Functional programming

in other languages


Scala


ReactiveX

Scala

Immutability:

```
var myVar = 10
val myVal = "Hello, Scala!"

myVar += 1
myVal += "!!"
```

Multiple assignments:

```
val (myVar1, myVar2) = Pair(40, "Foo")
```

Scala

Anonymous functions:

```
val mul = (x: Int, y: Int) => x*y

val userDir = () => { System.getProperty("user.dir") }
```

Higher-order functions:

```
private def apply(f: Int => String, v: Int) = f(v)
private def layout[A](x: A) = "[" + x.toString + "]"
```

Scala

Anonymous functions:

```
val mul = (x: Int, y: Int) => x*y

val userDir = () => { System.getProperty("user.dir") }
```

Higher-order functions:

```
private def apply(f: Int => String, v: Int) = f(v)
private def layout[A](x: A) = "[" + x.toString + "]"
```

Scala

Pattern matching:

```
private def howMuch(x: Int): String = x match {
  case 1 => "one"
  case 2 => "two"
  case _ => "many"
}
```

Kotlin

Immutability:

```
17      val immutable = 1
18      var mutable = 2
19
20      immutable += 1
21      mutable += 2
```

Error:(20, 5) Kotlin: Val cannot be reassigned

```
mutableListOf(3, 4, 5).add(6)
listOf(1, 2, 3).add(6)
```

immutable lists by default

Kotlin

FUNctions:

```
fun square(x: Int): Int {
    return x * x
}
```

Without body:

```
fun square(x: Int) = x * x
```

Extension functions:

```
fun String.tail() = if (isEmpty()) "" else substring( startIndex: 1)

"Hello, World!".tail()
```

Kotlin

Lambdas:

```
val greet = { name: String -> println("Hi, $name") }
greet( p1: "Bob")
```

Maps & filters:

implicit name of a single parameter

```
users.filter { it.firstName.startsWith("J") }
        .map { "${it.firstName} ${it.lastName}" }
        .forEach { it -> greet(it) }
```

```
class User(
        val firstName: String,
        val lastName: String)

val users = listOf(
        User("John", "Smith"),
        User("Ann", "Bush"),
        User("James", "May"))
```

Python

Defining a function:

```
def func(x, y):
    return x**2 + y**2
```

Defining a function with lambda:

```
square_sum = lambda x, y: x**2 + y**2
```

Python

Calling a lambda with argument:

```
(lambda x: x+2) (5)
```

Calling a function:

```
 : func (2, 3)
=> 13
 : square_sum (3, 4)
=> 25
 : (lambda x: x+2) (5)
=> 7
```

Python

List comprehension:

```
l = [x**2 for x in range(1,20)]
```

```
 : l
=> [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169,
196, 225, 256, 289, 324, 361]
```

```
m = [x*y for x in range(1,10) for y in range(1,10)]
```

```
 : m
=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 2, 4, 6, 8, 10, 12, 14,
16, 18, 3, 6, 9, 12, 15, 18, 21, 24, 27, 4, 8, 12, 16,
20, 24, 28, 32, 36, 5, 10, 15, 20, 25, 30, 35, 40, 45,
6, 12, 18, 24, 30, 36, 42, 48, 54, 7, 14, 21, 28, 35,
42, 49, 56, 63, 8, 16, 24, 32, 40, 48, 56, 64, 72, 9,
18, 27, 36, 45, 54, 63, 72, 81]
```

Python

Maps:

```
list1 = [1, 2, 3, 4, 5]
list2 = [-1, 1, -5, 4, 6]
map(lambda x, y: x*y, list1, list2)
```

```
: map(lambda x, y: x*y, list1, list2)
=> [-1, 2, -15, 16, 30]
```

```
names = ["Bob", "John", "Jack"]
greetings = ["Hello", "Hi", "Aloha"]
map(lambda name, greet: greet + ", " + name, names, greetings)
```

```
: map(lambda name, greet: greet + ", " + name, names, greetings)
=> ['Hello, Bob', 'Hi, John', 'Aloha, Jack']
```

