

Five Reasons to Move from C# to F#



JORGE FIORANELLI
LEAD CONSULTANT

@JORGEFIORANELLI

Who am I?

Readify Lead Consultant - Sydney, Australia

F# Sydney User Group (fsharpsydney.com)

F# Workshop (fsharpworkshop.com)

F# | > I ❤️

Why F#?

Why F#?

.Net

Functional-First

Full Support in VS

Interoperable

Multi Platform

Open Source

Great Syntax



fsharp.org

Reason #1:

Powerful Functions

Reason #1: Powerful Functions

```
int Add(int x, int y)
{
    return x + y;
}
```

Func<int,int,int>



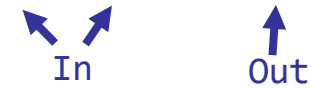
Partial Application

Pipelining

Composition

```
let add x y = x + y
```

int -> int -> int



```
let addOne = add 1 //addOne: int -> int
let result = addOne 2 //result = 3
```

```
let result = customer
    |> promoteToVip
    |> increaseCredit
```

```
let addThree = addOne >> addTwo
let result = addThree 1 //result = 4
```

let instead of
let parameters
no parentheses
no return
concess

Reason #1: Powerful Functions

```
[Fact]  
void Should_process_orders_when_they_are_approved()
```

```
[<Fact>]  
let ``Should process orders when they are approved`` () =
```

Run All | Run... ▾ | Playlist: All Tests ▾

▲ CSharp (1)

✔ Should_process_orders_when_they_are_approved

▲ FSharp (1)

✔ Should process orders when they are approved

Reason #1: Powerful Functions

F# |> I ❤️



I ❤️ F#

Reason #2: Great Types

Reason #2: Great Types

```
class Customer
{
    private readonly string name;

    public Customer(string name)
    {
        this.name = name;
    }

    public string Name
    {
        get { return name; }
    }
}
```

```
type Customer(name: string) =
    member this.Name = name
```

```
type Customer = { Name: string }
```

Class
type priority of
conflict
Record

Immutability
Structural Equality
Type Inference

Reason #2: Great Types

Tuples

```
Tuple<string, int> myTuple = GetNameAndAge(3);  
string name = myTuple.Item1;  
int age = myTuple.Item2;
```

```
let name, age = getNameAndAge 3  
let success, value = Int32.TryParse "42"
```

Options

```
int GetCustomerAgeById(int id)  
Nullable<int> GetCustomerAgeById(int id)  
  
Customer GetCustomerById(int id)  
Nullable<Customer> GetCustomerById(int id)
```

```
getCustomerAgeById (id: int) : int  
getCustomerAgeById (id: int) : Option<int>  
  
getCustomerById (id: int) : Customer  
getCustomerById (id: int) : Option<Customer>
```

Reason #2: Great Types

Units of Measure

```
var meters = 200;  
var kilometers = 30;  
  
var total = meters + kilometers; // 230!
```

```
[<Measure>] type m  
[<Measure>] type km  
  
let meters = 200<m>  
let kilometers = 30<km>  
  
let total = meters + kilometers // Error!
```

Object Expressions

```
var product = new { Name = "Product1" };  
return product; // ?
```

```
let product = { new IProduct with  
    member this.Name = "Product1"  
    member this.Show () =  
        printfn "%s" this.Name }
```

Reasons

#1

Powerful
Functions

#2

Great
Types

Reason #3:

Pattern Matching

Reason #3: Pattern Matching

```
enum Result
{
    Success,
    Error
}

void Show(Result result)
{
    switch (result)
    {
        case Result.Success:
            Console.WriteLine(":)");
            break;
        case Result.Error:
            Console.WriteLine(":(");
            break;
    }
}
```

```
type Result = Discriminated
| Success of quotient: int * remainder: int
| Error of Exception Union

let ([Pieces]_|) result =
    match result with
    | Success (q,0) when q > 1 -> Some q
    | _ -> None

let show result =
    match result with
    | [Pieces] q -> printfn "%i pieces" q
    | Success (q,r) -> printfn "%i %i" q r
    | Error e -> printfn "%s" e.Message
```

