



MF#K - Introduction to F#

ERFA meeting @ PFA 2015-02-27

Mødegruppe for Funktionelle Københavnerne
(MF#K)





- About me
- Matching of expectations
- Agenda
 - 09:00 |> Short introduction to F# (sales pitch)
 - 09:20 |> Demo: F# and LivNet (FK64: Settlement of remuneration)
 - 10:00 |> Summary: U want more?



- 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/>



- What are your expectations for this introduction to F#?



- Functional 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 expect that attendees to this introduction to F#, will get inspired to use the language in the future 😊



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



- 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](#))



- Conciseness
- Convenience
- Correctness
- Concurrency
- Completeness



```
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



```
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



```
[<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**



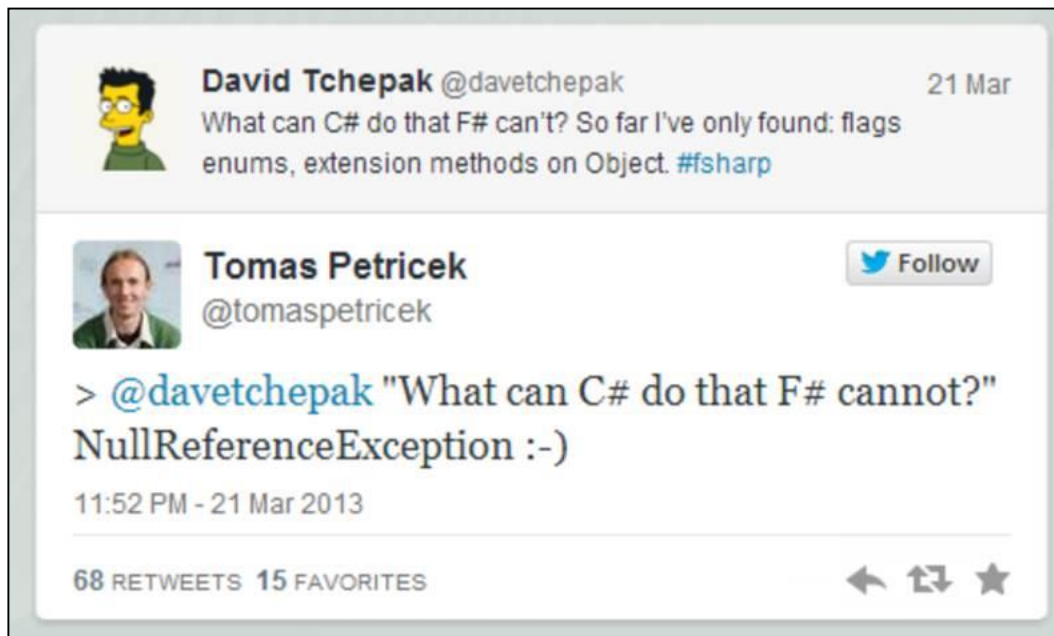
```
[|0 .. 10 .. (1 <<< 16)|]  
|> Array.map(fun x -> x * x)  
[|0 .. 10 .. (1 <<< 16)|]  
|> Array.Parallel.map(fun x -> x * x)
```

- 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



```
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 ...***



Remark: `string` in F# can be *null* as well (primitive .NET types)



- 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



- After a short demonstration of LivNet by Daniel Olsen we decided that it would make sense to showcase an example how to post-process a CSV file generated by the system
- We have chosen the following script (kørsel):
 - FK64: Afregning af vederlag
 - Task to be shown:
 1. Use the [Fsharp.Data](#) TypeProvider to *infer types* from .CSV file based on a sample of the script (*CsvProvider*)
 2. Load data from the script **FK64: Afregning af vederlag**
 3. Remove entries *without* a Bank
 4. Normalize amount by multiplying with **0.001**
 5. Show amount in a graph ([FSharp.Charting](#)) combined with *Value date*.

Remark: All the above in only less than 50 lines of well-written code 😊



- Code will be available @ dao@pfa.dk
- Slides will be available @ dao@pfa.dk
- 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 sponsor(s):



Shared Success