Python

Filters:

```
numbers = [10, 4, 2, -1, 6]
filter(lambda x: x < 5, numbers)
```

```
: filter(lambda x: x < 5, numbers)
=> [1, 2, 3, 4]
```

```
names = ["Bob", "John", "Jack"]
filter(lambda name: name.startswith("J"), names)
```

```
: filter(lambda name: name.startswith("J"), names)
=> ['John', 'Jack']
```

Python

Reduce:

```
from functools import reduce

numbers = [1, 2, 3, 4]
reduce(lambda res, x: res * x, numbers, 1)
```

accumulator

ReactiveX

ReactiveX is a library for composing asynchronous and event-based programs by using observable sequences.

Languages

- Java: RxJava
- JavaScript: RxJS
- C#: Rx.NET
- C#(Unity): UniRx
- Scala: RxScala
- Clojure: RxClojure
- C++: RxCpp
- Lua: RxLua
- Ruby: Rx.rb
- Python: RxPY
- Go: RxGo
- Groovy: RxGroovy
- JRuby: RxJRuby
- Kotlin: RxKotlin
- Swift: RxSwift
- PHP: RxPHP
- Elixir: reaxive
- Dart: RxDart

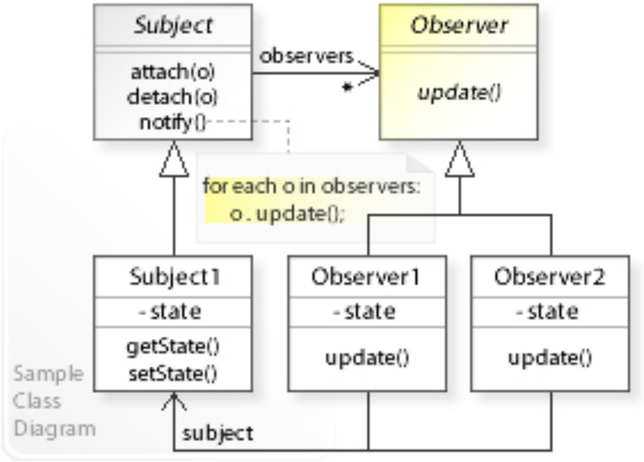
ReactiveX for platforms and frameworks

