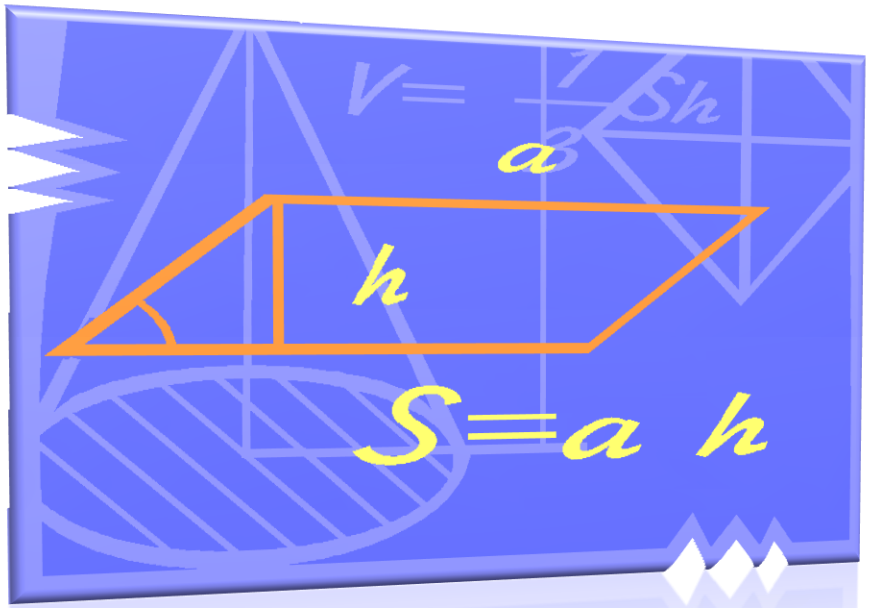
Functional Programming in C# 3.0

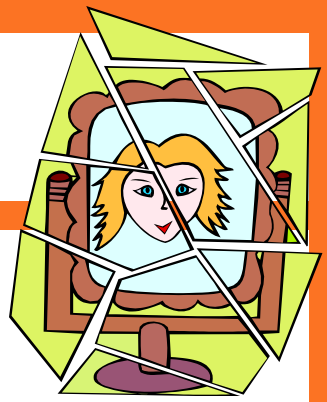


Oliver Sturm
oliver@sturmnet.org
olivers@devexpress.com
http://www.sturmnet.org/blog

Oliver Sturm

MVP Microsoft Most Valuable Professional

Who's that Oliver Sturm guy, anyway?



I am

- Interested in programming languages, databases and a whole bunch of other things
- Microsoft MVP for C#

I do

- Work for Developer Express as a Technical Evangelist and Lead Program Manager
- Blog at http://www.sturmnet.org/blog

You should

- Email me at oliver@sturmnet.org

Agenda

Focus: What C# can do with regard to FP and how it works

- What is Functional Programming?
- FP Features in C# 3.0 and .NET 3.5
- Map, Filter and Reduce
- Currying, Partial Application and Composition
- How does a C# programmer benefit from FP?

What is Functional Programming?

- A programming paradigm
- Focus on the application of functions
- Avoids state and mutable data
- Well-known languages include Lisp, Scheme, Haskell, ML and (recently) F#
- FP languages tend to have features that support Higher Order Functions, currying, recursion, list comprehensions, ...
- Many imperative and OO languages have FP features today

Why is Functional Programming interesting?

- Promotes modularization
- Lazy evaluation → greater efficiency
- The target of avoiding side effects has several advantages: scalability, optimization, debugging, testing
- C# 3.0 supports many important FP techniques

Demo

What's in the box

Interlude – Map, Filter, Reduce

- **Map/Select** does something with each element in a list
- **Filter/Where** extracts elements from a list based on some condition
- **Reduce/Aggregate** summarizes elements in a list according to some calculation
- Select, Where and Aggregate are .NET 3.5 implementations of these functions

Demo continued

What's in the box

Map, Filter, Reduce

- **Map** does something with each element in a list
- **Filter** extracts elements from a list based on some condition
- **Reduce** summarizes elements in a list according to some calculation

Demo

Map, Filter and Reduce

Currying and Partial Application

- Currying: Convert a function that takes multiple parameters into a chain of functions that each take one parameter and return the next function, until the deepest nested function performs the calculation with all the values and returns the result.
- Partial Application: Fixing one or more parameters of a function in curried form, creating a new function with a more specific purpose.

Demo

Manual and Automatic Currying

Demo

Composition

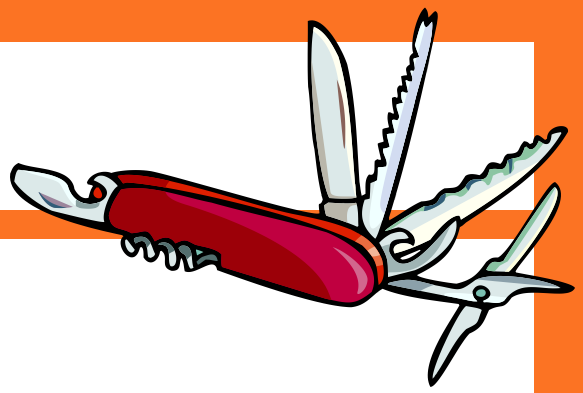
Function Construction

- The idea of creating new functions from existing ones
- Promotes modularization on a function level
- Partial Application is one way to do it
- Composition is another way:
  Assuming `B = f1(A), C = f2(B)`
  → `C = f2(f1(A))`

Combining approaches

- Aim: create function `int sumOfOddNumbers(int)`, based on Reduce
- Using Partial Application to define accumulation strategy for Reduce, as well as algorithm for sequence creation
- Using Composition to allow for easier usage, simplify parameters

Demo



Function Construction

FP in C# - what are the benefits?

- Functional modularization is not easy to get used to, but very rewarding
- Unit testing can benefit from a no-side-effects philosophy
- Programming for scalability is easier, whether you use your own threads, thread pools or toolkits like ParallelFX
- It's easier to get things done – try it yourself!
- BUT: Make sure your team members understand it, too!

Summary

- C# has good support for important Functional Programming ideas
- Some “manual” work is required
- Syntax is sometimes a bit weird
- FP provides Glueing techniques (Currying, Partial Application, Composition) on a function level, introduces an additional level of modularization

Thank you

Please feel free to contact me about the content anytime.

oliver@sturmnet.org