Functional Programming

Intro to F#

OOP makes code understandable by encapsulating moving parts. FP makes code understandable by eliminating moving parts.

— *Michael Feathers*
(author of Working with Legacy Code)

Riccardo Terrell

Agenda

- History and influence of Functional Programming
- What & Why Functional Programming
- What & Why F#
- F# |> intro |> Code samples

Something about me – Riccardo Terrell

- Originally from Italy. I moved here ~8 years ago
- +/- 15 years in professional programming
  - C++/VB → Java → .Net C# → C# & F# → ??
- Organizer of the DC F# User Group www.meetup.com/DC-fsharp
- Software Architect @ MSFT
- Passionate in Technology, believer in polyglot programming as a mechanism in finding the right tool for the job

Objectives

- **Use functions as building block and composition to solve complex problem with simple code**
- **Functional and imperative styles should NOT stand in opposition but they should COEXIST**
- **F# is a functional-first powerful language** that offers a major step forward in productivity and effectiveness

History

λ

1930

1958 LISP

1962 APL

1966 ISWIM

1973 ML

1977 FP

1988 Haskell

1986 Erlang

1996 Ocaml

2003 Scala

2005 F#

2007 Clojure

Functional Programming Influence

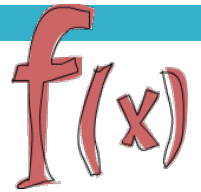
- Continue Evolution of GC – Lisp ~1958
- Generics – ML ~ 1973
- In early 1994, support for lambda, filter, map, and reduce was added to Python
- List comprehension - Linq in C# 3.0 and Java 8
- First-class functions were also introduce Visual Basic 9, C#, Java 8 and C++ 11
- Java 8 supports lambda expressions as a replacement for anonymous classes
- Is Lazy Evaluation is part of the main stream languages such as C#
- Type Inference, example is the "var" keyword in C#

Programming in a functional style can also be accomplished in languages that aren't specifically designed for functional programming, Lots of modern languages have elements from FP

Real World Uses

F#	Svea Ekonomi, Microsoft (Halo-Game), Credit Suisse, Prover
Haskell	AT & T, Scrive
Erlang	Ericsson, Amazon (SimpleDB), Facebook
Lisp	Yahoo Store
Scala	Twitter, LinkedIn

What is "Functional Programming?"



Functional programming is just a style, is a programming paradigm that treats computation as the evaluation of functions and avoids state and mutable data…

$$y = f(x)$$

- *Functions as primary building blocks (first-class functions)*
- *A function is a transformation. It transforms one or more inputs into exactly one output*
- *Programming with immutable variables and assignments*
- *Programs work by returning values instead of modifying data*

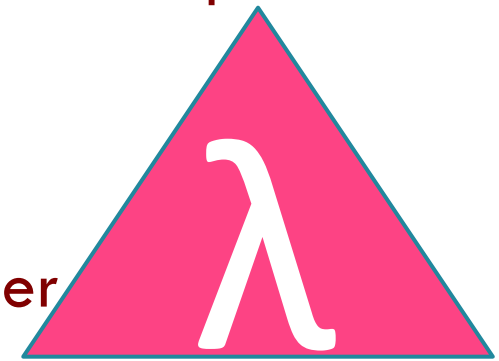
What is "Functional Programming?"

Functional programming is so called because a program consists entirely of functions.

— John Hughes, Why Functional Programming Matters

Composition

First class
Higher-Order
Functions



Immutability

… is all about functions! Identifying an abstraction and building a function, use existing functions to build more complex abstractions, pass existing functions to other functions to build even more complex abstractions

Higher-Order functions

A higher-order function is a function that takes another function as a parameter, or a function that returns another function as a value, or a function which does both.

```
let getGoodSeniorDeveloper developers =
        // Developer -> GoodSeniorDeveloper
        let convertToGSD (d:Developer) = {FirstName = d.FirstName;
          LastName=d.LastName};
        // Developer -> bool
        let isGoodSeniorDeveloper d = d.IsFunctional && d.YearsOfExperience >= 5

        developers
        |> List.filter isGoodSeniorDeveloper
        |> List.map convertToGSD
```

Immutability

- Immutability is very important characteristic of FP but mostly is about transformation of state…
  Immutable data forces you to use a “transformational” approach

- When you’re writing programs using immutable types, the only “thing a method can do is return a result, it can’t modify the state of any objects
- Immutable data makes the code predictable is easier to work
- Prevent Bugs
- Concurrency is much simpler, as you don’t have to worry about using locks to avoid update conflicts
  - Automatic thread-safe, *controlling mutable state in concurrency is very hard*

Immutability

Mutable values cannot be captured by closures!