Reasons

#1

Powerful
Functions

#2

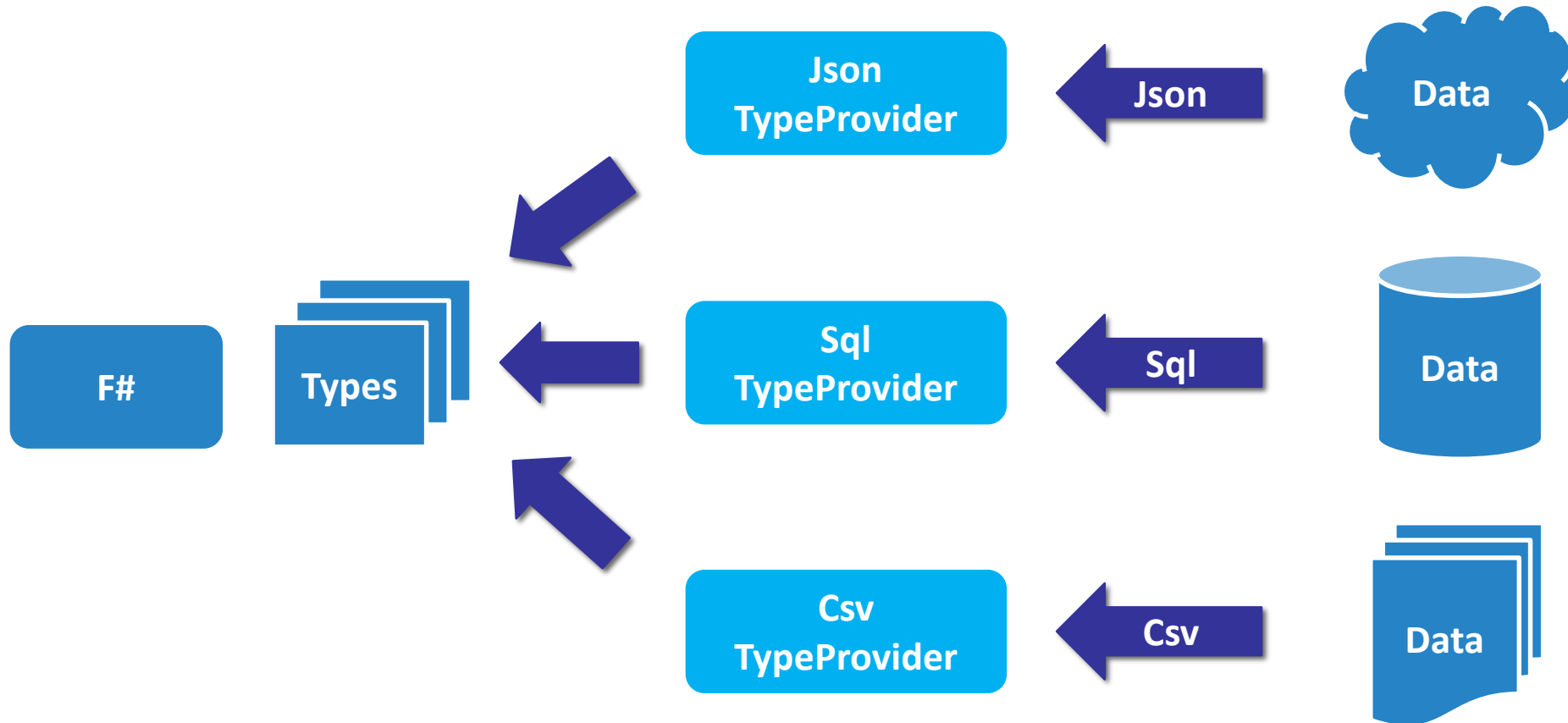
Great
Types

#3

Pattern
Matching

Reason #4: Type Providers

Reason #4: Type Providers



Reason #4: Type Providers

```
type Customer = CsvProvider<"sample.csv">  
let customers = Customer.Load "real.csv"  
  
customers.Rows  
|> Seq.iter (fun r -> printfn "%s: %g" r.Name r.Credit)
```

Id,Name,IsVip,Credit
1,Customer1,false,0.0

Id,Name,IsVip,Credit
1,Customer1,false,0.0
2,Customer2,false,10.0
3,Customer3,false,30.0
4,Customer4,true,50.0

```
type EntityConnection = SqlEntityConnection <ConnectionString=cs, Pluralize=true>  
let context = EntityConnection.GetDataContext ()  
let customers = query { for customer in context.Customers do  
    select customer.Name }
```

Data

Reason #4: Type Providers

Csv

OData

R

EF

Linq2Sql

Freebase

Json

SQL

WMI

Azure Storage

SQLClient

Hadoop

Xml

Reflection

WSDL

Excel

World Bank

and more...

Reasons

#1

Powerful
Functions

#2

Great
Types

#3

Pattern
Matching

#4

Type
Providers

Reason #5:

Computation Expressions

Reason #5: Computation Expressions

```
let x = isEven 1
let y = isEven 2
x && y
```

```
Value: false
Value: true
Result: false
```



```
log { let! x = isEven 1
      let! y = isEven 2
      return x && y }
```

```
Value: false
Value: true
Result: false
```

```
type LogBuilder() =
  member this.Bind! (value, continuation) =
    printfn "Value: %b" value
    continuation value
  member this.Return! value =
    printfn "Result: %b" value
    value

let log = LogBuilder()
```



Reason #5: Computation Expressions

Async
Expression

```
let readAsync (file: string) = async {  
    use reader = new StreamReader(file)  
    let! content = reader.AsyncReadToEnd ()  
    content |> printfn "Content: %s" }
```

EF Type Provider
Query Expression

```
let result = query {  
    for student in db.Student do  
    where (student.StudentID > 4)  
    select student }
```

MBrace
Cloud

```
let first = cloud { return 15 }  
let second = cloud { return 27 }  
cloud {  
    let! x = first  
    let! y = second  
    return x + y }
```


Reasons

#1

Powerful
Functions

Partial Application
Pipelining
Composition
Spaces in Names

#2

Great
Types

Classes
Records
Tuples
Options
Units of Measure
Object Expressions

#3

Pattern
Matching

Discriminated Unions
Pattern Matching
Active Patterns

#4

Type
Providers

CSV
EF

#5

Computation
Expressions

Log
Async
Query
Cloud

Thank you!

Jorge Fioranelli
@jorgefioranelli

[github.com/jorgef/
fivereasons](https://github.com/jorgef/fivereasons)

Resources



fsharp.org



Real-World Functional Programming
By Tomas Petricek



Try F#: tryfsharp.org



Scott Wlaschin fsharpforfunandprofit.com
fpbridge.co.uk/why-fsharp.html



Skills Matter: skillsmatter.com (tag: f#)



Community for F#: c4fsharp.net



F# Workshop: fsharpworkshop.com