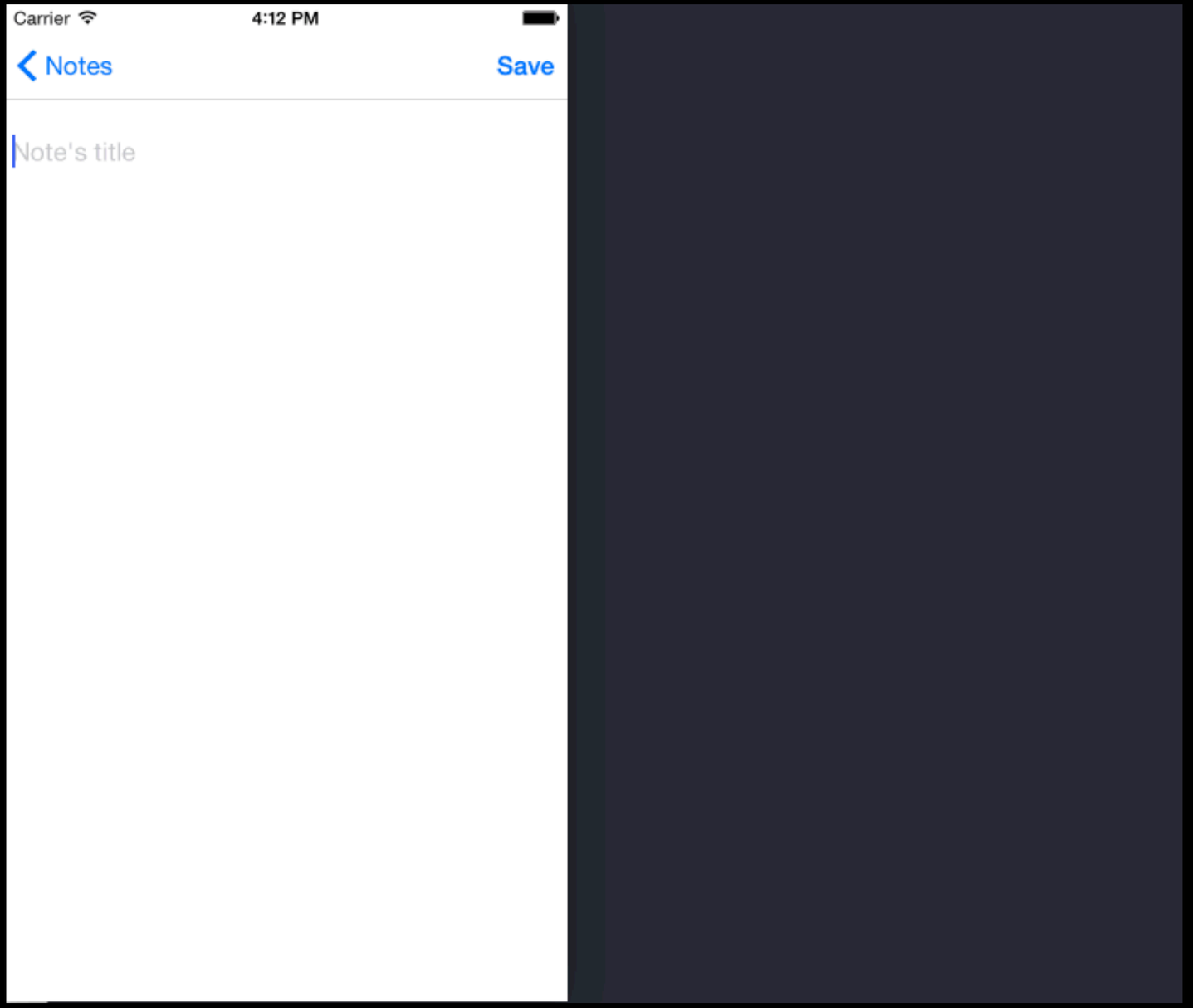
```
done := make(chan bool)

watcher := notificator.New()
for {
  select {
    case newText := <-watcher.Next:
    fmt.Println(newText)

    case error := <-watcher.Error:
    log.Fatal(error)
  }
}

<-done
```

```
[self.emailTextField.rac_textSignal subscribeNext:^(id value) {
  NSLog(@"%@", value);
}];
```


Carrier
4:12 PM
Notes
Save
Note's title

```
@interface ViewController()<UITextFieldDelegate>

// ...

self.emailTextField.delegate = self;

// ...

-(BOOL)textField:(UITextField *)textField
       shouldChangeCharactersInRange:(NSRange)range
       replacementString:(NSString *)string {
          NSLog(@"%@", [textField.text stringByAppendingString:string]);

          return YES;
}
```

