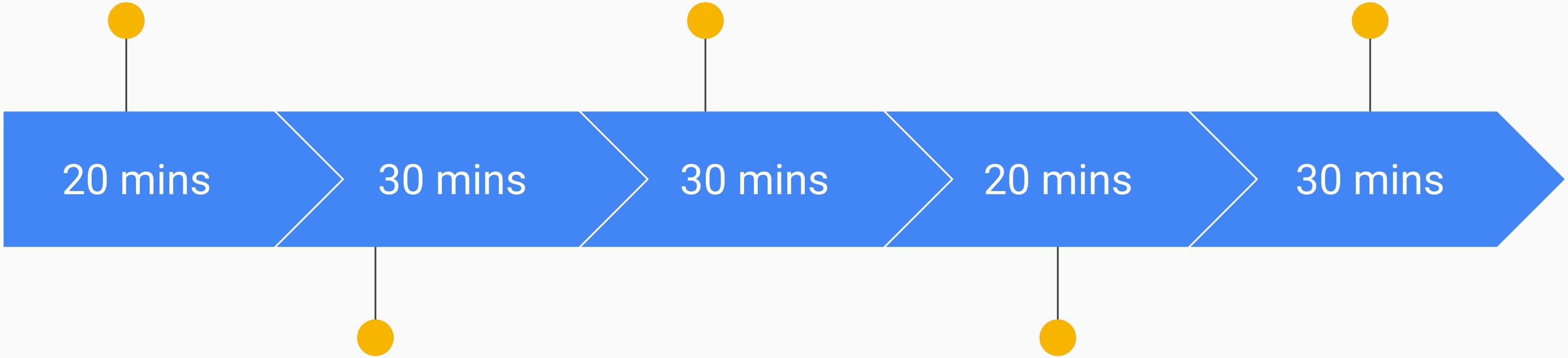
Little Introduction To Functional Programming & WCF Example in F#

Audience

- UP Team Members
- UP Management
- Other Folks Who Wants to Learn

Objective

- An overview of basic concepts of Functional Programming Paradigm
- Inspire your curiosity about the Functional Programming Approach
- Understand the benefits, where use it
- Little Introduction to the F# language
- Explanation of the UP Login Service



Why use a functional programming language?

20 mins

History of Functional Programming

30 mins

Railway Oriented Programming Distilled

30 mins

Questions and Answers

20 mins

Live explanation of UP Login Service

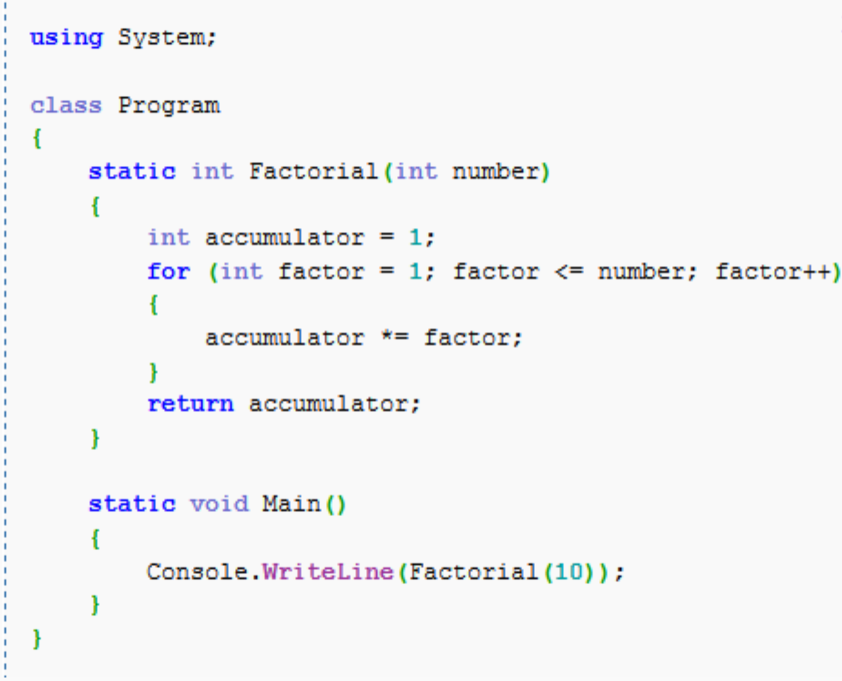
30 mins

Why Functional Programing

Why Functional Programming - Key benefits of F# compared with C#

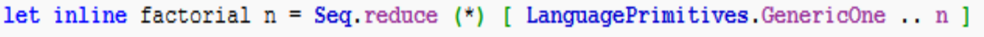
- **Conciseness:** No Brackets (use indentation instead), No semicolons, Type Inference and **LESS CODE TO WRITE!!** (not as haskell but a great deal anyway)

C#



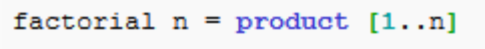
```
using System;

class Program
{
    static int Factorial(int number)
    {
        int accumulator = 1;
        for (int factor = 1; factor <= number; factor++)
        {
            accumulator *= factor;
        }
        return accumulator;
    }

    static void Main()
    {
        Console.WriteLine(Factorial(10));
    }
}
```

F#



```
let inline factorial n = Seq.reduce (*) [ LanguagePrimitives.GenericOne .. n ]
```

Haskell



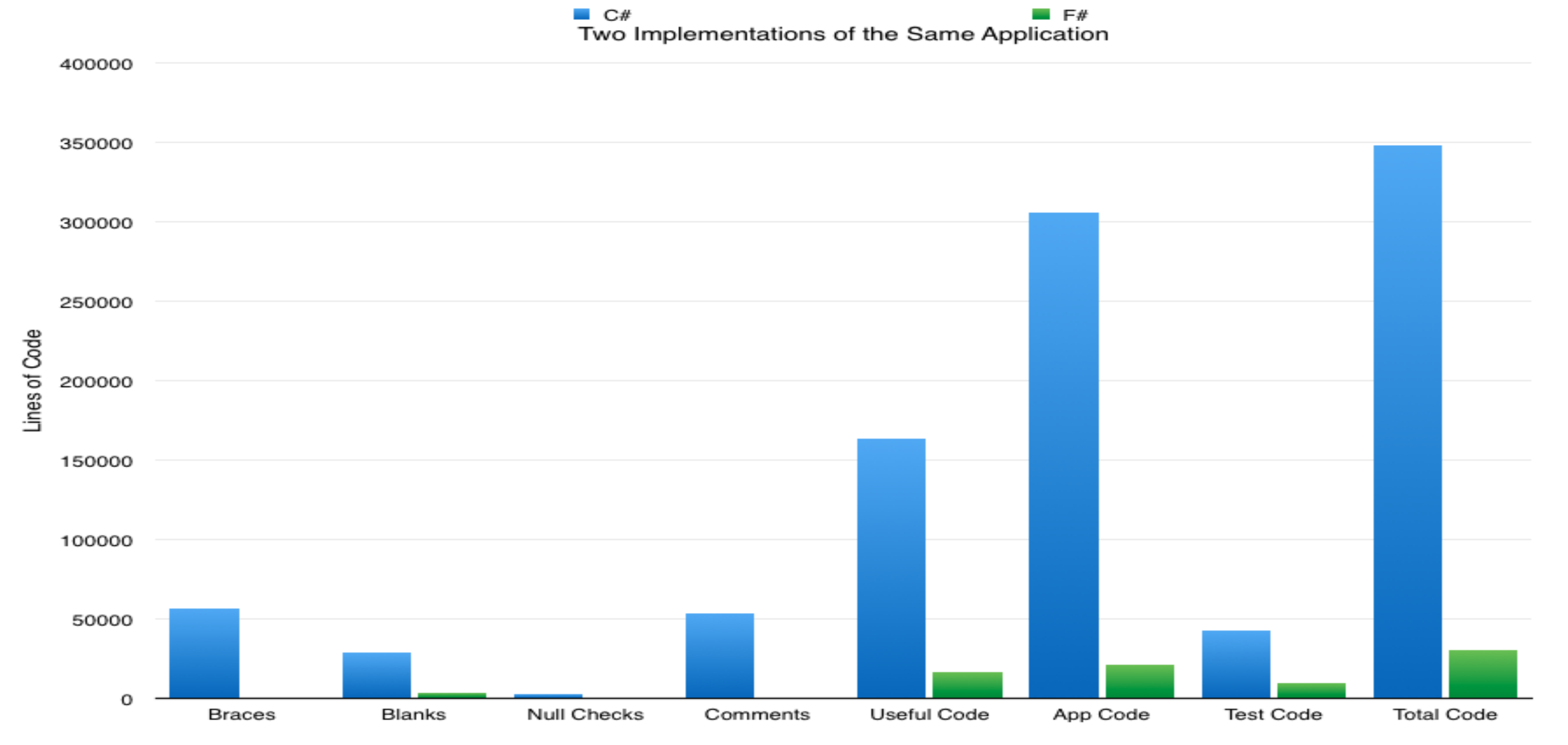
```
factorial n = product [1..n]
```

Have you see some types here? Semicolon? Brackets? What's 'static'? :)

