F# Succinct, Expressive, Functional

The F# Team
Microsoft Developer Division
Microsoft Research
Topics

• What is F# about?
• Some Simple F# Programming
• A Taste of Parallel/Reactive with F#
What is F# about?

Or: Why is Microsoft investing in functional programming anyway?
Simplicity
Economics
Programmer Productivity
Simplicity
More Noise Than Signal!
Pleasure

type Command = Command of (Rover -> unit)

let BreakCommand = Command(fun rover -> rover.Accelerate(-1.0))

let TurnLeftCommand = Command(fun rover -> rover.Rotate(-5.0<degs>))

Pain

abstract class Command
{
    public virtual void Execute();
}
abstract class MarsRoverCommand : Command
{
    protected MarsRover Rover { get; private set; }
    public MarsRoverCommand(MarsRover rover)
    {
        this.Rover = rover;
    }
}
class BreakCommand : MarsRoverCommand
{
    public BreakCommand(MarsRover rover) : base(rover)
    {
    }
    public override void Execute()
    {
        Rover.Rotate(-5.0);
    }
}
class TurnLeftCommand : MarsRoverCommand
{
    public TurnLeftCommand(MarsRover rover) : base(rover)
    {
    }
}
Pleasure

```csharp
public abstract class Expr { }
public abstract class UnaryOp : Expr
{
    public Expr First { get; private set; }  
    public UnaryOp(Expr first)
    {
        this.First = first;
    }
}

public abstract class BinExpr : Expr { }
{
    public Expr First { get; private set; }  
    public Expr Second { get; private set; }  

    public BinExpr(Expr first, Expr second)
    {
        this.First = first;
        this.Second = second;
    }
}

public class TrueExpr : Expr { }

public class And : BinExpr
{
    public And(Expr first, Expr second) : base(first, second)
    {
    }
}
```

Pain

```csharp
type Expr =
 | True
 | And of Expr * Expr
 | Nand of Expr * Expr
 | Or of Expr * Expr
 | Xor of Expr * Expr
 | Not of Expr
```

```csharp
http://stepheneasey.wordpress.com/tag/c/
```
Pleasure

let rotate (x,y,z) = (z,x,y)

let reduce f (x,y,z) = f x + f y + f z

Pain

Tuple<V,T,U> Rotate(Tuple<T,U,V> t)
{
    return new Tuple<V,T,U>(t.Item3,t.Item1,t.Item2);
}

int Reduce(Func<T,int> f,Tuple<T,T,T> t)
{
    return f(t.Item1) + f(t.Item2) + f(t.Item3);
}
You Can Interoperate With Everything
Everything Can Interoperate With You
Economics
Fun!
F#: Influences

OCaml: Similar core language

F#: Similar object model

C#/.NET: Similar object model
F#: Combining Paradigms

I've been coding in F# lately, for a production task.

F# allows you to move smoothly in your programming style... I start with pure functional code, shift slightly towards an object-oriented style, and in production code, I sometimes have to do some imperative programming.

I can start with a pure idea, and still finish my project with realistic code. You're never disappointed in any phase of the project!

F#: The Combination Counts!

- Statically Typed
- Succinct
- Scalable
- Libraries
- Explorative
- Interoperable
- Efficient
F# in More Detail
Quick Tour

Comments

// comment

(* comment *)

/// XML doc comment

let x = 1
Quick Tour

**Overloaded Arithmetic**

- \( x + y \)  
  Addition
- \( x - y \)  
  Subtraction
- \( x * y \)  
  Multiplication
- \( x / y \)  
  Division
- \( x \% y \)  
  Remainder/modulus
- \(-x\)  
  Unary negation

**Booleans**

- \( \text{not } expr \)  
  Boolean negation
- \( expr \&\& expr \)  
  Boolean “and”
- \( expr \|\| expr \)  
  Boolean “or”
Orthogonal & Unified Constructs

• Let “let” simplify your life…

```csharp
let data = (1, 2, 3)

let f(a, b, c) =
    let sum = a + b + c
    let g(x) = sum + x * x
    g(a), g(b), g(c)
```

