Learning F# and the Functional Point of View

Robert Pickering, LexiFi

http://strangelights.com

Session Objectives

- Why was F# created? Why learn F#?
- A taste of the F# language
  - Especially the functional side!
- A look at some wizzy features F#

Part 1

Why?

Why?

Why?

Why?

Why?

Why?

I'll let you in on a secret: I'm doing F# simply because it's lots and lots of fun. In a very broad sense of the word: functional programming is fun, OO programming with F# is fun, watching people use F# is fun.

One of the wonderful things about F# is that you can actually end up working in your domain. In the zone. With F#, you're not necessarily "just" a programmer! You're likely to also be a probabilistic modeller, or an AutoCAD engineer, or a finance engineer, or a symbolic programmer, or one of many other things.

- Don Syme,

F#’s creator

F# is unique amongst both imperative and declarative languages in that it is the golden middle road where these two extremes converge. F# takes the best features of both paradigms and tastefully combines them in a highly productive and elegant language that both scientists and developers identify with. F# makes programmers better mathematicians and mathematicians better programmers.

- Eric Meijer,
Forward to Expert F#

Functions are much easier to test than operations that have side effects. For these reasons, functions lower risk.
Place as much of the logic of the program as possible into functions, operations that return results with no observable side effects.

- Domain Driven Design, Eric Evans

F# frees you of the fluffy pink hand cuffs of C#

- Amanda Laucher,
Consultant and F# Author

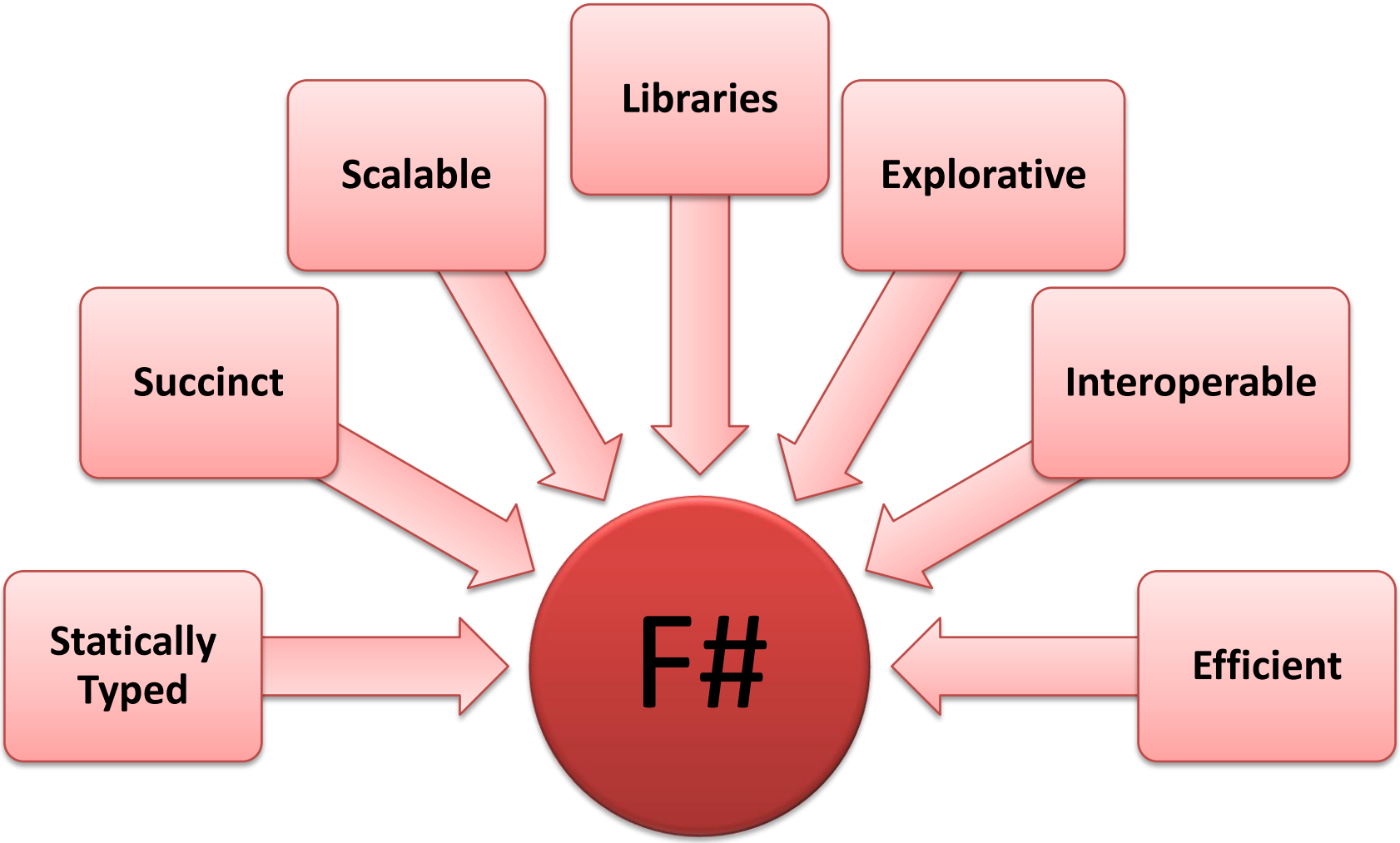
F# - What is it For?

