MF#K - Introductory Course in F#

@ Skillshouse 2015-02-24

Mødegruppe for Ƒunktionelle Københavnere (MF#K)

Overview

- About me
- Matching of expectations
- Agenda
  - 09:00 |> Short introduction to F# (sales pitch)
  - 09:15 |> Environment setup, Basics and Domain modeling
  - 10:15 |> Break
  - 10:30 |> .NET applications / libraries
  - 11:30 |> Lunch
  - 12:30 |> Data and TypeProviders
  - 13:30 |> Break
  - 13:45 |> Parallelism and concurrency
  - 14:45 |> Break
  - 15:00 |> Robust and bulletproof applications
  - 16:00 |> Summary: U want more?

About me (very shortly)

- Ramón Soto Mathiesen
- MSc. Computer Science from DIKU (Minors in Mathematics)
- Managing Specialist |> CTO of CRM Department @ Delegate A/S
  - ER-modeling, WSDL, OData (REST API)
- F# / C# / JavaScript / C++: Delegate A/S @ GitHub
- Blog: http://blog.stermon.com/

Matching of expectations

- What are you expectations for this course?

Matching of expectations

- Ƒunctional Copenhageners Meetup Group will try to get more and more software projects to be based on functional programming languages. We mainly focus on F# and Haskell, but other functional programming languages like Scala, Lisp, Erlang, Clojure, OCaml, etc. are more than welcome.
- We noticed that customers would like to get "proper training" in F# and that's why we are standing here today. We are also expecting an increase in the need for training so we can attract renowned trainers to Denmark (Phil Trelford, Tomas Petricek, …)
- We expect that our attendees to this introductory course in F#, will be able to make production-ready application afterwards

Short introduction to F# (sales pitch) - Buzzwords

- *less code, error-free projects, only one code base, big data, parallelism, concurrency, asynchronous processes*

Short introduction to F# (sales pitch) - F# - What is it?

- Is an ***open-source***, ***strongly typed***, ***multi-paradigm*** programming language encompassing ***functional***, ***imperative*** and ***object-oriented*** designed by Don Syme (MS Research Cambridge UK) and maintained by Microsoft, F# Software Foundation and open contributors
- It's a mature language that is part of Visual Studio and the .NET Framework
- Loved by the ***very talented*** who contribute to it for free with sometimes very usable projects:
  - Special mention to (among others):
    - Tomas Petricek (TomASP.NET)
    - Scott Wlaschin (F# for fun and profit)

Short introduction to F# (sales pitch) - F# - Why use it?

- Conciseness
- Convenience
- Correctness
- Concurrency
- Completeness

Short introduction to F# (sales pitch) - F# - Why use it?

```
let swap (x,y) = y,x
let foo = swap(42,0)
let bar = swap("42","0")
```

```
> val swap : x:'a * y:'b -> 'b * 'a
> val foo : int * int = (0, 42)
> val bar : string * string = ("0", "42")
```

- Conciseness:
  - F# is not cluttered up with coding noise such as curly brackets, semicolons and so on
  - You almost never have to specify the type of an object, thanks to a powerful ***type inference system***.
  - And, compared with C#, it generally takes ***fewer lines of code*** to solve the same problem

Short introduction to F# (sales pitch) - F# - Why use it?

```
let f g x = g x
f (fun x -> x * x) 42
```

```
> val f : g:('a -> 'b) -> x:'a -> 'b
> val it : int= 1764
```

- Convenience:
  - Many common programming tasks are much simpler in F#. This includes things like creating and using ***complex type definitions***, doing ***list processing***, ***comparison and equality***, ***state machines***, and much more
  - And because functions are first class objects, it is very easy to create powerful and reusable code by creating functions that have ***other functions as parameters***, or that ***combine existing functions*** to create new functionality

Short introduction to F# (sales pitch) - F# - Why use it?

```fsharp
[<Measure>] type DKK
[<Measure>] type USD
let rate : float<USD/DKK> = 0.2<USD/DKK>
let usd2dkk (amount: float<USD>) = amount / rate
type OpportunityDK = { Customer : string; Amount : float<DKK> }
type OpportunityUS = { Customer : string; Amount : float<USD> }
type Opportunities = | DK of OpportunityDK | US of OpportunityUS
let odk0 = { OpportunityDK.Customer = "Skillshouse A/S"; Amount = 42.<DKK> }
let odk1 = { OpportunityDK.Customer = "Microsoft Danmark ApS"; Amount = 42.<DKK> }
let ous2 = { OpportunityUS.Customer = "Microsoft Redmond HQ"; Amount = 42.<USD> }
[ DK(odk0); DK(odk1); US(ous2); ]
|> List.map(fun x -> match x with | DK y -> y.Amount | US y -> usd2dkk y.Amount)
|> List.reduce(+)
```