Bind a static value

Bind a static function

Bind a local value

Bind a local function

Type inference. The safety of C# with the succinctness of a scripting language.
Demo: Let’s WebCrawl...
Orthogonal & Unified Constructs

- Functions: like delegates + unified and simple

```plaintext
(fun x -> x + 1)
let f(x) = x + 1
(f,f)
val f : int -> int
```

One simple mechanism, many uses

Anonymous function value

Declare a function value

A pair of function values

A function type

Predicate = 'a -> bool
Send = 'a -> unit
ThreadStart = unit -> unit
Comparer = 'a -> 'a -> int
Hasher = 'a -> int
Equality = 'a -> 'a -> bool

One simple mechanism, many uses
let f x = x+1

let pair x = (x,x)

let fst (x,y) = x

let data = (Some [1;2;3], Some [4;5;6])

match data with
| Some(nums1), Some(nums2) -> nums1 @ nums2
| None, Some(nums)   -> nums
| Some(nums), None   -> nums
| None, None         -> failwith "missing!"
F# - Functional

List.map    Seq.fold
Array.filter Lazy    Set.union
Map    LazyList   Events   Async...

[ 0..1000 ]
[ for x in 0..10 -> (x, x * x) ]
[| for x in 0..10 -> (x, x * x) |
seq { for x in 0..10 -> (x, x * x) }

Range Expressions
List via query
Array via query
IEnumerable via query
Immutability the norm...

```plaintext
// Part 1. Adjust some constants

let PI = 3.141592654

PI <- 4.0

// This value is not mutable.
```

Data is immutable by default

```plaintext
type Person =
    { Name : string;
      Birth: DateTime }

let bob =
    { Name = "bob";
      Birth = DateTime(15,8,1980) }

// OK
let bobJunior =
    { bob with Birth = DateTime(23,5,2006) }

// Not OK!
bob.Birth <- DateTime(23,5,2006)
```

Values may not be changed

```plaintext

Not Mutate

Copy & Update
```

```sql
1 Error
0 Warning
```

```plaintext
Description          File  Line  Column
error FS0005: This field is not mutable        test.fs 18     1
```
In Praise of Immutability

• Immutable objects can be relied upon

• Immutable objects can transfer between threads

• Immutable objects can be aliased safely

• Immutable objects lead to (different) optimization opportunities
open System.IO

let rec allFiles(dir) =  
[ for file in Directory.GetFiles(dir) do  
  yield file  
  for sub in Directory.GetDirectories(dir) do  
    yield! allFiles(sub)  
]  

allFiles(@"C:\Demo")
open System.IO

let rec allFiles(dir) = seq
   { for file in Directory.GetFiles(dir) do yield file
     for sub in Directory.GetDirectories(dir) do yield! allFiles(sub) }

allFiles(@"C:\WINDOWS") |> Seq.take 100 |> show
Looks Weakly typed? Maybe Dynamic?
Typed
Yet rich, dynamic
Efficient
Yet succinct
Untyped
Interpreted
Reflection
Invoke
F#
Objects

Class Types

type ObjectType(args) =

  let internalValue = expr
  let internalFunction args = expr
  let mutable internalState = expr

  member x.Prop1 = expr
  member x.Meth2 args = expr

Constructing Objects

new FileInfo(@"c:\misc\test.fs")
type Vector2D(dx:double, dy:double) =

    member v.DX = dx

    member v.DY = dy

    member v.Length = sqrt(dx*dx + dy*dy)

    member v.Scale(k) = Vector2D(dx*k, dy*k)
F# - Objects + Functional

```fsharp
type Vector2D(dx:double,dy:double) =

    let norm2 = dx*dx+dy*dy

    member v.DX = dx

    member v.DY = dy

    member v.Length = sqrt(norm2)

    member v.Norm2 = norm2

Internal (pre-computed) values and functions
```
F# - Objects + Functional

```fsharp
type HuffmanEncoding(freq: seq<char*int>) =

...< 50 lines of beautiful functional code...

member x.Encode(input: seq<char>) =
    encode(input)

member x.Decode(input: seq<char>) =
    decode(input)
```
type Vector2D(dx:double, dy:double) =

    let mutable currDX = dx
    let mutable currDY = dy
    member v.DX = currDX
    member v.DY = currDY
    member v.Move(x, y) =
        currDX <- currDX + x
        currDY <- currDY + y
F# and adCenter