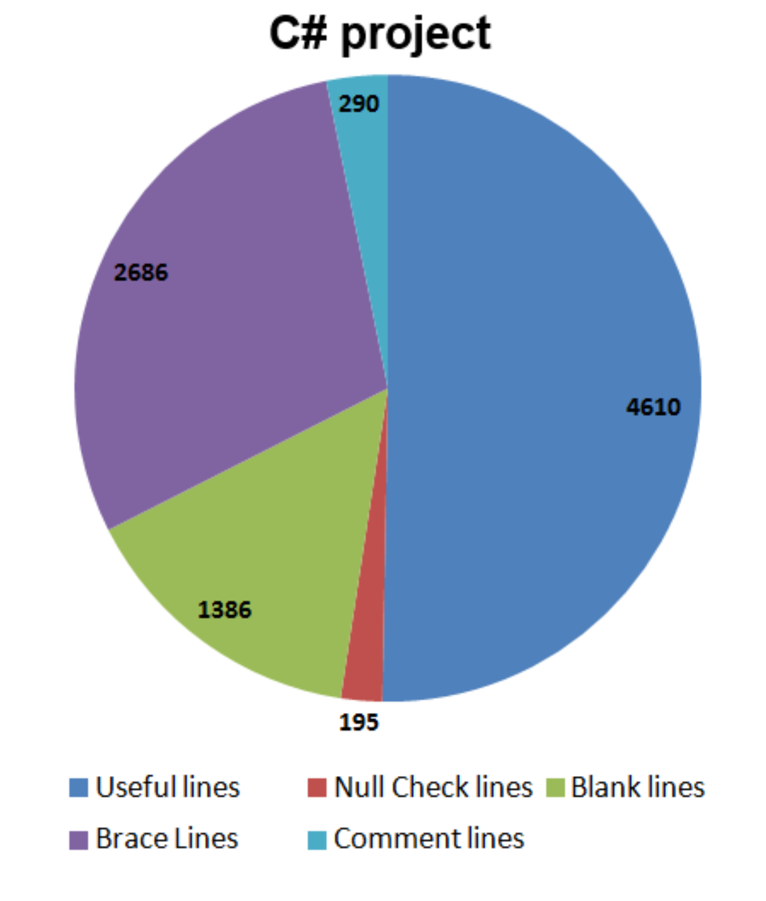
References:

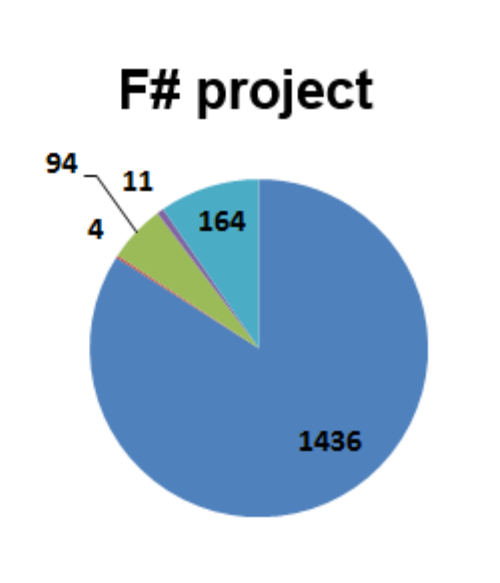
- http://rosettacode.org/wiki/Factorial
- http://rosettacode.org/wiki/N-queens_problem
- http://fsharpforfunandprofit.com/posts/fvsc-sum-of-squares/

Why Functional Programming - Key benefits of F# compared with C#



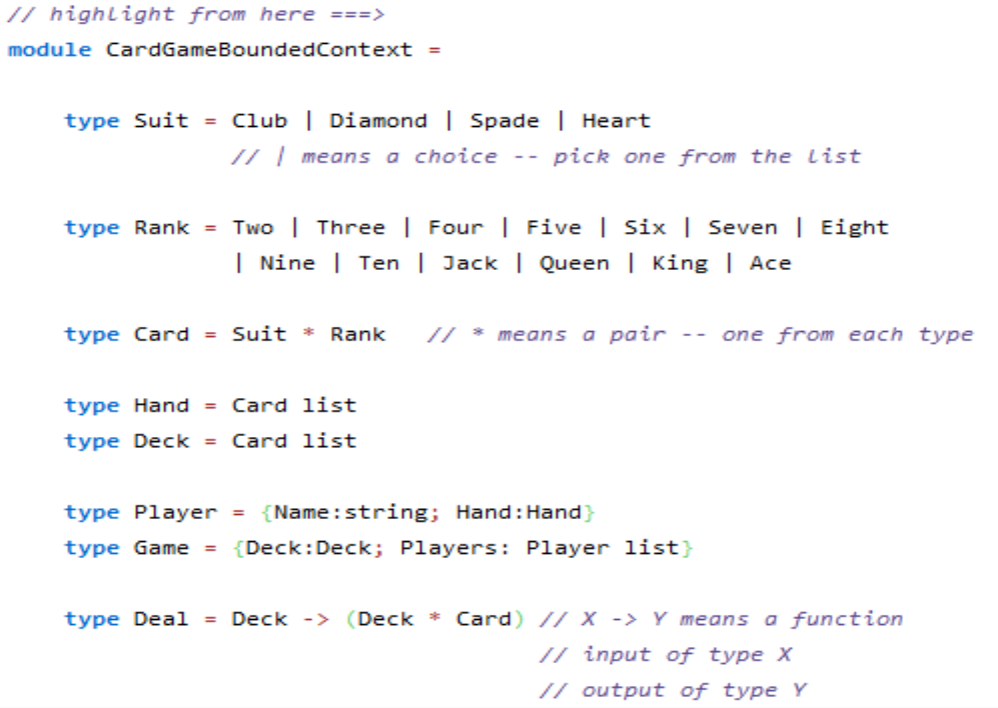
Why Functional Programming - Key benefits of F# compared with C#





Why Functional Programming - Key benefits of F# compared with C#

- **Convenience:** Many common tasks are much simpler in F# (creating and using complex type definitions, doing list processing, comparison and equality, state machines…) and function as first class citizen, let the composition and reuse of code easier.



