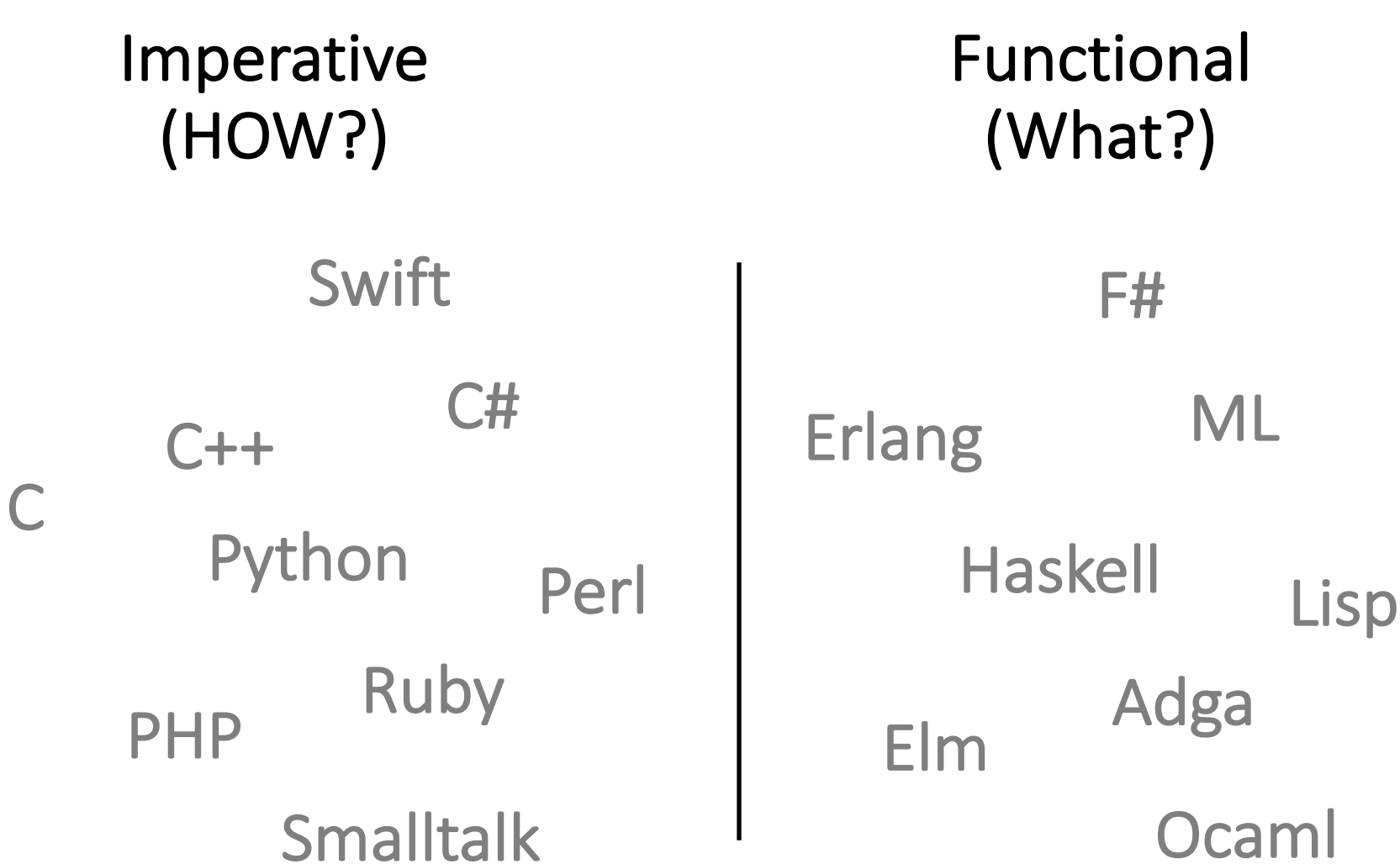
Introduction to Functional Programming with Haskell

Mohammed Al Rujayi
Works @ Wasphi
Twitter, Github: @mohalrej

What is Functional Programming?


Imperative
(HOW?)
Functional
(What?)
Swift
C#
C++
C
Python
Perl
Ruby
PHP
Smalltalk
F#
Erlang
ML
Haskell
Lisp
Adga
Elm
Ocaml

Functional Programming has the following…

- High order functions, functions that take other functions
- Pure functions, no side effects
- No mutations
- Recursion not loops

It’s just like Legos!