- Correctness:
  - F# has a ***powerful type system*** which prevents many common errors such as ***null reference exceptions***.
  - Values are ***immutable by default***, which prevents a large class of errors
  - In addition, you can often encode ***business logic*** using the ***type system*** itself in such a way that it is actually ***impossible to write incorrect code*** or mix up ***units of measure***, greatly ***reducing the need for unit tests***

Short introduction to F# (sales pitch) - F# - Why use it?

```
[|0 .. 10 .. (1 <<< 16)|]
|> Array.map(fun x -> x * x)
[|0 .. 10 .. (1 <<< 16)|]
|> Array.Parallel.map(fun x -> x * x)
```

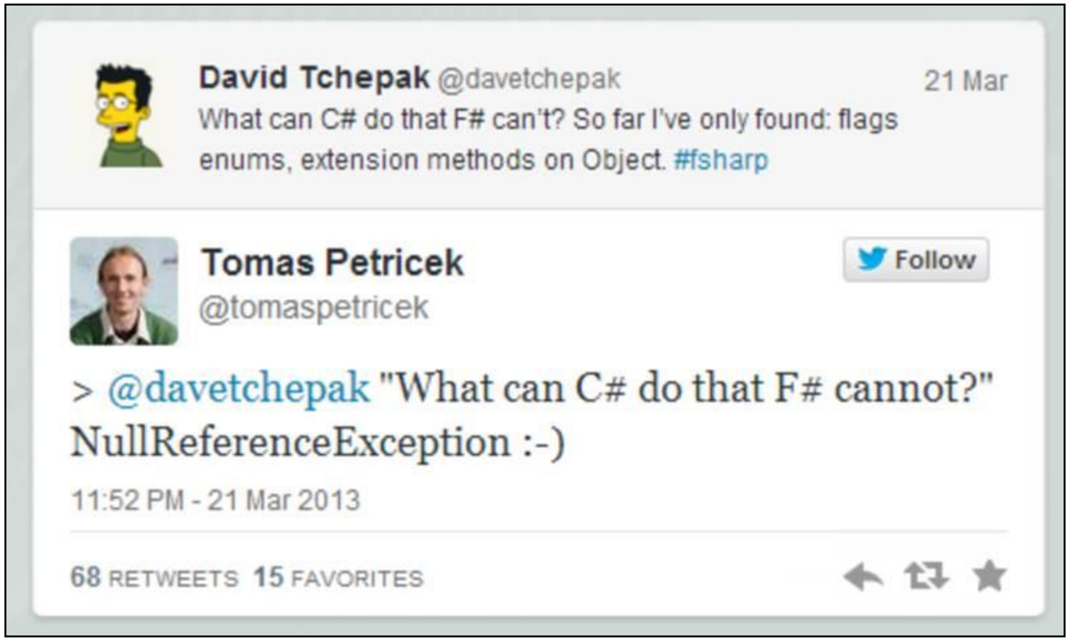
- Concurrency:
  - F# has a number of built-in libraries to help when more than one thing at a time is happening. Asynchronous programming is ***very easy***, as is parallelism. F# also has a built-in ***actor model***, and excellent support for event handling and ***functional reactive programming***
  - And of course, because ***data structures are immutable by default***, ***sharing state*** and ***avoiding locks*** is much easier

Short introduction to F# (sales pitch) - F# - Why use it?

```
open System
let ts () = DateTime.Now.ToString("o")          // ISO-8601
let ts' () = (ts ()).Replace(":", String.Empty) // Filename safe
let cw (s:string) = Console.WriteLine(s)
let cew (s:string) = Console.Error.WriteLine(s)
```

- Completeness:
  - Of course, ***F# is part of the .NET ecosystem***, which ***gives you*** seamless ***access to all the third party .NET libraries and tools***.
  - Finally, ***it is well integrated with Visual Studio***, which means you get a great IDE with ***IntelliSense support***, ***a debugger***, and many plug-ins for unit tests, source control, and other development tasks
  - Although it is a functional language at heart, F# does support other styles which are not 100% pure, which makes it much easier to interact with the non-pure world of web sites, databases, other applications, and so on. In particular, F# is designed as a hybrid functional/OO language, ***so it can do virtually everything that C# can do except …***

Short introduction to F# (sales pitch) - F# - Why use it? (*f*unny)



David Tchepak @davetchepak 21 Mar

What can C# do that F# can't? So far I've only found: flags enums, extension methods on Object. #fsharp

Tomas Petricek @tomaspetricek

> @davetchepak "What can C# do that F# cannot?" NullReferenceException :-)

11:52 PM - 21 Mar 2013

68 RETWEETS 15 FAVORITES

Short introduction to F# (sales pitch) - F# - Why use it? (Business)