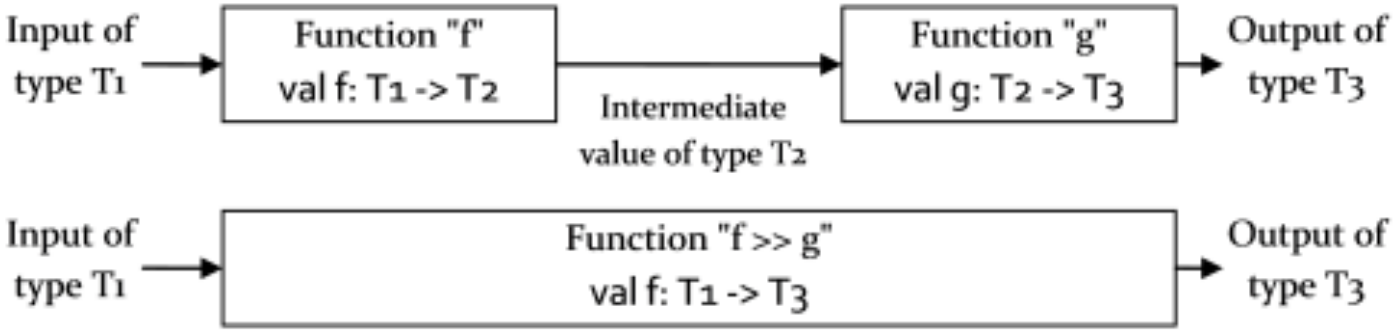
```
List<Func<int>> actions = new List<Func<int>> ();
for (int i = 0; i < 5; i++)
        actions.Add (() => i * 2);

foreach (var action in actions)
        Console.WriteLine (action ());
```


```
let actions = List.init 5 (fun i -> fun() -> i * 2)
for action in actions do
    printfn "%d " (action())
```

Function Composition

Function Composition - Building with composition. Composition is the 'glue' that allows us build larger systems from smaller ones. This is the very heart of the functional style. Almost every line of code is a composable expression. Composition is used to build basic functions, and then functions that use those functions, and so on.

Composition — in the form of passed parameters plus first-class functions



Sa far we talk about …

- Higher Order Function
- Immutability
- Composition

Compose Mario

Mario game sample, based on functional reactive Elm script (http://elm-lang.org/edit/examples/Intermediate/Mario.elm)

The functions that depend on keyboard take the current keyboard state as the first argument. This is represented as a tuple int*int (dir) consisting of x and y directions.

```
// If the Up key is pressed (y > 0) and Mario is on the ground,
// then create Mario with the y velocity 'vy' set to 5
let jump (_,y) m = if y > 0 && m.y = 0. then  { m with vy = 5. } else m
// If Mario is in the air, then his "up" velocity is decreasing
let gravity m = if m.y > 0. then { m with vy = m.vy - 0.1 } else m
// Apply physics - move Mario according to the current velocities
let physics m = { m with x = m.x + m.vx; y = max 0. (m.y + m.vy) }
// When Left or Right keys are pressed, change the 'vx' velocity and direction
let walk (x,_) m =
    { m with vx = float x
             dir = if x < 0 then "left" elif x > 0 then "right" else m.dir }
```

The step function of the game takes previous Mario value and returns a new one. It is composed from 4 functions that represent different aspects of the game.

```
let step dir mario = mario |> physics |> walk dir |> gravity |> jump dir
```


DEMO

Why Functional Programming?

- Changes the way you think about programming and problem solving
- Functional Programming promotes Composition and Modularity
- Simple Code for Complex Problems
- Declarative programming style
- Concurrency, FP is easy to parallelize
  - *Locking vs. immutable memory*
- Conciseness, less code *(no null checking)*
- Correctness & Reduced bugs *(immutability)*
- Rapid Prototyping

F# |> INTRO

What is F#

- A strongly typed, functional first, hybrid, open source, programming language on the CLR
  - By Don Syme and his team @ Microsoft Research, who productized Generics
  - Based on oCaml and influenced by C# & Haskell