- 4 week project, 4 machine learning experts
- 100 million probabilistic variables
- Processes 6TB of training data
- Real time processing
AdPredict: What We Observed

- Quick Coding
- Agile Coding
- Scripting
- Performance
- Memory-Faithful
- Succinct
- Symbolic
- .NET Integration

F#’s powerful type inference means less typing, more thinking

Type-inferred code is easily refactored

“Hands-on” exploration.

Immediate scaling to massive data sets

mega-data structures, 16GB machines

Live in the domain, not the language

Schema compilation and “Schedules”

Especially Excel, SQL Server

Especially Excel, SQL Server
Smooth Transitions

• Researcher’s Brain ➔ Realistic, Efficient Code
• Realistic, Efficient Code ➔ Component
• Component ➔ Deployment
UNITS OF MEASURE
1985

Mirror on underside of shuttle

Big mountain in Hawaii

SDI experiment: The plan
1985

SDI experiment:
The reality
1985

The reality
Metric mishap caused loss of NASA orbiter

By Robin Lloyd
CNN Interactive Senior Writer

(CNN) -- NASA lost a $125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation, according to a review finding released Thursday.

The units mismatch prevented navigation information from transferring between the Mars Climate Orbiter spacecraft team in at Lockheed Martin in Denver and the flight team at NASA's Jet Propulsion Laboratory in Pasadena, California.
Metric mishap caused loss of NASA orbiter

September 20, 2000
Web posted at: 4:21 p.m. EDT (2021 GMT)

In this story:
Metric system used by NASA for many years
Error points to nation’s conversion lag

By Robin Lloyd
CNN Interactive Senior Writer

(CNN) -- NASA lost a $125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency’s team used the more conventional metric system for a key spacecraft operation, according to a review finding released Thursday.

The units mismatch prevented navigation information from transferring between the Mars Climate Orbiter spacecraft team in at Lockheed Martin in Denver and the flight team at NASA’s Jet Propulsion Laboratory in Pasadena, California.
let EarthMass = 5.9736e24<kg>

// Average between pole and equator radii
let EarthRadius = 6371.0e3<m>

// Gravitational acceleration on surface of Earth
let g = PhysicalConstants.G * EarthMass / (EarthRadius * EarthRadius)
F# Async/Parallel
async { ... }

• For users:
  
  You can run it, but it may take a while

  Or, your builder says...

  OK, I can do the job, but I might have to talk to someone else about it. I’ll get back to you when I’m done
The F# Approach

- Good Architecture
  - Know your techniques
  - Know your requirements
  - Know your limits (CPU, disk, network, latency)

- Translate Good Architecture into Good Code with F#
  - A great platform
  - A massive increase in isolation and immutability
  - A massive reduction in mutation

In parallel programming, F# is a **power tool** for good architects and good developers.
async { ... }

You're actually writing this (approximately):

```csharp
async { let! image = ReadAsync "cat.jpg"
    let image2 = f image
    do! writeAsync image2 "dog.jpg"
    do printfn "done!"
    return image2 }
```

Asynchronous "non-blocking" action

Continuation/Event callback
async { ... }

• Built on a much more general mechanism called “computation expressions”

seq { ... } (queries/sequences)

eventStream { ... } (queries over event streams)

parser { ... } (parser combinators)

resumable { ... } (resumptions)
8 Ways to Learn

- FSI.exe
- Samples Included
- Go to definition
- Lutz’ Reflector

- [http://cs.hubfs.net](http://cs.hubfs.net)
- Codeplex Fsharp Samples
- Books
- ML
Books about F#

Visit www.fsharp.net
Getting F#

• September 2008: CTP released
  
  F# will be a supported language in Visual Studio 2010
  
• Next stop: Visual Studio 2010 Beta 1
  
  Look for it soon!
Questions & Discussion