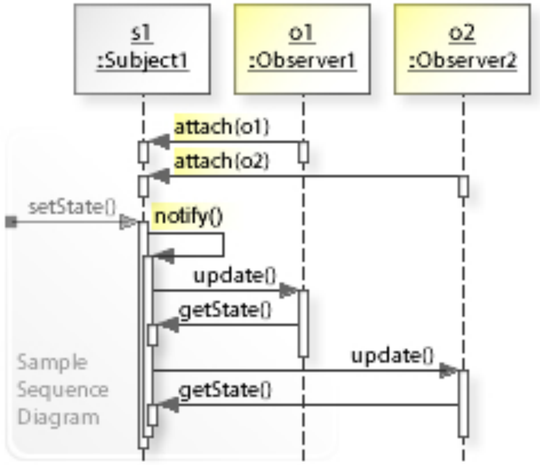
- RxNetty
- RxAndroid
- RxCocoa

http://reactivex.io

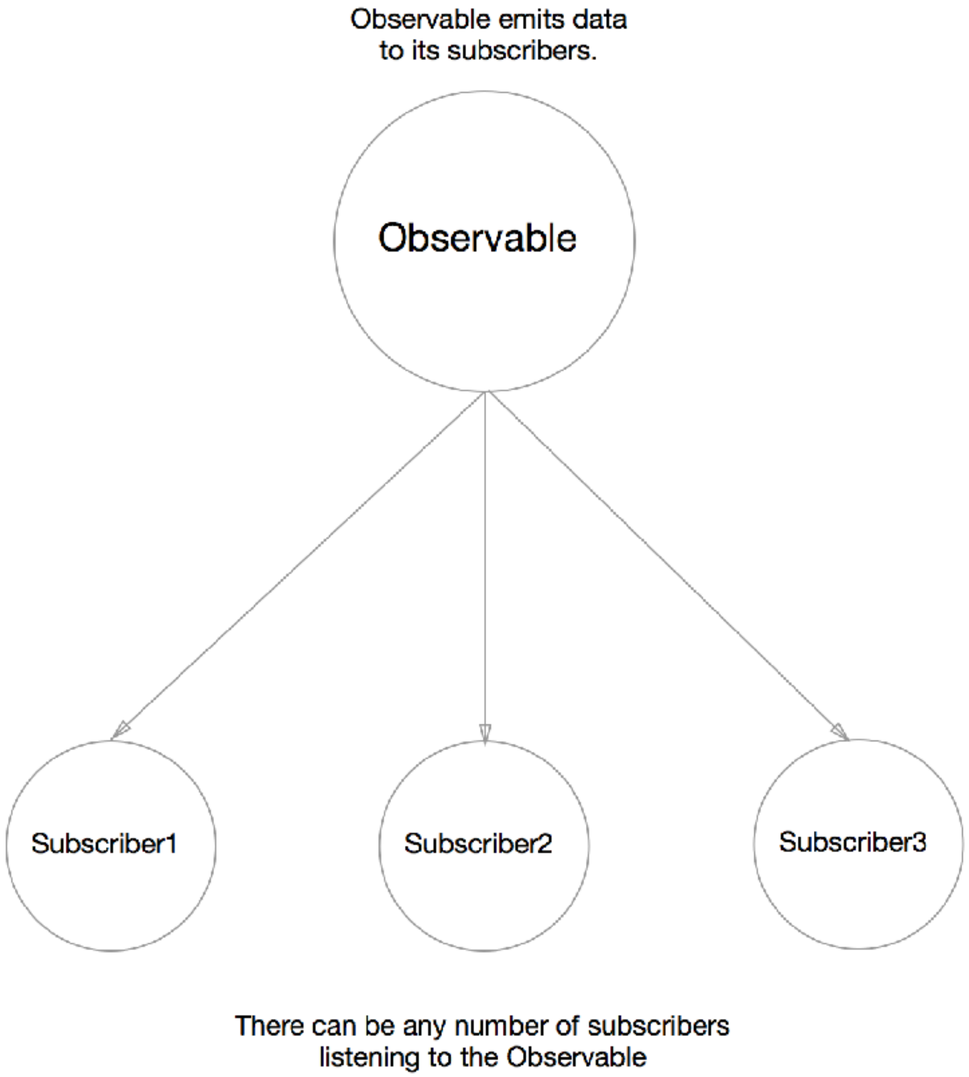
ReactiveX

Implements Observer pattern with some additions to it



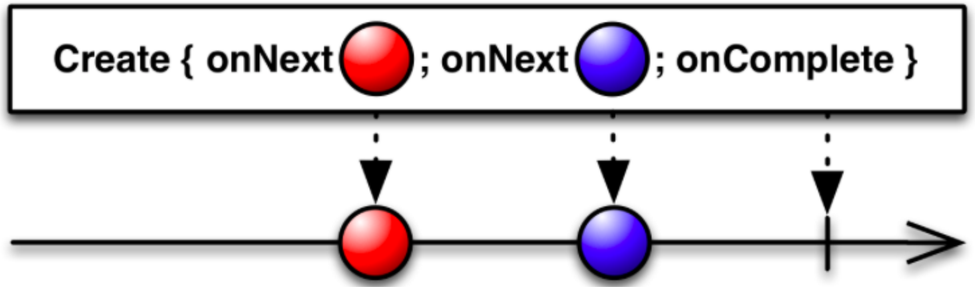


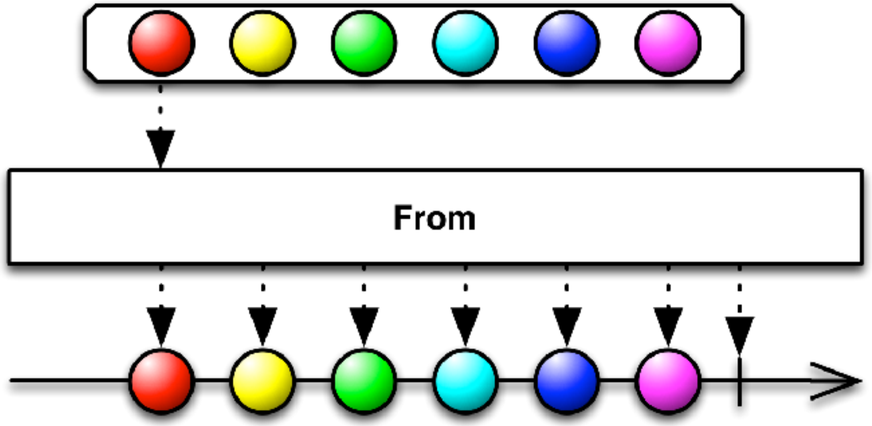
ReactiveX

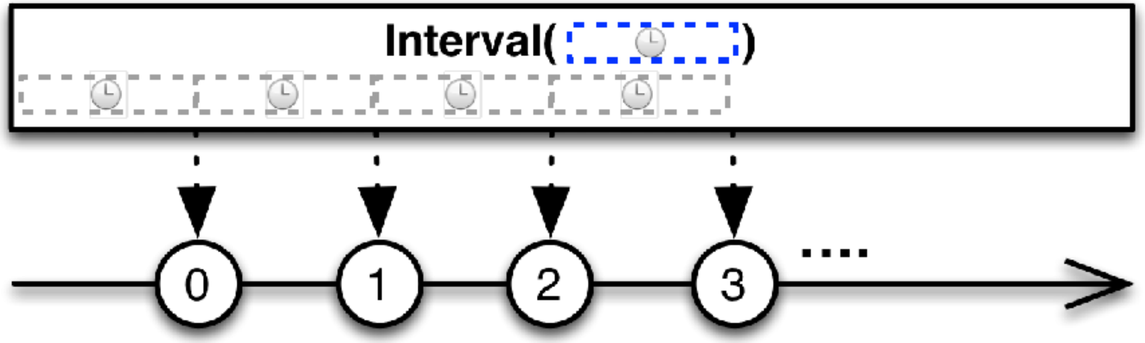

Observable emits data
to its subscribers.
Observable
Subscriber1
Subscriber2
Subscriber3
There can be any number of subscribers
listening to the Observable