- History
  - 2002: F# language design started
  - 2005 January: F# 1.0.1 releases to public
    - Not a product. Integration with VS2003
    - Works in .NET 1.0 through .NET 2.0 beta, Mono
  - 2005 November: F# 1.1.5 with VS 2005 RTM support
  - 2009 October: VS2010 Beta 2, CTP for VS2008
  - 2010: F# is “productized” and baked into VS 2010
  - November 2010 F# Compiler and libraries released under Apache 2.0

What is F#

… a productive, expressive, interoperable, functional language that allows you to write simple code to solve complex problems

What is F#


Miguel de Icaza
@migueldeicaza
Following
The latest Mono (3.0.2) now ships with F# 3 on OSX
Reply
Retweet
Favorite
47
RETWEETS
9
FAVORITES
12:33 AM - 6 Dec 12 · Embed this Tweet

F# Community

http://fsharp.org

http://tryfsharp.org

F# Characteristics

1. Basic types from the .NET CLI plus classes, interfaces, generics
2. Records and Structures
3. Tuples, Lists, Arrays
4. Type Inference – Statically typed
5. Immutable
6. Supports Lazy evaluations
7. Recursion
8. Function Types, Lambdas - High Order Functions
9. Curry - Partial Application
10. Discriminated Unions
11. Pattern Matching
12. Unit of Measure


DEMO

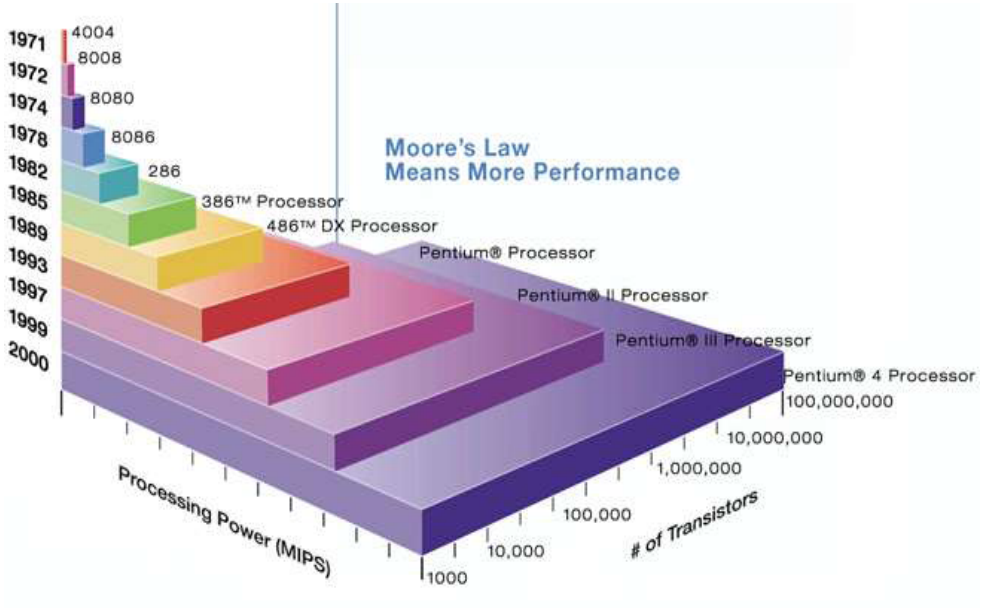
Why F#

- **F# is compatible**: runs on CLR/Mono, interoperability with other CLR languages
- **F# is simple and compact**: a very small language core with a flat learning curve
- **F# is high-level**: Higher level of abstraction by combining OO with functional programming
- **F# is statically typed**: type inference gives F# the „look&feel“ of a dynamically typed language
- **Easy to parallelize**: Sets you up for multi-core programming
- **Easy to test and debug**
- **Increase of productivity**

Concurrency

Modern computers come with multiple processors, each equipped with multiple cores, but the single processor is slower than used to be!