An example

The factorial functions

$5! = 5 * 4 * 3 * 2 * 1$

$3! = 3 * 2 * 1$

Factorial in Python…

```
def factorial(n):
    result = 1
    while n >= 1:                  # loop
        result = result * n # mutation
        print (result)             # side effects
        n = n – 1                  # mutation
    return result
```

Factorial in Haskell…

```
Factorial 0 = 1
Factorial 1 = 1
Factorial n = n * Factorial (n-1)
```

Haskell demo

We’ll cover

- Lists
- Application
- Lambda functions
- Maps, folds, filters
- Pattern matching
- Composition

Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13 …

Starting with 0, and 1

Fib (n) = Fib (n-1) + Fib(n-2)

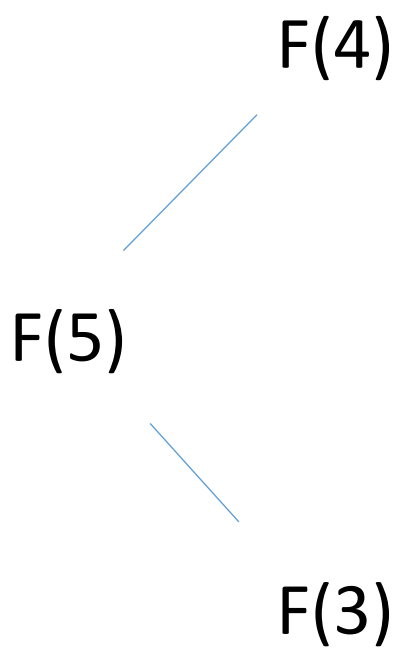
Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13 …

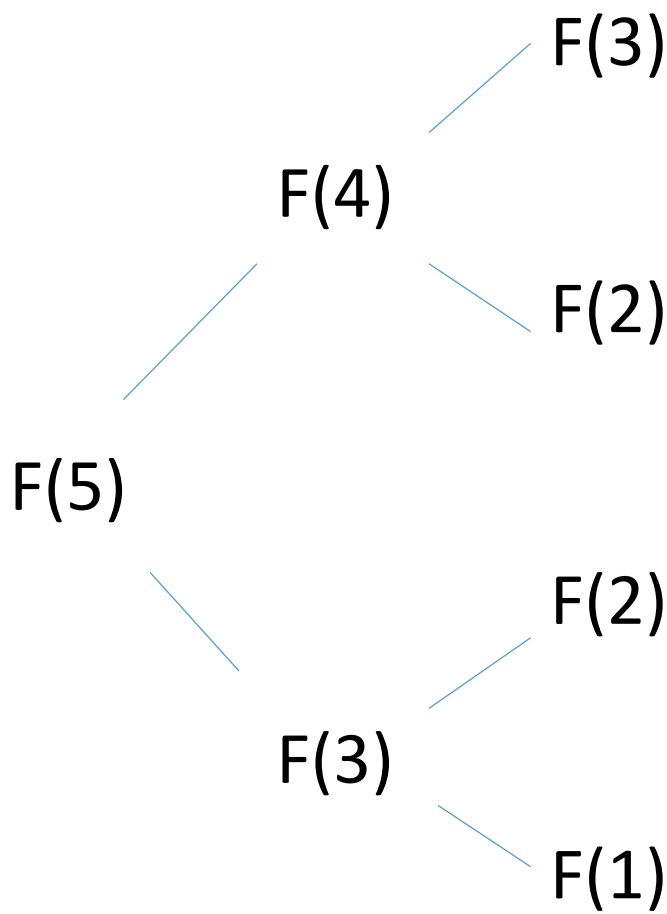
Starting with 0, and 1

Fib (n) = Fib (n-1) + Fib(n-2)

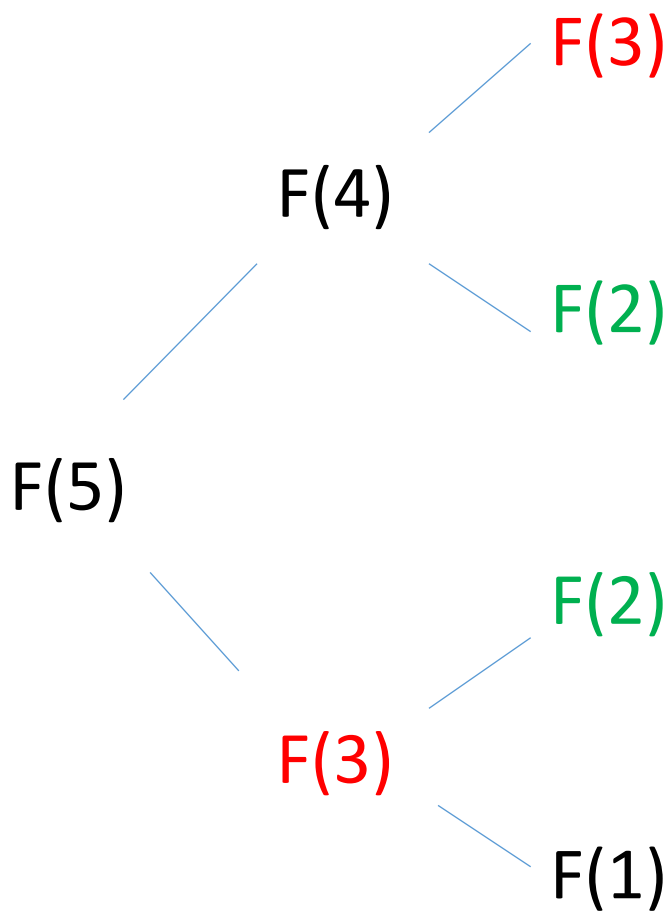
Not a good definition!



