



CLOUD **FOUNDRY**
S U M M I T

RUNNING AT SCALE

APRIL 18-20, 2018 | BOSTON



CLOUD **FOUNDRY**
S U M M I T

What the F#nc

Tim Hess

Senior Software Engineer, Pivotal

@timhessWI

Fire Exit Announcement

- Please note the locations of the surrounding emergency exits & located the nearest lit EXIT sign to you
- In the event of a fire alarm or other emergency, please calmly exit to the public concourse area
- Emergency exit stairwells leading to the outside of this facility are located along the public concourse
- For your safety in an emergency, please follow the directions of the Public Safety Staff



What this talk is not

- Getting into the depths of the language
- An enumeration of features
- Advanced math
- Touching monads



Why learn a new language?

- Learning F# is more about personal/professional development
- “A language that doesn’t affect the way you think about programming, is not worth knowing”

Alan Perlis

- Some F# features have crossed over to C#
 - Generics, async, auto-properties, pattern matching...
 - <http://blog.ploeh.dk/2015/04/15/c-will-eventually-get-all-f-features-right/>



What is F#?

- Part of the .NET family
- Developed by Microsoft Research
 - First released in 2005
- Cross-platform since 2010
 - .NET Core, .NET Framework
 - Xamarin for mobile apps
 - Use [WebSharper](#) and [Fable](#) to write F# and get JavaScript
- Open Source
 - github.com/fsharp
- Friendly, Active community
 - #fsharp on Twitter
 - [Slack team](#)
 - fsharp.org



How does F# compare to C#?

- Both are strongly typed
- Both can interoperate
- Different syntax
- Different defaults
- Different philosophy
- Functional-first
- Type Inference
- Other features not covered today but worth checking out:
 - [record types](#), [units of measure](#)



Basic Shopping Cart Class in C#

```
using System;
using System.Collections.Generic;

namespace funstore.shared.models
{
    public class Cart
    {
        private List<CartItem> _contents;

        public Cart()
        {
            Id = Guid.NewGuid();
            _contents = new List<CartItem>();
        }

        public Guid Id;

        public bool AddItem(CartItem item)
        {
            _contents.Add(item);
            return true;
        }

        public bool UpdateItem(CartItem item)
        {
            var toUpdate = _contents.Find(x => x.CartItemId == item.CartItemId);
            toUpdate.Count = item.Count;
            return true;
        }

        public bool RemoveItem(CartItem item)
        {
            return _contents.Remove(item);
        }

        public bool Clear()
        {
            _contents.Clear();
            return true;
        }
    }
}
```

- 42 lines
- 900 characters



Basic Shopping Cart Class in F#

```
namespace funstore.service.cart

open funstore.shared.models
open System
open System.Collections.Generic

type Cart() =
    member this.Id = Guid.NewGuid()

    member this.Contents = new System.Collections.Generic.List<CartItem>()

    member this.AddItem item =
        this.Contents.Add(item)
        true

    member this.UpdateItem (item:CartItem) =
        let toUpdate = this.Contents.Find(fun f -> f.CartItemId = item.CartItemId)
        toUpdate.Count <- item.Count
        true

    member this.RemoveItem item =
        this.Contents.Remove(item)

    member this.Clear =
        this.Contents.Clear()
        true
```

- 26 Lines
- 684 characters
- No curly braces
 - Whitespace instead
- No semi-colons
- Many type declarations omitted
 - Type inference FTW
- No access modifiers
- **type** and **let** instead of **class** and **var**
- Don't need return