- F# is a General Purpose language

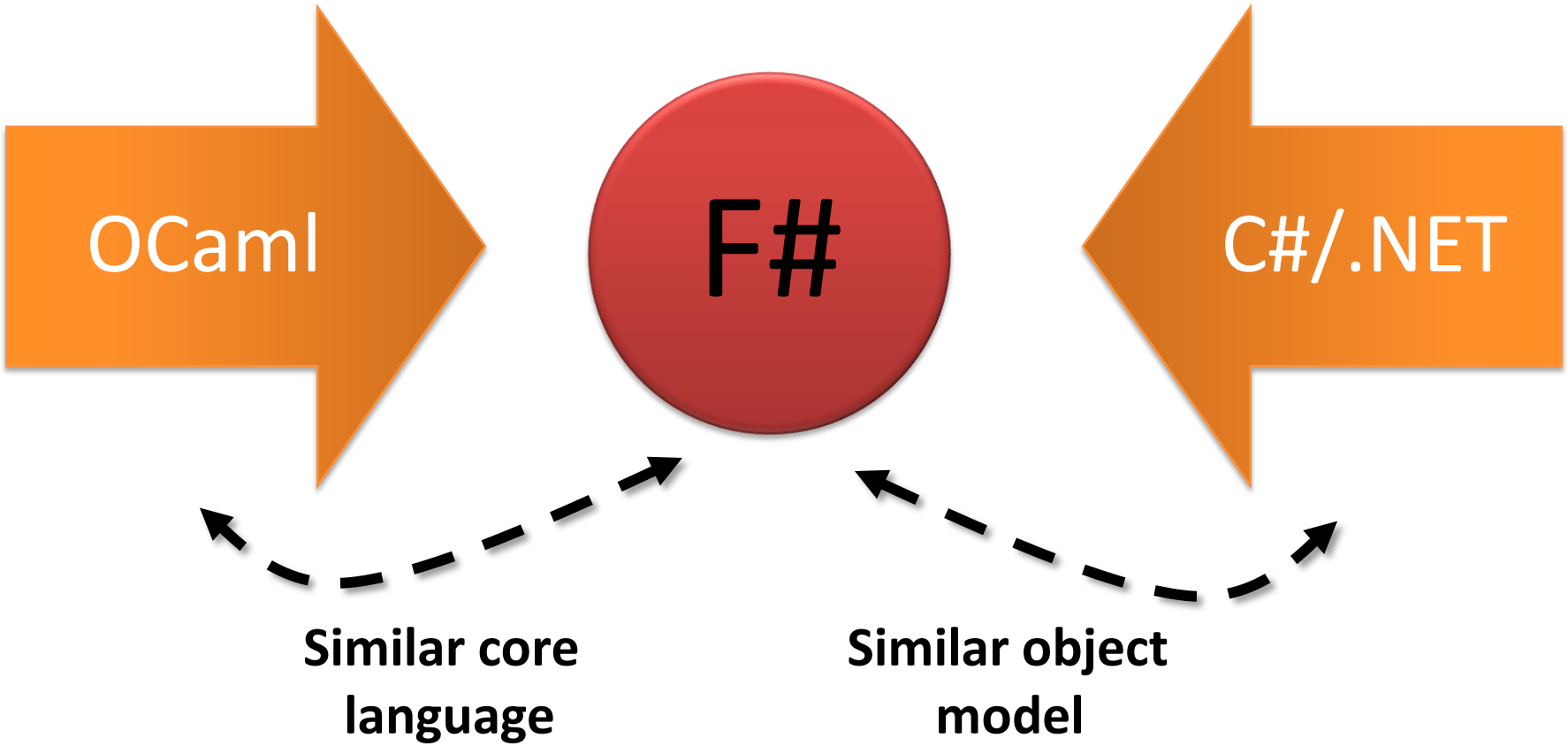
- F# is also “A Bridge Language”
  - “A Language Both Researchers and Developers Can Speak”