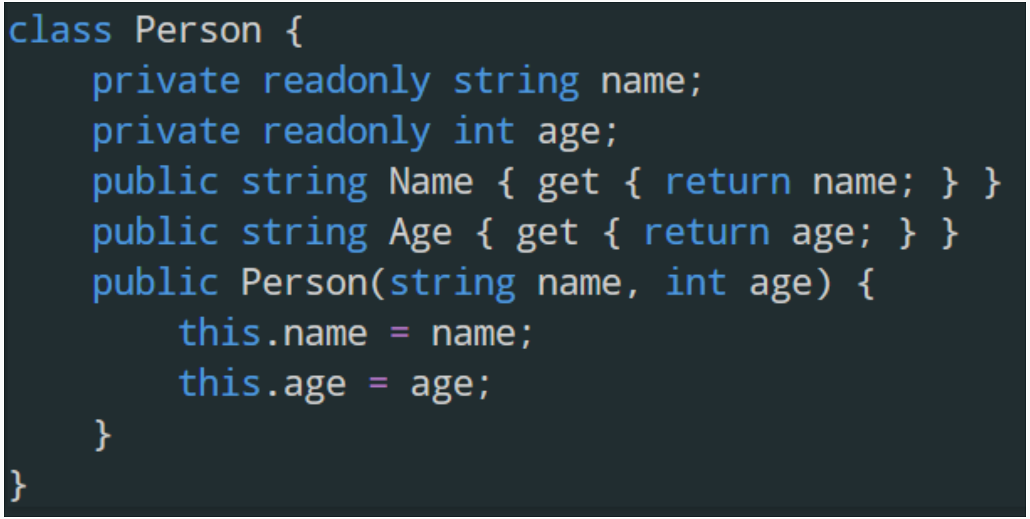
```
// highlight from here ===>
module CardGameBoundedContext =

    type Suit = Club | Diamond | Spade | Heart
                // | means a choice -- pick one from the list

    type Rank = Two | Three | Four | Five | Six | Seven | Eight
                | Nine | Ten | Jack | Queen | King | Ace

    type Card = Suit * Rank   // * means a pair -- one from each type

    type Hand = Card list
    type Deck = Card list

    type Player = {Name:string; Hand:Hand}
    type Game = {Deck:Deck; Players: Player list}

    type Deal = Deck -> (Deck * Card) // X -> Y means a function
                                      // input of type X
                                      // output of type Y
```

How much classes do you need to build a domain driven design like this one?

References:

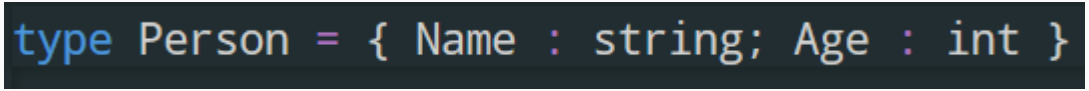
- http://fsharpforfunandprofit.com/series/understanding-fsharp-types.html
- http://fsharpforfunandprofit.com/ddd/

Why Functional Programming - Key benefits of F# compared with C#



```
class Person {
    private readonly string name;
    private readonly int age;
    public string Name { get { return name; } }
    public string Age { get { return age; } }
    public Person(string name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

VS



```
type Person = { Name : string; Age : int }
```

Why Functional Programming - Key benefits of F# compared with C#

- **Correctness:**
  - Less occurrence of NullReferenceException and other runtime problems in general.
  - Business Logic through types
  - **If it build, it almost works**
  - **Less Code -> Fewer bugs**
- **Concurrency:**
  - built-in tools and libraries: Actors, Async, Reactive and Multitasking
  - Immutability means thread safe
- **Completeness:** It can do almost all that C# can do. Full support to .NET Framework
- **Testable:** more testable thanks to the separation between pure functional computation and side effects*. You can also run single functions in "standalone mode" in a REPL.
- **More Readable:** See Conciseness. Self documenting code.
- **Can you use it as a scripting language!**
- **Change Your Mind:** help you to approach problems in different way
- **Most of others programming languages added functional concepts:** Java 8(lambdas, streams,..), C# (delegates, Func<>, lazy,...)
- **Performance:** thanks to the lazy evaluation.* possibility to handle infinite data structures.
- **Static Typing:** the compiler become a real friend!
- **Type Inference:** Forgot about Generics and to add functions type informations, the compiler will do it for you

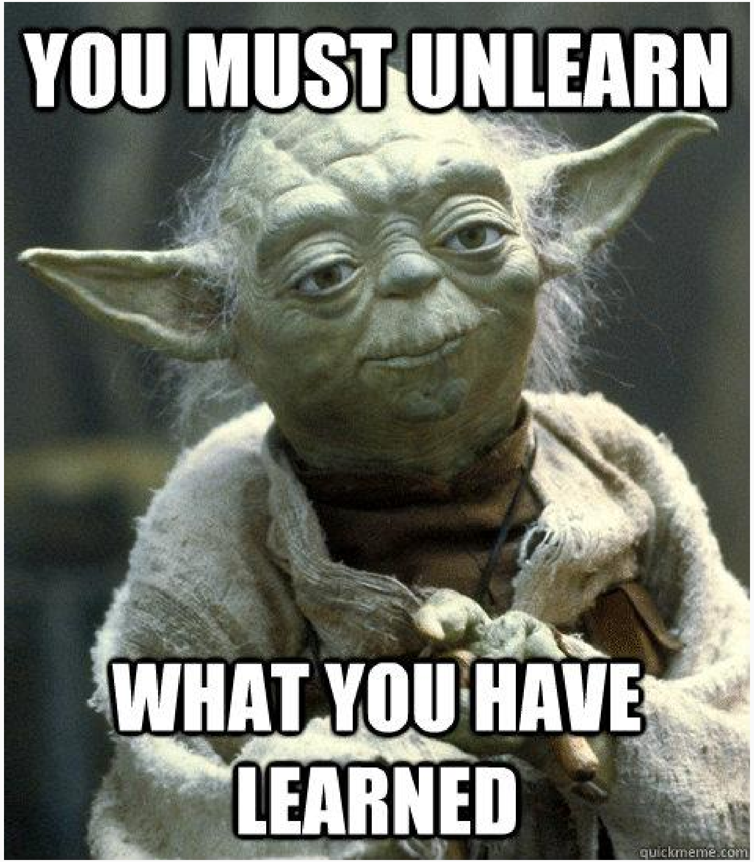
Why Functional Programming - Disadvantages

- **Higher Learning Curve**
- **Changing your mind need effort**
- **F# is not a pure functional programming language:**(better learn haskell)
  - you still can do shitty OOP code, but require specific keywords (mutable, new, ...)
  - you don’t have lazy evaluation by default
  - you can do side effects for free (es. println)
- **Performance:** some imperative implementation perform better, but they are for specific cases.
- **it is very difficult to predict the time and space costs of evaluating a *lazy* functional program.**
- **Less IDE Support and Templates**
- **Less Examples over the Internet**

FP - Basic Concepts

Down to the rabbit hole

Functional Programming - Basic Concepts

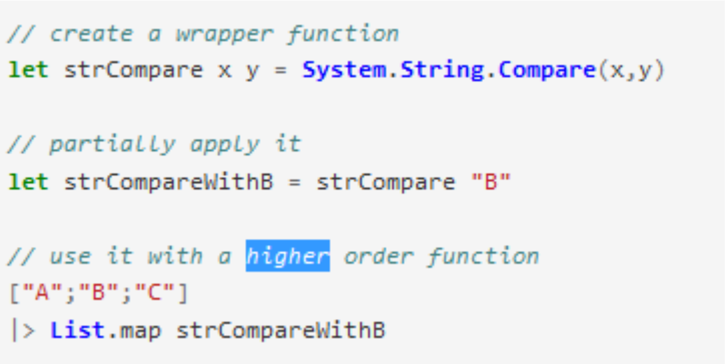


Functional Programming - Basic Concepts

Functions Everywhere!!! You know them from math

- **Functions are first class citizen:**
  - you can call it
  - you can pass it to another function, and so have function as parameters (JS?)
  - you can return a function

If a function have one of the last 2 properties you can call it an **Higher Order Function**



```
// create a wrapper function
let strCompare x y = System.String.Compare(x,y)

// partially apply it
let strCompareWithB = strCompare "B"