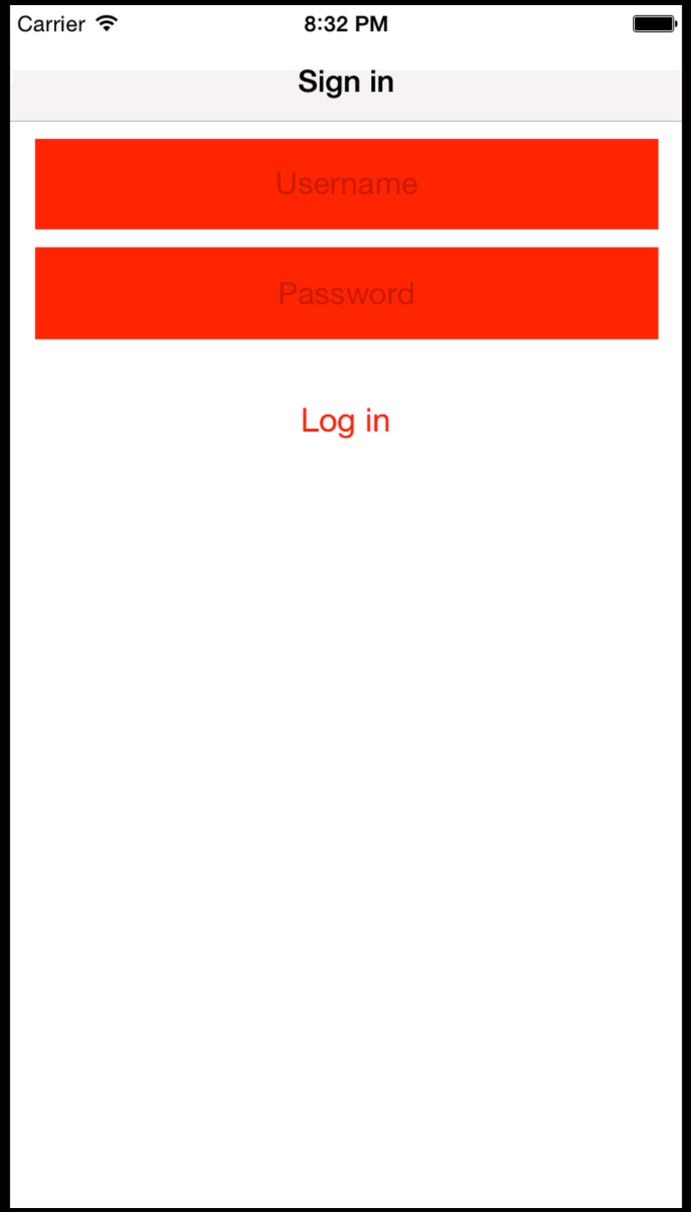
```
[self.emailTextField.rac_textSignal subscribeNext:^(id value) {
  NSLog(@"%@", value);
}];
```

```objc
// 1
RACSignal *usernameTextSignal = self.usernameTextField.rac_textSignal;
RACSignal *passwordTextSignal = self.passwordTextField.rac_textSignal;

// 2
RACSignal *validUsernameSignal = [[usernameTextSignal map:^id(NSString *username) {
    return @([self isValidUsername:username]);
}] distinctUntilChanged];
RACSignal *validPasswordSignal = [[passwordTextSignal map:^id(NSString *password) {
    return @([self isValidPassword:password]);
}] distinctUntilChanged];

// 3
RAC(self.usernameTextField, backgroundColor) = [validUsernameSignal map:^id(NSNumber *usernameValid) {
    return [usernameValid boolValue] ? [UIColor greenColor] : [UIColor redColor];
}];
RAC(self.passwordTextField, backgroundColor) = [validPasswordSignal map:^id(NSNumber *passwordValid) {
    return [passwordValid boolValue] ? [UIColor greenColor] : [UIColor redColor];
}];

// 4
[[RACSignal combineLatest:@[validUsernameSignal, validPasswordSignal]
                  reduce:^id(NSNumber *usernameValid, NSNumber *passwordValid){
                      return @([usernameValid boolValue] == YES && [passwordValid boolValue] == YES);
                  }] subscribeNext:^(NSNumber *loginButtonActive) {
                      self.loginButton.enabled = [loginButtonActive boolValue];
                  }];
```


Carrier
8:32 PM
Sign in
Username
Password
Log in

Functional Reactive Programming lets you be more concise with your code.

Focus on what you *want* to do,

and not on *how* to do it.

(That's the Functional *motto*.)

Resources?

Ruby: frapuccino

```ruby
class Button
  def push
    emit(:pushed)
  end
end

button = Button.new
stream = Frappuccino::Stream.new(button)

counter = stream
            .map {|event| event == :pushed ? 1 : 0 }
            .inject(0) {|sum, n| sum + n }

counter.now # => 0
button.push
counter.now # => 1
```

JS: RxJS

```
var source = getStockData();

source
  .filter(function (quote) {
      return quote.price > 30;
  })
  .map(function (quote) {
      return quote.price;
  })
  .forEach(function (price) {
    console.log('Prices higher than $30: $' + price);
  });
```

Learning FRP is hard.

Rx.Observable.prototype.flatMapLatest(selector, [thisArg])

Projects each element of an observable sequence into a new sequence of observable sequences by incorporating the element's index and then transforms an observable sequence of observable sequences into an observable sequence producing values only from the most recent observable sequence.

But it's worth it.

Questions?