- Some important domains
  - Scientific data analysis
  - Data mining
  - Domain-specific modeling

F#: The Combination Counts!



F#: Influences



Part 2

F# the Language ...

... and the Functional Point of View

Hello World

```
printfn "hello world"
```

Values & “let” Bindings

```
let anInt    = 42           // an integer
let aString  = "Stringy"    // a string
let aFloat   = 13.          // a float
let aList    = ["Collect"; "ion"]
                // a list of strings
let aTuple   = "one", 2     // a tuple
let anObject = new FileInfo(@"c:\src.fs")
                // a .NET object
```

Functions

```
// shortcut version
let addTen x = x * 10 + 10


// multi parameters and
// intermediate results
let addThenTimesTwo x y =
    let result = x + y
    result * 2
```

Function as Values

```
// define a list
let list = [1; 2; 3]

// define a function
let addNine x = x + 9

// pass function to "addNine" to
// higher order function "List.map"
let result = List.map addNine list
```

Anonymous Functions

```
// define a list
let list = [1; 2; 3]

// pass function definition directly to
// higher order function "List.map"
let result =
    List.map (fun x -> x + 9) list
```

Everything's an Expression

```
// bind name to "Robert"
// or to "Pickering"
let name =
    if useFirst then "Robert"
    else "Pickering"

// we can bind more than one value at once
let myTuple =
    if useFirst then "Robert", 1
    else "Pickering", 2
```

Loop With Recursion

```
let cMax = complex 1.0 1.0   // Max complex value
let cMin = complex -1.0 -1.0 // Min complex value

let iterations = 18          // Max iterations

let isInMandelbrotSet c0 =
    let rec check n c =
        (n = iterations) // exit if max iterations
                         // reached
        || (cMin < c) && (c < cMax) // exit if escaped
                                    // complex number bounds
        && check (n + 1) ((c * c) + c0) // recurse !
    // start recursion
    check 0 c0
```

Record Types

```
// a "Person" type definition
type Person =
    { FirstName: string;
      LastName: string; }

// an instance of a "Person"
let aPerson =
    { FirstName = "Robert";
      LastName = "Pickering"; }
```

Creating New Records

```
// a single person
let single =
    { FirstName = "Robert";
      LastName = "Pickering"; }

// create record with different
// last name
let married =
    { single with
       LastName = "Townson"; }
```

Union Types – The Option Type

```
// The pre-defined option type
type Option<'a> =
    | Some of 'a
    | None

// constructing options
let someValue = Some 1
let noValue = None

// pattern matching over options
let convert value =
    match value with
    | Some x -> Printf.sprintf "Value: %i" x
    | None -> "No value"
```