// use it with a higher order function
["A";"B";"C"]
|> List.map strCompareWithB
```

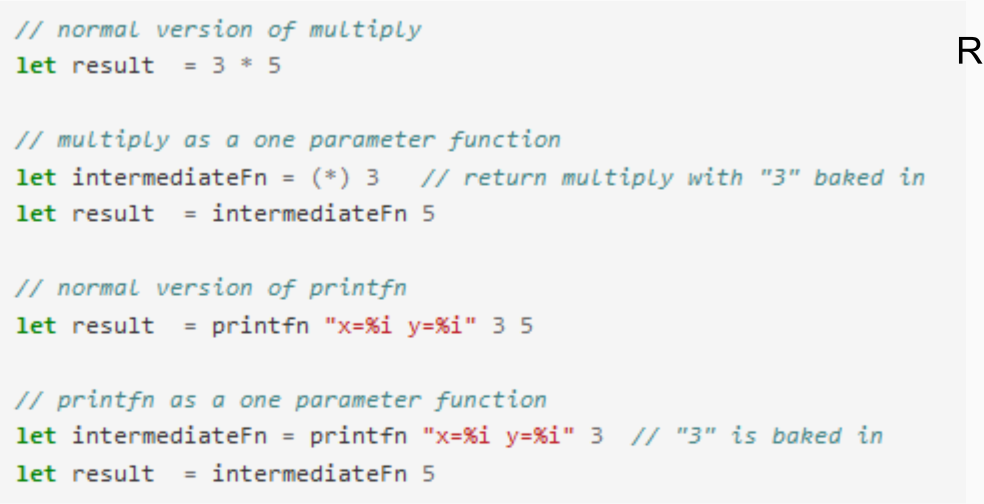
Example: the strCompareWithB return a function with 1 parameter, and in the last line this is called for every element of the list
Did you remember the MAP function from the Zoffoli’s talk?
Can you see it?

Here’s also an example of currying! (wait for it :))

Functional Programming - Basic Concepts

Currying and Partial Application

You can call(apply) a function even if you don’t have all the parameters available.
So what will you get back? GUESS WHAT? FUNCTION! with one less parameter.
currying is the technique of translating the evaluation of a function that takes multiple arguments (or a tuple of arguments) into evaluating a sequence of functions, each with a single argument (partial application)



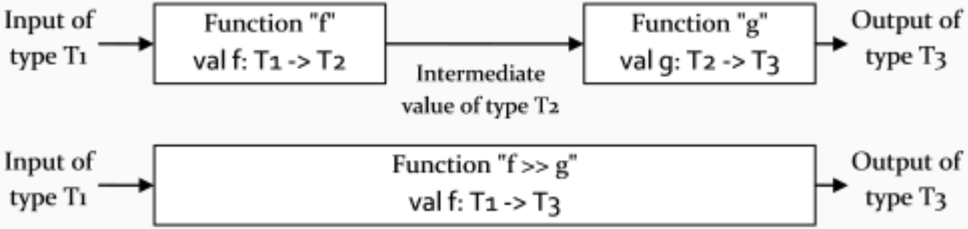
```
// normal version of multiply
let result  = 3 * 5

// multiply as a one parameter function
let intermediateFn = (*) 3   // return multiply with "3" baked in
let result  = intermediateFn 5

// normal version of printfn
let result  = printfn "x=%i y=%i" 3 5

// printfn as a one parameter function
let intermediateFn = printfn "x=%i y=%i" 3  // "3" is baked in
let result  = intermediateFn 5
```

References:

- https://en.wikipedia.org/wiki/Currying
- http://fsharpforfunandprofit.com/posts/currying/
- http://fsharpforfunandprofit.com/posts/partial-application/

Functional Programming - Basic Concepts

PLEASE!!! PLEASE STOP COPY&PASTE CODE(also in OOP) **REUSE YOUR CODE!!**
IN PARTICULAR FUNCTIONS, WITH COMPOSITION!



Input of type T1

Function "f"
val f: T1 -> T2

Intermediate value of type T2

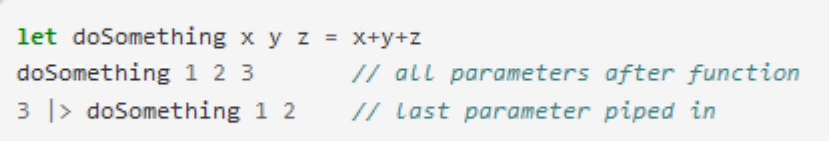
Function "g"
val g: T2 -> T3

Output of type T3

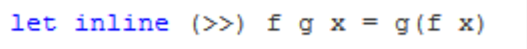
Input of type T1

Function "f >> g"
val f: T1 -> T3

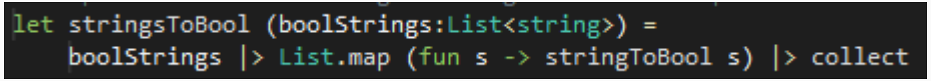
Output of type T3



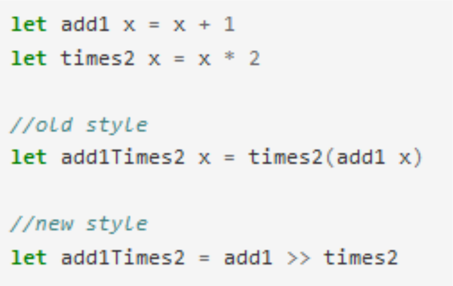
```
let doSomething x y z = x+y+z
doSomething 1 2 3        // all parameters after function
3 |> doSomething 1 2     // last parameter piped in
```



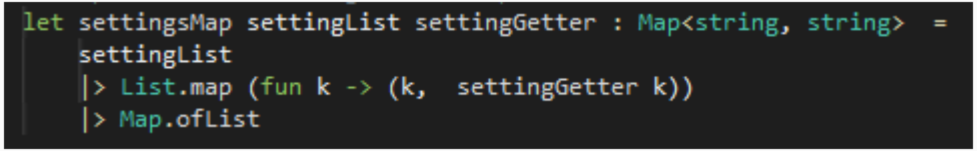
```
let inline (>>) f g x = g(f x)
```



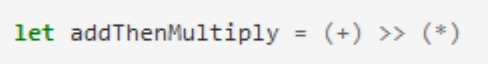
```
let stringsToBool (boolStrings:List<string>) =
    boolStrings |> List.map (fun s -> stringToBool s) |> collect
```



```
let add1 x = x + 1
let times2 x = x * 2

//old style
let add1Times2 x = times2(add1 x)

//new style
let add1Times2 = add1 >> times2
```



```
let settingsMap settingList settingGetter : Map<string, string>  =
    settingList
    |> List.map (fun k -> (k,  settingGetter k))
    |> Map.ofList
```



```
let addThenMultiply = (+) >> (*)
```

And here there’re some lambdas
fun k -> ….

Functional Programming - Basic Concepts

The Importance to be PURE!

your functions won’t modify anything in the outside world

Applying a function with the same parameters always returns the same value.

Take the previous sentence as a rule for your programming, try to separate your side effects from your pure functions!

This is true in a pure fundamentalistic functional programming way. Unfortunately F# is a multiparadigm language so you can do object and side effects inside a function, not like Haskell. This leads to more **problems** instead of a pure approach, but still far better than OOP.

Keep this advice also in your daily programming!!

Functional Programming - Basic Concepts

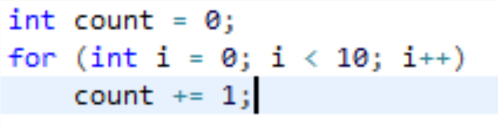
IMMUTABILITY - ONLY VALUES AND EXPRESSIONS

What’s this mean?..take a breath...sit down...

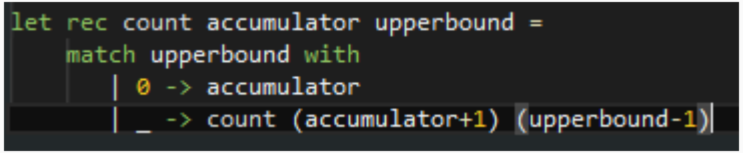
- **NO VARIABLES**
- **NO ASSIGNMENTS**
- **NO LOOPS**

DON’T PANIC!! you can still manage all in a functional way:

- Instead of loops use recursion