ReactiveX

Creating observable

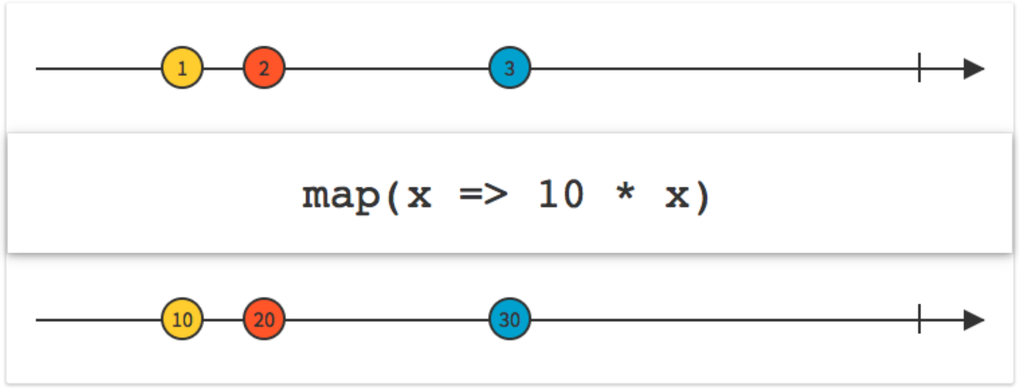


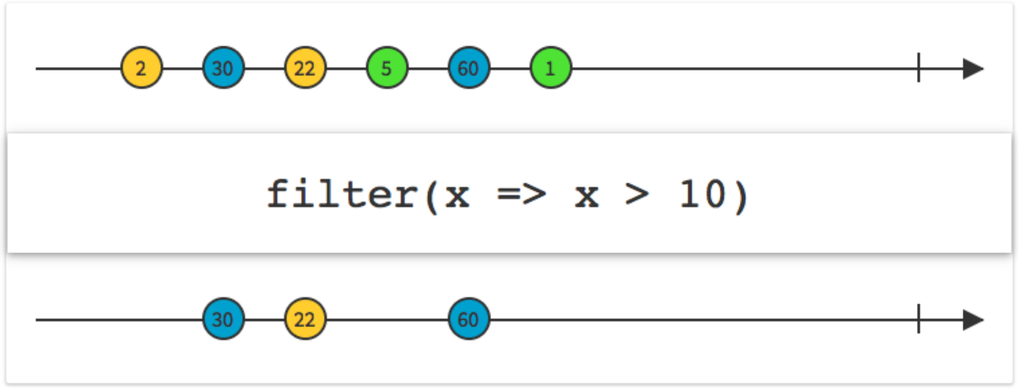


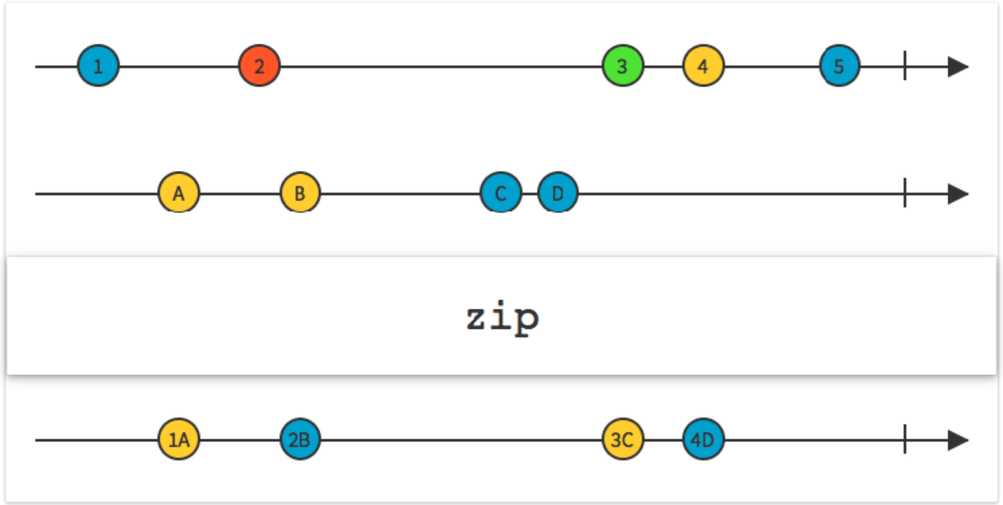


ReactiveX

Transforming observables







Back to Haskell

Pandoc a universal document converter

Pandoc is a Haskell library for converting from one markup format to another, and a command-line tool that uses this library.

Usage example:

pandoc README.md -o readme.docx

```
# Functional programming course
Learning materials for a course on functional programming using Haskell

## Slides
0. [Introduction to course](lectures/0 intro to fun.pdf)
0. [Basic syntax and function calls](lectures/1_basics_and_functions.pdf)
0. [Working with lists](lectures/2 working with lists.pdf)
0. [Recursion](lectures/3_recursion.pdf)
0. [Higher order functions](lectures/4_higher_order_functions.pdf)
0. [Own data types](lectures/5 making own data types.pdf)
0. [Input/Output](lectures/6_input_output.pdf)
```

Functional programming course

Learning materials for a course on functional programming using Haskell

Slides

0. Introduction to course
1. Basic syntax and function calls
2. Working with lists
3. Recursion
4. Higher order functions
5. Own data types
6. Input/Output

http://pandoc.org **https://github.com/jgm/pandoc**