Union Types - Trees

```
// a binary tree definition
type BinaryTree<'a> =
    | Node of BinaryTree<'a> * BinaryTree<'a>
    | Leaf of 'a


// walk the tree collection values
let rec collectValues acc tree =
    match tree with
    | Node(ltree, rtree) ->
        // recursively walk the left tree
        let acc = collectValues acc ltree
        // recursively walk the right tree
        collectValues acc rtree
    | Leaf value -> value :: acc
                    // add value to accumulator
```

Using the Tree

```
// define a tree
let tree =
    Node(
        Node(Leaf 1, Leaf 2),
        Node(Leaf 3, Leaf 4))

// recover all values from the leaves
let values = collectValues [] tree
```

.NET Objects

```
open System.Windows.Forms
let form =
    // create a new form instance
    let form = new Form(Text = "Hello")
    // create a couple of controls
    let textBox = new TextBox(Text = "Hello")
    // add the controls
    form.Controls.Add(textBox)
    // return the form
    form

form.Show()
```

Part 3

A brief look at …

… Language Oriented Programming

A Command Line Argument Parse

Ever Written an Arg Parser in C#?

Was it an enjoyable experience?

Or was it more like:

```
static void Main(string[] args) {
      int reps = 0;
      for (int index = 0; index < args.Length; index++) {
        switch (args[index]) {
          case "-reps":
            int nextArg = index + 1;
            if (nextArg < args.Length) {
              if (!int.TryParse(args[nextArg], out reps)) {
                throw new Exception("Agrument not an integer");
              }
            }
            else {
              throw new Exception("Argument expected");
            }
          // ... etc. ...
```

```
let argDefs =
    [ "-outfile",
      Arg.String(fun x -> outfile := x),
      "The output file to be used";
      "-reps",
      Arg.Int(fun x -> reps := x),
      "The number of repetitions";
      "-res",
      Arg.Float(fun x -> res := x),
      "Sets the value resolution"; ]
```

An F# Command-Line Argument Parse

DEMO

... Concurrency

Calling Web Services Asynchronously

Calling Web Services

- Demonstration of calling a web service synchronously and asynchronously using workflows

- This demonstration will analyse:
  - Changes in the code required
  - How the results are effected
  - How is performance effected

Asynchronous Workflows and Web Services

- Synchronous

```
let getAtoms() =
    let pt = new PeriodicTableWS.periodictable()
    let atoms = pt.GetAtoms()
    let atoms = getNodeContentsList atoms
        "/NewDataSet/Table/ElementName"
    atoms
```

- Asynchronous

```
let getAtoms =
    async { let pt = new PeriodicTableWS.periodictable()
            let! atoms = pt.AsyncGetAtoms()
            let atoms = getNodeContentsList atoms
                "/NewDataSet/Table/ElementName"
            return atoms }
```

Where did the “Async” Come From?

- The programmer must add these to the web service proxies

```
type PeriodicTableWS.periodictable with
    member ws.AsyncGetAtoms() =
        Async.BuildPrimitive(ws.BeginGetAtoms,
                             ws.EndGetAtoms)
```

```
type PeriodicTableWS.periodictable with
    member ws.AsyncGetAtomicWeigh(s) =
        Async.BuildPrimitive(s,
                             ws.BeginGetAtomicWeight,
                             ws.EndGetAtomicWeight)
```

Calling a web service

DEMO

Interpreting the Results

Synchronous

```
[.NET Thread 1]Get Element Data List
[.NET Thread 1]Got 112 Elements
[.NET Thread 1]Get Data For: Actinium
[.NET Thread 1]Actinium: 227
[.NET Thread 1]Get Data For: Aluminium
[.NET Thread 1]Aluminium: 26.9815
[.NET Thread 1]Get Data For: Americium
[.NET Thread 1]Americium: 243
[.NET Thread 1]Get Data For: Antimony
[.NET Thread 1]Antimony: 121.75
[.NET Thread 1]Get Data For: Argon
[.NET Thread 1]Argon: 39.948
[.NET Thread 1]Get Data For: Arsenic
...
...
```