```
int count = 0;
for (int i = 0; i < 10; i++)
    count += 1;
```



```
let rec count accumulator upperbound =
    match upperbound with
        | 0 -> accumulator
        | _ -> count (accumulator+1) (upperbound-1)
```

Here you can see also an example of pattern matching (wait for it)

Functional Programming - Basic Concepts

But how i can manage state, caching, …
Yes, you can:

- Create a new IMMUTABLE value with the modification you want to perform.
- Use some little advanced concepts like monads (i will not explain them to you but google is your friend)

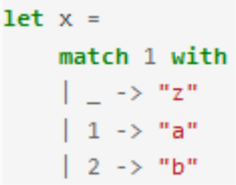
Here there's some references to this topic and examples developed in functional style you can find interesting:

- http://matthewmanela.com/blog/functional-stateful-program-in-f/
- http://fsharpforfunandprofit.com/posts/enterprise-tic-tac-toe/
- http://www.alexeyshmalko.com/2015/io-is-your-command-pattern/

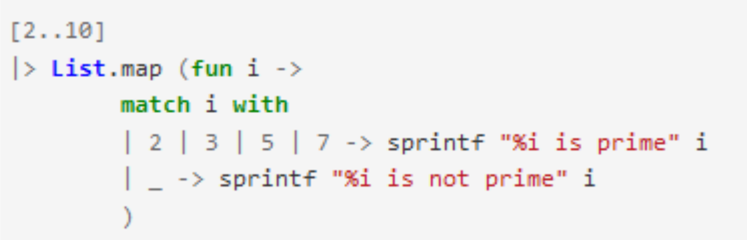
Functional Programming - Basic Concepts

Pattern Matching

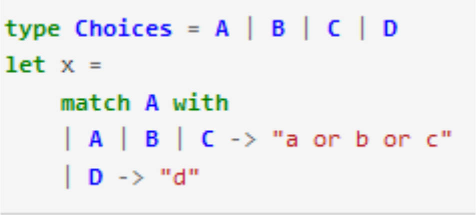
Some examples:



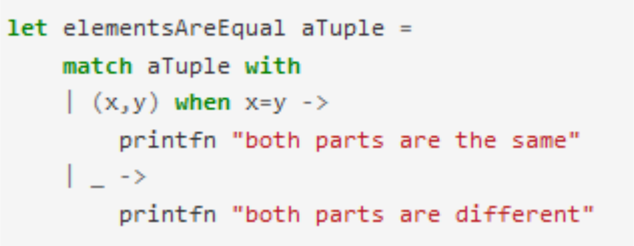
```
let x =
    match 1 with
    | _ -> "z"
    | 1 -> "a"
    | 2 -> "b"
```



```
[2..10]
|> List.map (fun i ->
        match i with
        | 2 | 3 | 5 | 7 -> sprintf "%i is prime" i
        | _ -> sprintf "%i is not prime" i
        )
```



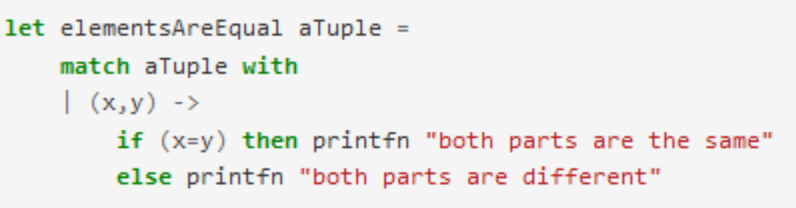
```
type Choices = A | B | C | D
let x =
    match A with
    | A | B | C -> "a or b or c"
    | D -> "d"
```



```
let elementsAreEqual aTuple =
    match aTuple with
    | (x,y) when x=y ->
        printfn "both parts are the same"
    | _ ->
        printfn "both parts are different"
```

http://fsharpforfunandprofit.com/posts/match-expression/

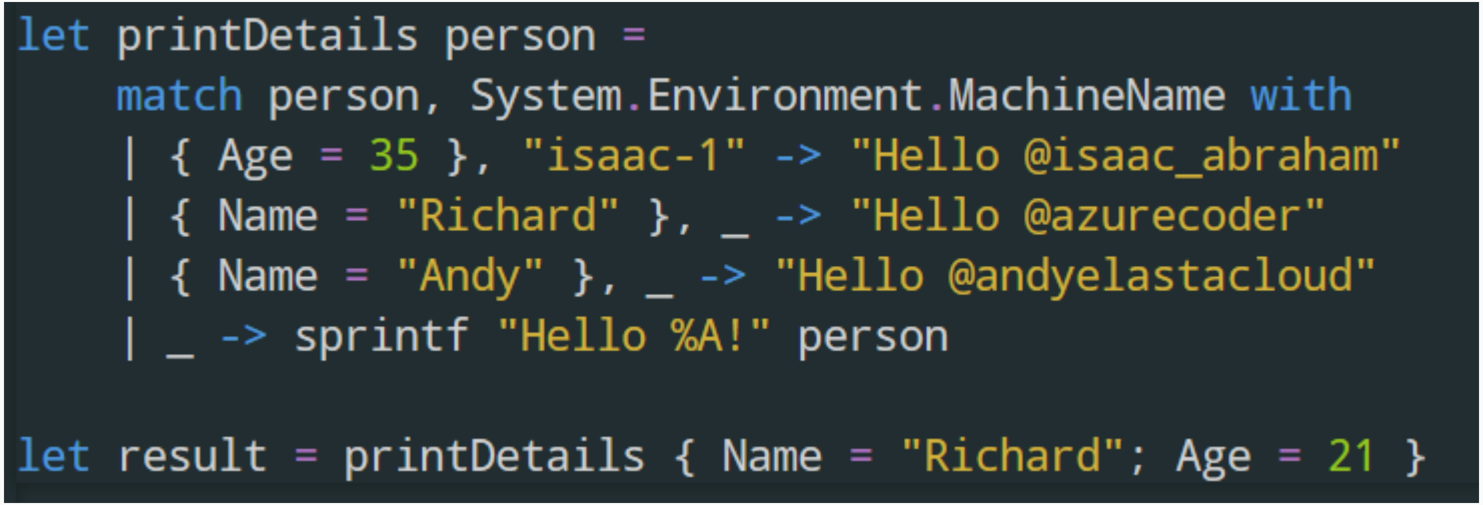
When Clause is a syntactic sugar for this:



```
let elementsAreEqual aTuple =
    match aTuple with
    | (x,y) ->
        if (x=y) then printfn "both parts are the same"
        else printfn "both parts are different"
```

Pattern Matching

Someone will argue that pattern matching is the same of the switch statement of OOP
Remember the previous slide about person type?

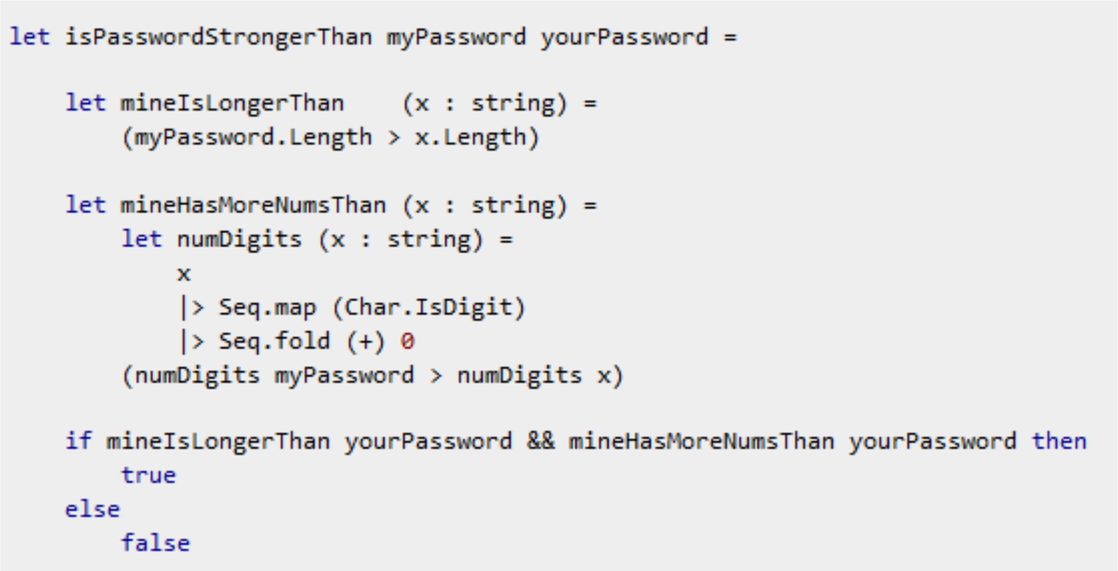