Not a good definition!



Not a good definition!



A better idea

$$(x, y) \to (x + y, x)$$

A better idea

(x, y) -> (x + y, x)

At (0, 1):

(0, 1) -> (1, 0)

(1, 0) -> (1, 1)

(1, 1) -> (2, 1)

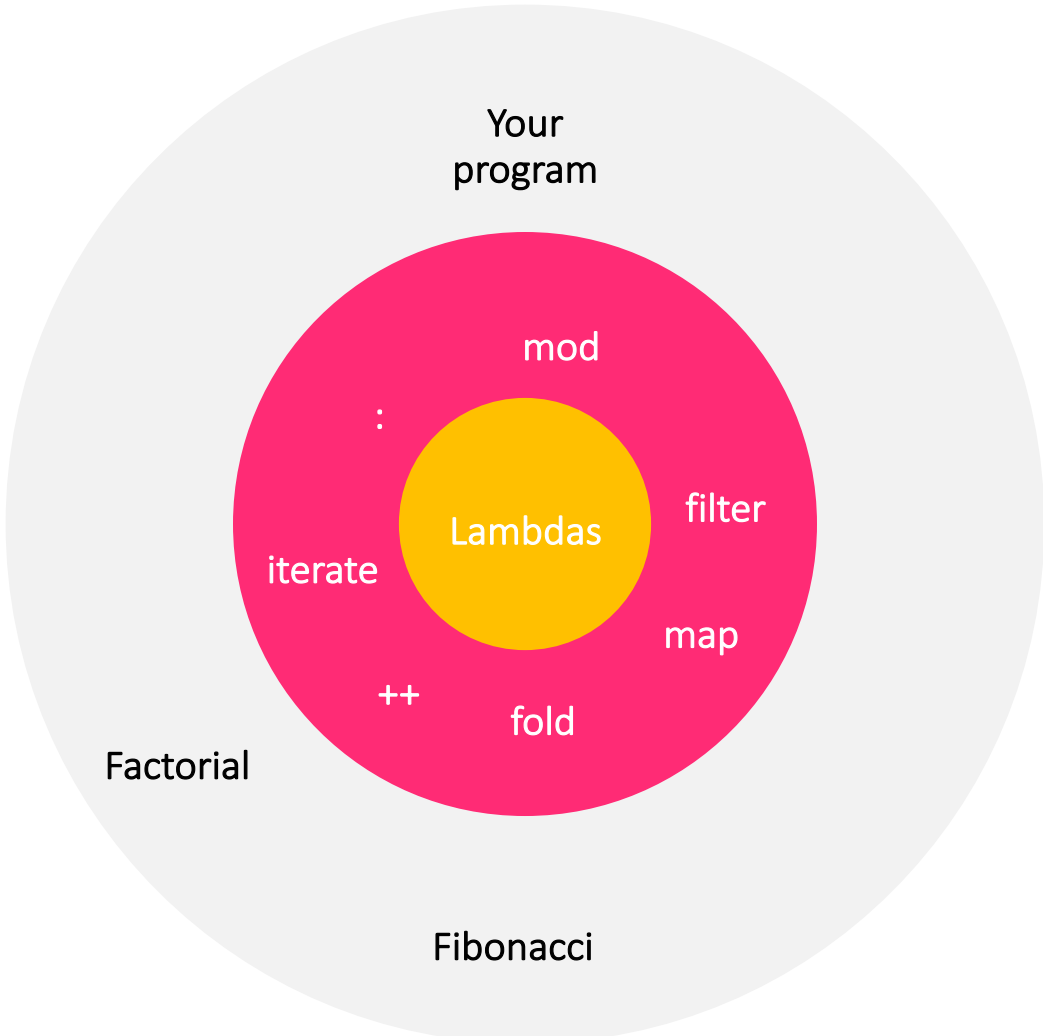
(2, 1) -> (3, 2)

…

All standard functions are implemented in Haskell!!

```
map f [] = []
map f (x:xs) = (f x) : map f xs
```

The core of Haskell



Easiest way to get Haskell

- Haskell platform

  https://www.haskell.org/platform/

- Haskell for mac

  http://haskellformac.com

Appendix: Demo code

```
module Ksu where

-- First factorial, implemented with recursion
fac 0 = 1
fac 1 = 1
fac n = n * fac(n-1)

-- Composition of two functions
ksu_01 = map (\ x-> x^2) . map (\ x -> x^2)

-- Sum implemented in terms of fold
sum_01 list = foldl (\ x y -> x + y) 0 list

-- The fib function implemented with recursion
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)

-- The fib function implemented in terms of iterate
fib_01 n = take n $ iterate (\(x, y) -> (x+y, x)) (0, 1)
fib_02 = last . map (\(x, y) -> y) . fib_01
```

يعطيكم العافية