Asynchronous

```
[.NET Thread 1]Get Element Data List
[.NET Thread 6]Got 112 Elements
[.NET Thread 11]Get Data For: Actinium
[.NET Thread 11]Get Data For: Aluminium
[.NET Thread 10]Get Data For: Americium
[.NET Thread 11]Get Data For: Antimony
[.NET Thread 11]Get Data For: Argon
...
[.NET Thread 6]Actinium: 227
[.NET Thread 6]Aluminium: 26.9815
[.NET Thread 6]Americium: 243
[.NET Thread 6]Antimony: 121.75
[.NET Thread 6]Arsenic: 74.9216
[.NET Thread 6]Astatine: 210
...
```

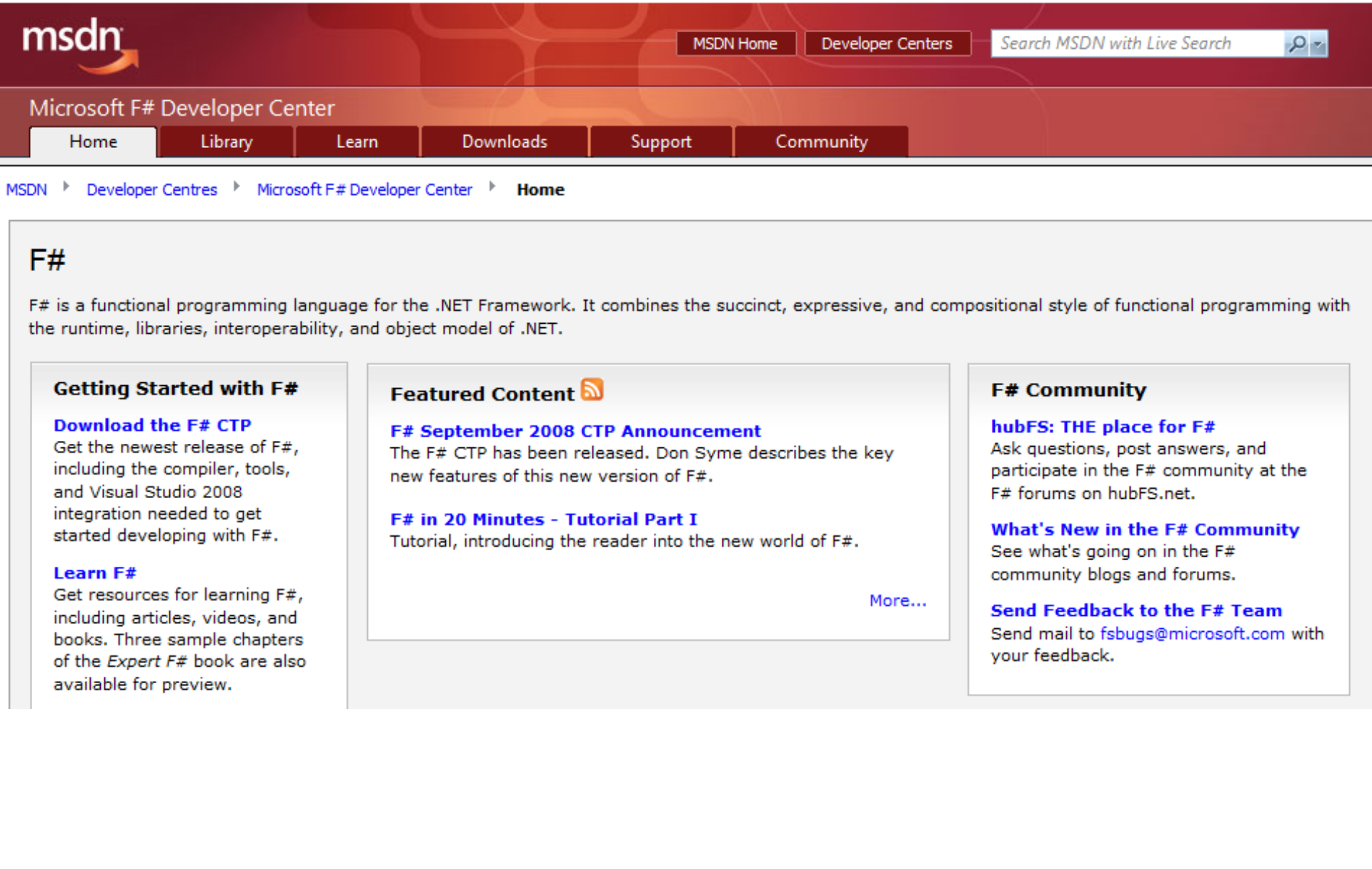
The Timings

Synchronous		Asynchronous	
Real	**CPU**	**Real**	**CPU**
48.976	00.187	24.571	00.142
48.270	00.109	24.432	00.156
54.240	00.078	24.641	00.218