```
let printDetails person =
    match person, System.Environment.MachineName with
    | { Age = 35 }, "isaac-1" -> "Hello @isaac_abraham"
    | { Name = "Richard" }, _ -> "Hello @azurecoder"
    | { Name = "Andy" }, _ -> "Hello @andyelastacloud"
    | _ -> sprintf "Hello %A!" person

let result = printDetails { Name = "Richard"; Age = 21 }
```

How can do this with the switch?

Functional Programming - Basic Concepts

Closures



```
let isPasswordStrongerThan myPassword yourPassword =

    let mineIsLongerThan    (x : string) =
        (myPassword.Length > x.Length)

    let mineHasMoreNumsThan (x : string) =
        let numDigits (x : string) =
            x
            |> Seq.map (Char.IsDigit)
            |> Seq.fold (+) 0
        (numDigits myPassword > numDigits x)

    if mineIsLongerThan yourPassword && mineHasMoreNumsThan yourPassword then
        true
    else
        false
```

captures values and brings them into an inner scope without being passed as parameters.

In the example 'myPassword' is used in all the inner functions, but it wasn't passed as a parameter.

Functional Programming - Other Concepts

- Lazy evaluation
- Tail Recursion
- Differences Between Expression and Statement
  - Expression: Evaluated to produce a value (C# Conditional Expression: ?: operator)
  - Statement: Executed to update a variable (C# Conditional Statement: if-else statement)
- Functional Origins and Theory: Lambda Calculus
- Concurrency
- For/List Comprehension
- Differences Between Types and Classes
- Monads, Monoid, Map/Reduce
- Standard Higher Order Function
- ...

Railway Oriented Programming Distilled

What’s this S*#@???

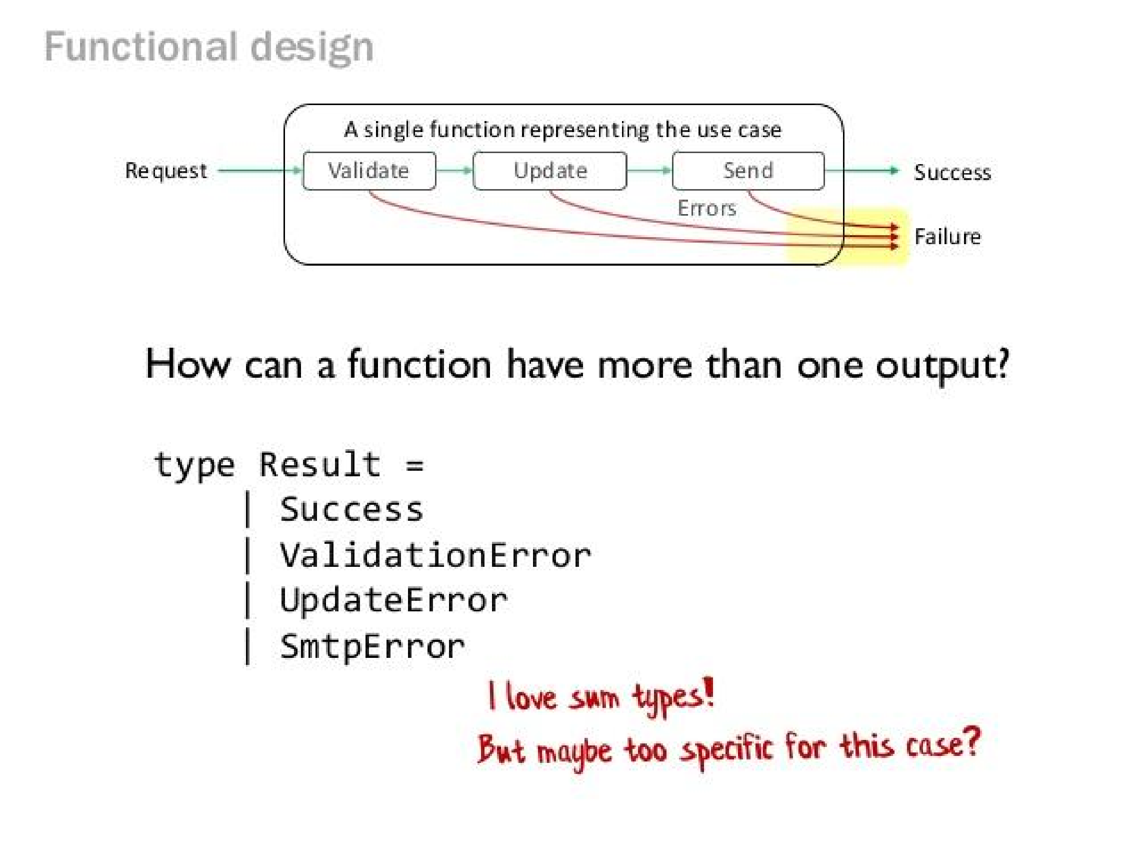
A simple way to manage the Error handling and bad stuff. (it work also with c#)

Before getting started we must define some term:
Happy Path: list of steps in a program without errors, from start to end. (like unicorns)
Errors: simply #@[&@+!!/&!/@!!

Unfortunately inside it use the monads concept, but them are not required to know. (but better for sure)
References:

- http://fsharpforfunandprofit.com/rop/
- https://github.com/fsprojects/Chessie
- https://fsprojects.github.io/Chessie/

Railway Oriented Programming



Functional design

A single function representing the use case

Request → Validate → Update → Send → Success

Errors

Failure

How can a function have more than one output?