- Time to Market:
  - Easy prototyping (REPL: Read-Evaluate-Print-Loop)
  - Run as .NET code
- Efficiency:
  - JIT compilation (as C#)
  - Easy to implement parallelism
- Complexity:
  - Flexible language
  - Type inference
- Correctness:
  - Advanced types
  - Close to math

Hands-on

- 09:15 |> Environment setup, Basics and Domain modeling
- 10:15 |> Break
- 10:30 |> .NET applications / libraries
- 11:30 |> Lunch
- 12:30 |> Data and TypeProviders
- 13:30 |> Break
- 13:45 |> Parallelism and concurrency
- 14:45 |> Break
- 15:00 |> Robust and bulletproof applications

Hands-on: Environment setup, Basics and Domain modeling

- Environment setup
  - MS Visual Studio 2013 Pro/Premium
  - Visual F# Power Tools
- Basics
  - Types / functions (F#) vs. Classes / Methods (C#)
  - Non-mutable data structures vs. mutable data structures
  - Recursion / tail recursion vs. for / while loops
  - Some v / None vs. null checks
  - HelloWorld.fsx (Your First F# script)
    1. Print "Hello World!" to the Console
    2. Print a folders hierarchical structure, files/folder, to the Console
- Domain modeling
  - Define the business logic as types and limit the possibility of wrong code (fewer errors, found on compile time, not runtime)
  - HelloWorldLibDomainTypes.fs
    1. Define the Domain model for "Hello World!" (Union types vs Types)
    2. Define the Domain model for a card game

Hands-on: .NET applications / libraries

- .NET applications / libraries
  - Write Unit Test and test code without having to build the final library or application
    - SetupUnitTests.fsx based on ovatsus/Setup.fsx
    - Hook up to .fsproj ***Target Name="BeforeBuild"***
  - HelloWorld.dll (Your First F# library)
    - Re-use previous data model
    - Signatures, Modules, HelperModules, UnitTests, …
  - HelloWorld.exe (Your First F # console application)
    - Call functions from the modules from the above library (assembly)

Hands-on: Data and TypeProviders

- Data and TypeProviders
  - Sequences, lists, arrays, multidimensional arrays, maps, sets (set)
  - Create types dynamically from your database using TypeProviders in your IDE ("erased types")
  - HelloWorldTypeProvider.dll (Your First F# Type Provider)
    1. Read first: Type Providers From the Ground Up from Mavnn's blog
    2. Install: Nuget FSharp.TypeProviders.StarterPack
    3. Inspiration: Phil Trelford's Date Types
    4. Create the types so when the TypeProvider is called with Visual Studios (“.” → intellisense) it will only provide “H” and when intellisense is called again it will provide “e” and so on until the last letter “!” will return the full string: “Hello World!”
    5. Create a TypeProvider that when called with Visual Studios intellisense will present the course categories from SkillsHouse. When a category is chosen, the available list are presented. A record containing the common information will be returned when a course is selected

Remark: TypeProviders can’t be used from C# code [link].

Hands-on: Parallelism and concurrency

- Parallelism and concurrency
  - Go from a sequential algorithm to a parallel by changing a few lines of code
  - HelloWorldParallel.exe and HelloWorldAsync.exe (Your first parallel / asynchronous applications)
    1. Retrieve the names and urls for all the programming languages in the following Wikipedia article: List of programming languages
    2. For each of the retrieved urls, check for each of the Wikipedia article if there is a “Hello World!” example in their respective articles
    3. Parallelize the algorithm
    4. Async and parallelize the algorithm

Hands-on: Robust and bulletproof applications

- Robust and bulletproof applications
  - Recipe for making robust and error-free applications using Some v / None and Computational Expressions
  - HelloWorldBulletproof.exe (Your first robust F# application)
    - Re-use your application from ".NET applications / libraries" and add Some v / None to all the functions
    - Implement a non-deterministic function that it will either return Some value or an Exception (failwith)
    - Ensure that you application keeps running no matter how many "minor" Exceptions are thrown
    - **Remark**: Look into "Maybe Computational Expression" / >>= operator:
      - Scott Wlaschin: Railway oriented programming
      - Ramón Soto Mathiesen: Recipe for bulletproof applications with F#

U want more?

- Code will be available @ https://github.com/gentauro
- Slides will be available @ MF#K (Files)
- Sign up @ MF#K for:
  - More ***fun***
  - Hands-on:
    - Phil Trelford: Hands On Fparsec (2015-03-17)
  - Talks:
    - In the pipeline talks about: ***Erlang***, ***Haskell***, ***Rust***, ...
      - Up next: Erlang in general and Haskell with CUDA (May month)
- MF#K would like to thank our sponsors:

skillshouse learning by doing

delegate Shared Success

Microsoft