Moore's law *(http://en.wikipedia.org/wiki/Moore's_law)*



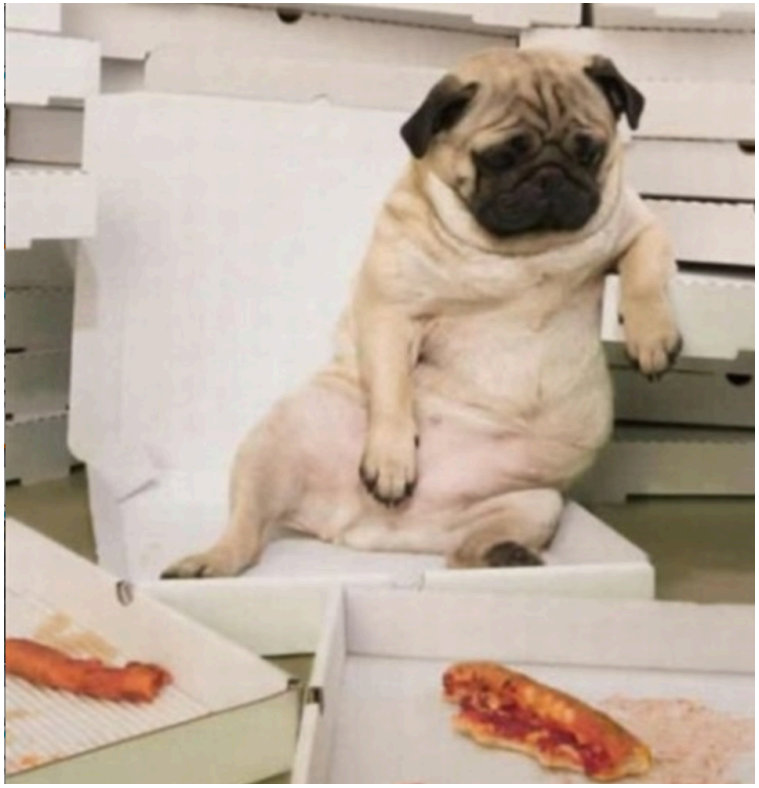
… to achieve great performances the application must be leveraging a Concurrent Model

Concurrency

There is a problem…

the free lunch is over

- Programs are not doubling in speed every couple of years for free anymore
- We need to start writing code to take advantage of many cores



F# and Multi-Core Programming

- *Thanks to the **immutability** (and **isolation**), we avoid introducing race conditions and we can write lock-free code.*
- F# is, technically speaking, neutral with respect to concurrency
  - it allows the programmer to exploit the many different techniques for concurrency and distribution supported by the .NET platform
- Asynchronous workflows make writing parallel programs in a "natural and compositional style"
- F# supports the Actor Model, F# Agents by themselves allow you to define an asynchronous encapsulated subprocess that you design to receive a certain type of messages and emit an answer

Objectives

- **Use functions as building block and composition to solve complex problem with simple code**

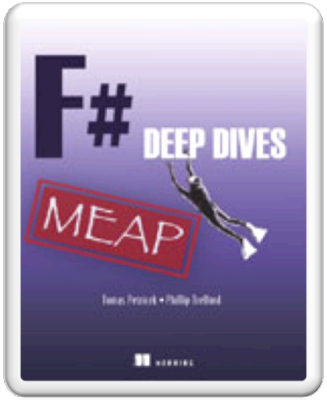
- ***Poly-paradigm programming is more powerful and effective than polyglot programming, learn a new paradigm it will make a better developer***

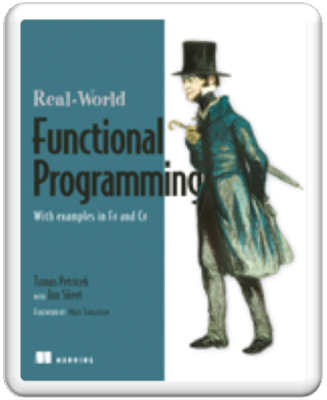
- **F# is a functional-first powerful language** that offers a major step forward in productivity and effectiveness

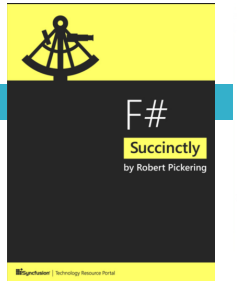
The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities.

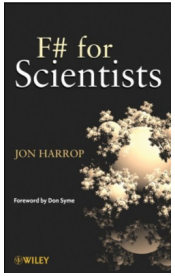
-- Edsger Dijkstra

Resources



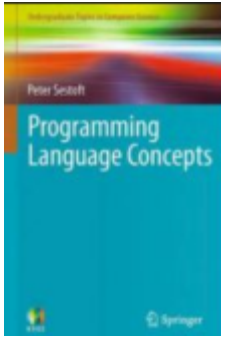






http://fsharp.org

http://www.tryfsharp.org



How to reach me

https://github.com/dcfsharp/FSIntro

meetup.com/DC-fsharp/

@TRikace @DCFsharp

rterrell@microsoft.com