```
type Result =
     | Success
     | ValidationError
     | UpdateError
     | SmtpError
```

I love sum types!

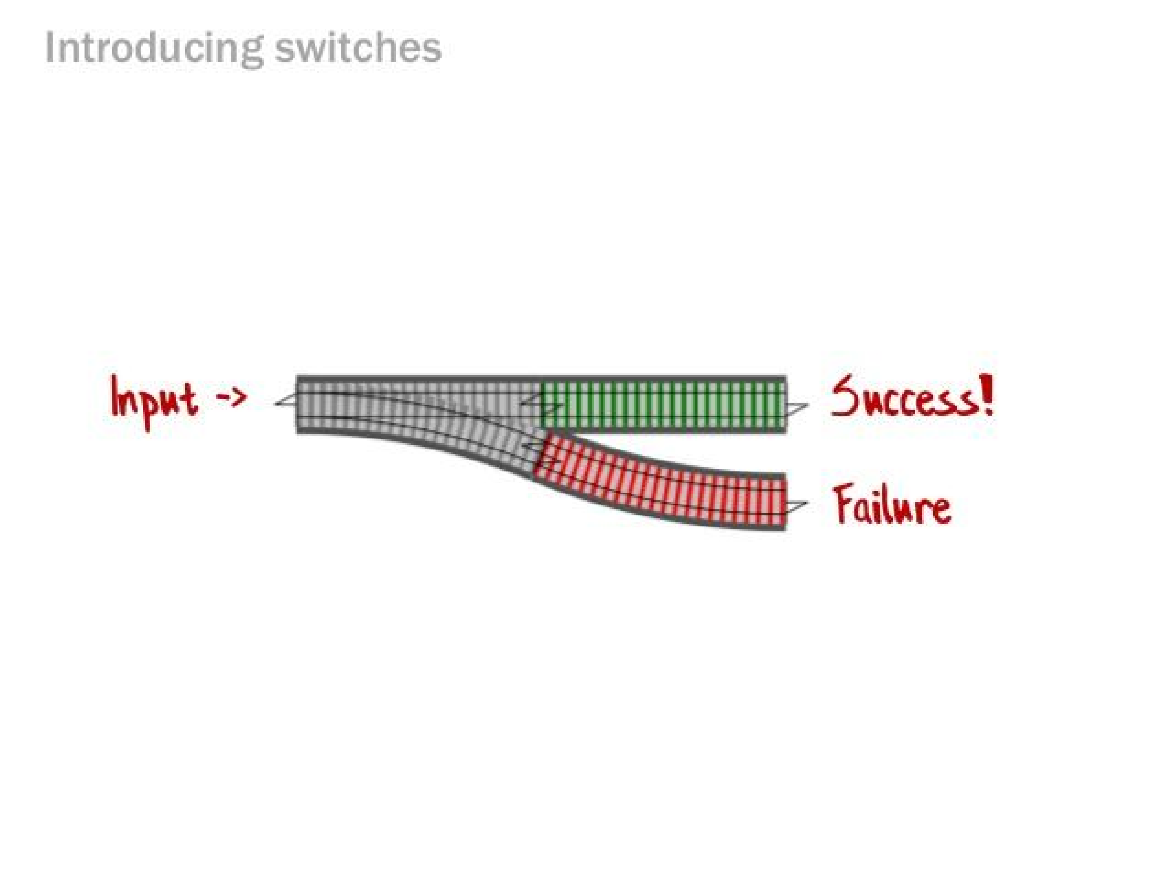
But maybe too specific for this case?

(Stolen Slide)

Green: Happy Path
Red: Errors of some sort

But for every error i must made a specific type?
A little bit tricky, isn’t it?

Railway Oriented Programming



Introducing switches

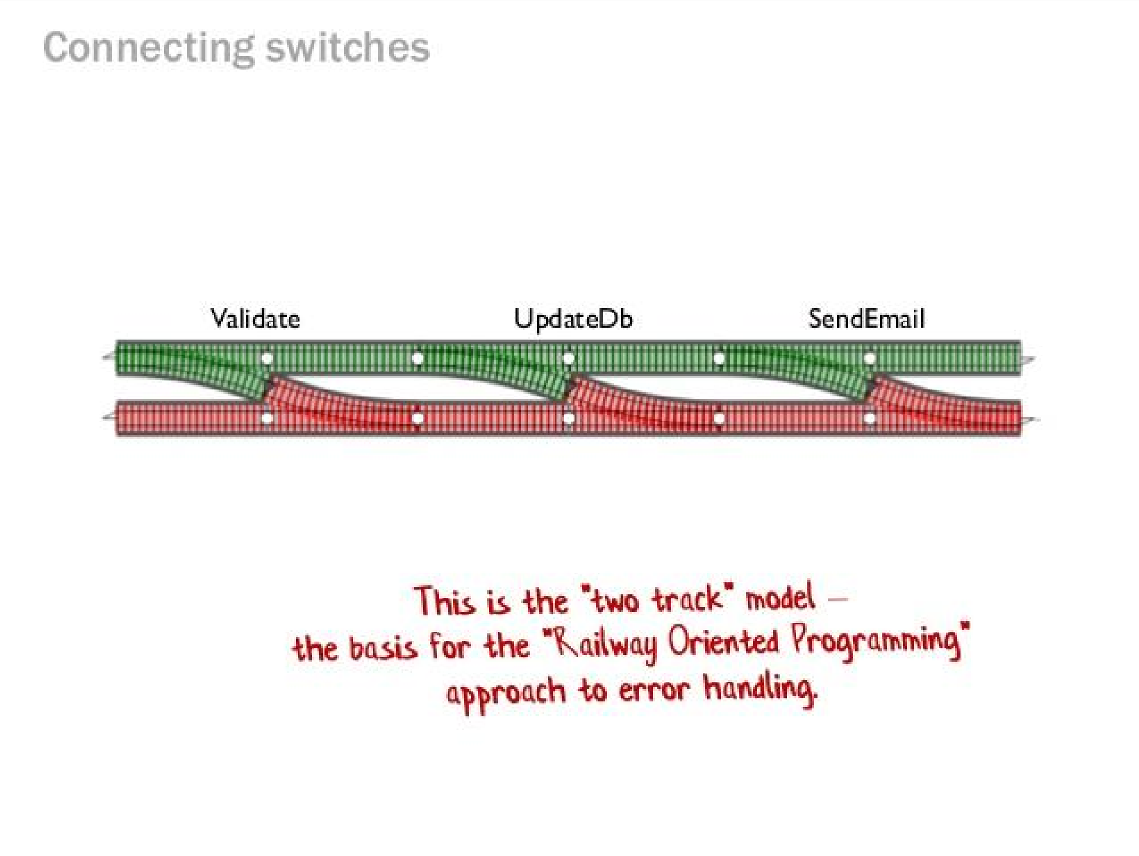
This is why Railways!

Did you remember function composition?
So we can compose railways!

In particular:

- Continue to in the Happy Path if all goes well.
- Go away if something wrong happens!
- And in the end...

Railway Oriented programming



Connecting switches

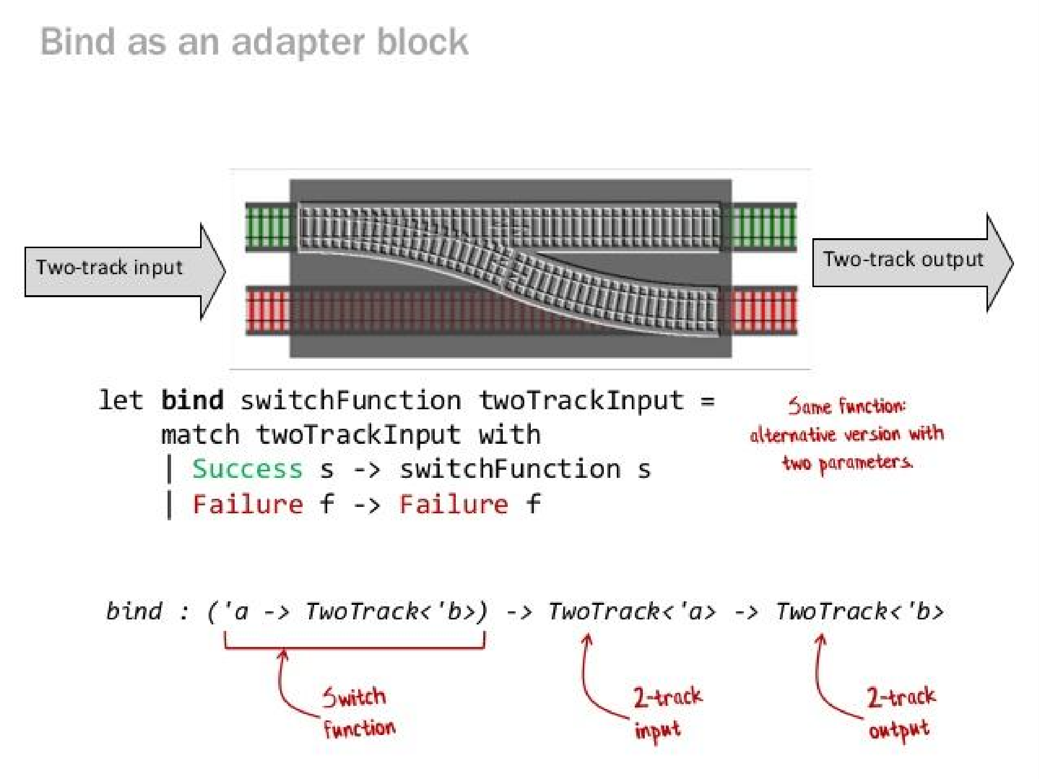
This is the "two track" model –
the basis for the "Railway Oriented Programming"
approach to error handling.

Ok, looks fine...but....

Hey, the previous switch has one rail as input, not two!!

So we need an operator to BIND things made of one input, to things with two input

Railway Oriented Programming



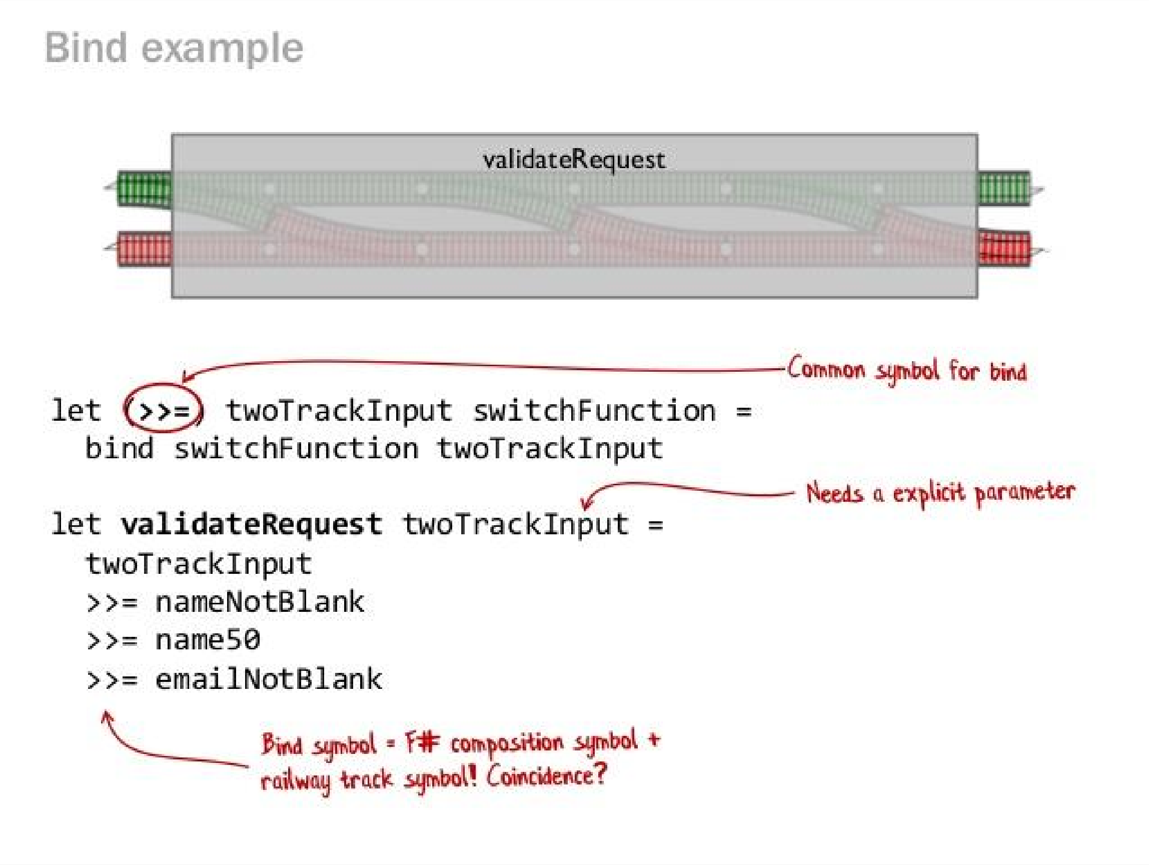
Bind as an adapter block