Side by Side

```
using System;
using System.Collections.Generic;

namespace funstore.shared.models
{
    public class Cart
    {
        private List<CartItem> _contents;

        public Cart()
        {
            Id = Guid.NewGuid();
            _contents = new List<CartItem>();
        }

        public Guid Id;

        public bool AddItem(CartItem item)
        {
            _contents.Add(item);
            return true;
        }

        public bool UpdateItem(CartItem item)
        {
            var toUpdate = _contents.Find(x => x.CartItemId == item.CartItemId);
            toUpdate.Count = item.Count;
            return true;
        }

        public bool RemoveItem(CartItem item)
        {
            return _contents.Remove(item);
        }

        public bool Clear()
        {
            _contents.Clear();
            return true;
        }
    }
}
```

```
namespace funstore.service.cart

open funstore.shared.models
open System
open System.Collections.Generic

type Cart() =
    member this.Id = Guid.NewGuid()

    member this.Contents = new System.Collections.Generic.List<CartItem>()

    member this.AddItem item =
        this.Contents.Add(item)
        true

    member this.UpdateItem (item:CartItem) =
        let toUpdate = this.Contents.Find(fun f -> f.CartItemId = item.CartItemId)
        toUpdate.Count <- item.Count
        true

    member this.RemoveItem item =
        this.Contents.Remove(item)

    member this.Clear =
        this.Contents.Clear()
        true
```



Reset to Default

- Nullability
 - In C#, just about anything could be null
 - In F#, you need to add [`<AllowNullLiteralAttribute>`] for null to be an option
- Immutability
 - In F#, you must declare variable as mutable
 - `let mutable myAwesomeThing = "boomerang"`
 - Different operator for assignment
 - `myAwesomeThing <- "dune buggy"`
 - Side affects:
 - Generally won't specify access modifiers
 - Don't need to worry about unexpected mutations



Reset to Default

- Structural equality
 - Don't *need* to override `.Equals()` or `.GetHashCode()`

```
type awesomeThing = { Name:string, Color:string }  
let awesome1 = { Name:"boomerang", Color:"blue"}  
let awesome2 = { Name:"boomerang", Color:"blue"}  
printfn "awesome1=awesome2 is %A" (awesome1=awesome2)
```

this is a record type



Philosophical Differences

- F# and C# have different origin stories...
 - C# comes from C
 - F# comes from ML or MetaLanguage
- F# intends to help you write predictable code
 - Immutability by default
 - Not nullable by default
 - Can't compare objects of different type



Pipelining

- Passing the output of a function to another is trivial
- One of the most commonly used symbols in F#
 - `|>` to pass forward
 - `<|` to pass backward
- Implicit returns make piping functions easy



Pipelining

```
let square x = x * x
```

```
let isEven x = x % 2 = 0
```

```
let numbers = [0..5]
```

```
let evens = List.filter isEven numbers
```

```
let result = List.map square evens
```

```
AssertEquality result [0;4;16]
```



Pipelining

```
let square x = x * x
```

```
let isEven x = x % 2 = 0
```

```
let numbers = [0..5]
```

```
let result = List.map square (List.filter isEven numbers)
```

```
AssertEquality result [0;4;16]
```



Pipelining

```
let square x = x * x
```

```
let isEven x = x % 2 = 0
```

```
let result =  
    [0..5]  
    |> List.filter isEven  
    |> List.map square
```

```
AssertEquality result [0;4;16]
```



Type Inference

- “The idea of type inference is that you do not have to specify the types of F# constructs except when the compiler cannot conclusively deduce the type.” – [Microsoft](#)
- Less typing
- More focus on what matters



Type Inference

```
// C#  
public static IEnumerable<TSource> Distinct<TSource>(  
    this IEnumerable<TSource> source, IEqualityComparer<TSource> comparer)  
{  
    ...  
}
```

```
// F#  
let Distinct source comparer =  
    ...
```



Additional Resources

- Tons of F# content by [@ScottWlaschin](#)
 - <http://fsharpforfunandprofit.com/>
- F# Koans (fill in gaps to get tests to pass)
 - <https://github.com/ChrisMarinos/FSharpKoans>
- F# Workshop
 - <http://www.fsharpworkshop.com/>
- Learn/Run F# in your browser
 - <http://www.tryfsharp.org/> | <https://dotnetfiddle.net/>
- Many additional links
 - <http://fsharp.org/learn.html>



Source Code and Contact Info

- Source Code: <https://github.com/TimHess/what-the-func>
- Twitter: [@timhessWI](https://twitter.com/timhessWI)
- GitHub: [@TimHess](https://github.com/TimHess)
- SteeltoeOSS Slack <https://slack.steeltoe.io/>