Part 4

The End Bit

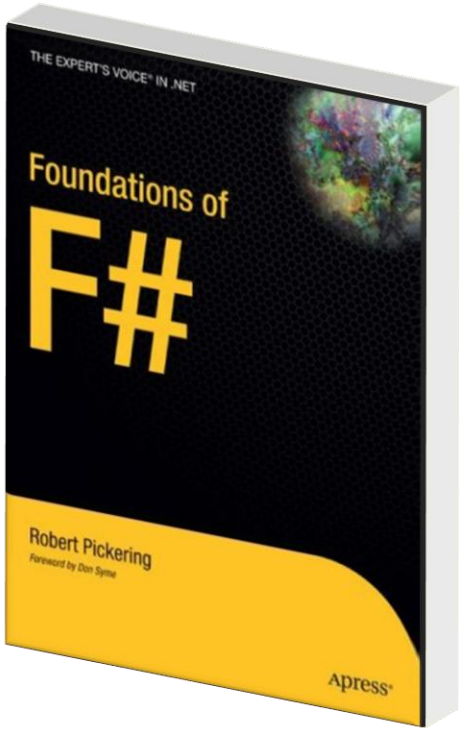
msdn.microsoft.com/fsharp/

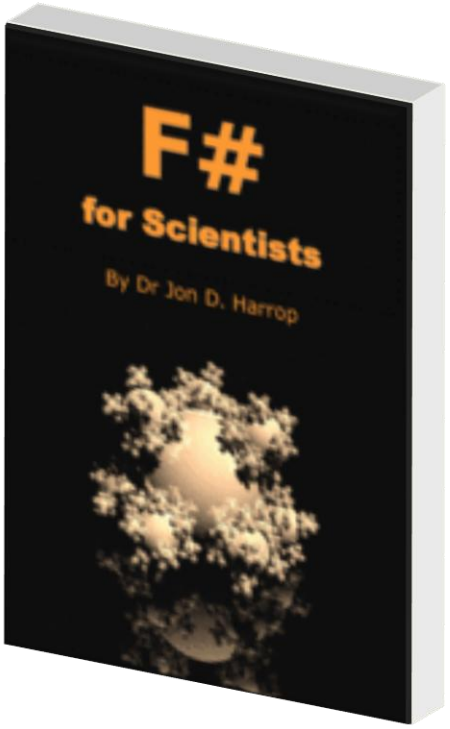


F# Resources

- MDSN Resource center: http://msdn.microsoft.com/fsharp/
- User forums: http://cs.hubfs.net/forums
- Blogs (there are lots of others!):
  - http://blogs.msnd.com/dsyme
  - http://strangelights.com/blog
- Samples on the web:
  - http://code.msdn.microsoft.com/fsharpsamples
  - http://code.google.com/hosting/search?q=label:fsharp
  - http://codeplex.com/Project/ProjectDirectory.aspx?TagName=F%23
- Source available with the distribution: %ProgramFiles%\FSharp-1.9.6.2\source

Books about F#





Questions? !?