```
let bind switchFunction twoTrackInput =
    match twoTrackInput with
    | Success s -> switchFunction s
    | Failure f -> Failure f
```

Same function: alternative version with two parameters.

```
bind : ('a -> TwoTrack<'b>) -> TwoTrack<'a> -> TwoTrack<'b>
```

Railway Oriented Programming



Bind example

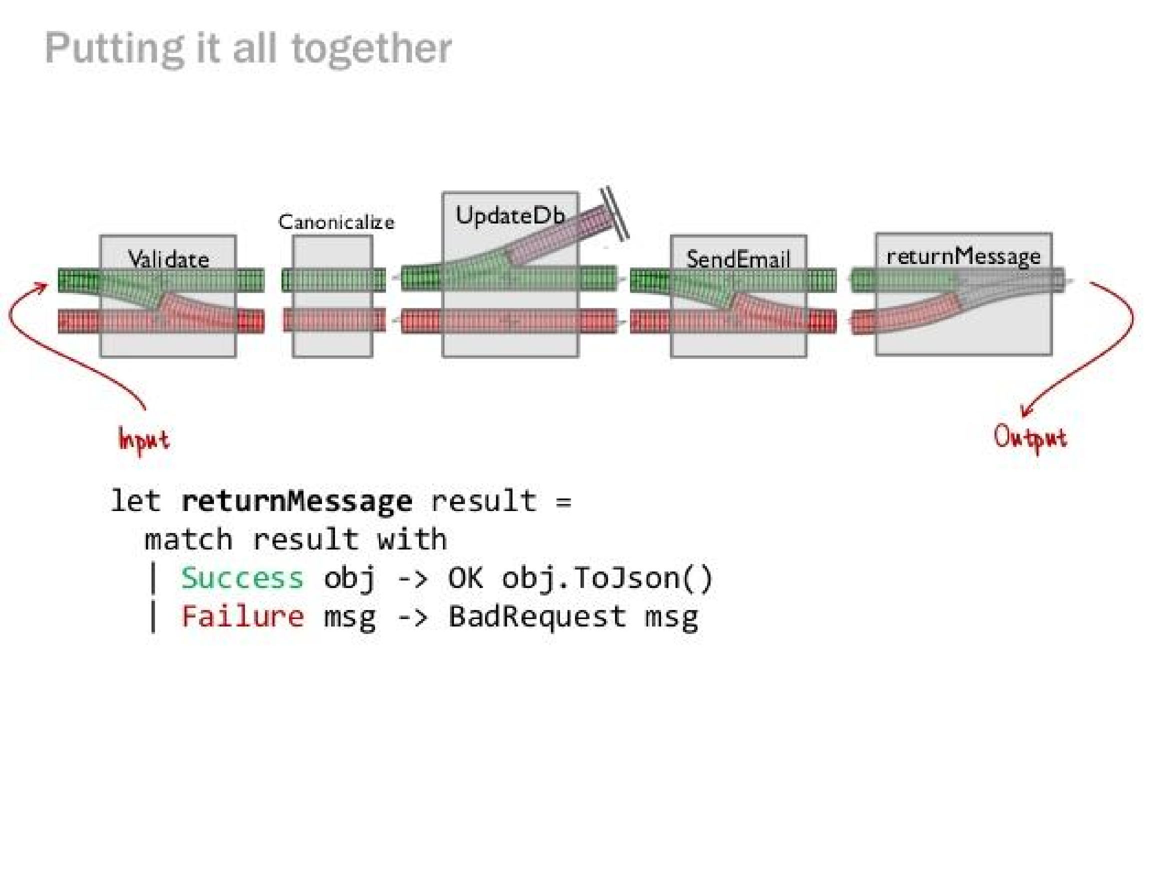
```
let (>>=) twoTrackInput switchFunction =
  bind switchFunction twoTrackInput

let validateRequest twoTrackInput =
  twoTrackInput
  >>= nameNotBlank
  >>= name50
  >>= emailNotBlank
```

Common symbol for bind

Needs a explicit parameter

Bind symbol = F# composition symbol + railway track symbol! Coincidence?

Railway Oriented Programming



Putting it all together

```
let returnMessage result =
  match result with
   | Success obj -> OK obj.ToJson()
   | Failure msg -> BadRequest msg
```

In the end we need a function that checks the error and decide what to do.

HEY, you are hiding stuff!! What’s all that railways!!

This is the Railway Oriented Programming Distilled. If you want to know see the original post and talk!

You, Lazy Programmer!!

But I don’t want to learn this stuff all by myself!!

Here some reference to you that help me a lot:

- http://fsharpforfunandprofit.com/
- https://www.coursera.org/course/progfun
- https://www.edx.org/course/introduction-functional-programming-delftx-fp101x-0
- https://www.youtube.com/playlist?list=PLoJC20gNfC2gpI7Dl6fg8uj1a-wfnWTH8
- http://pitofsuccess.azurewebsites.net/#/4/1
- https://ptgmedia.pearsoncmg.com/images/9780735670266/samplepages/9780735670266.pdf

Time to see some of this stuff LIVE