Embracing a Functional Style

Mark Heath
SOFTWARE DEVELOPER

@mark_heath   www.markheath.net

Functional Programming

C# is an "object oriented" language

Primary building block is classes

Classes contain their own data

Functional programming

Primary building block is functions

Functions without side effects

Can seem intimidating at first

Requires a different way of approaching problems

Overview

LINQ and Functional Programming

- You're already doing it!
- Start applying it more broadly

Key Functional Programming Concepts

- How they relate to LINQ
- How to use them elsewhere

Declarative Code

Focuses on what we want to do not how to do it

Makes our intent clear

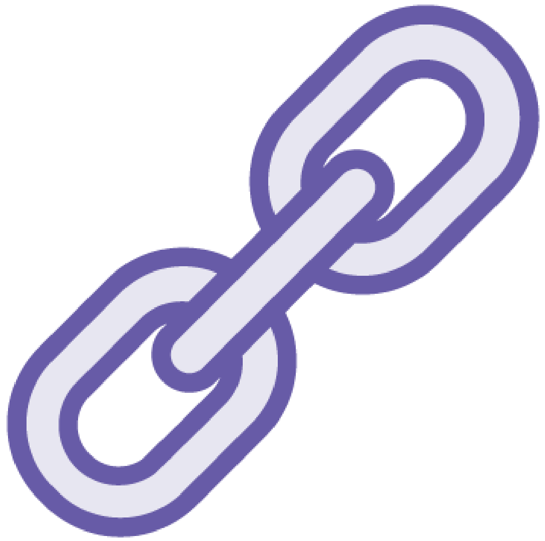
```
orders.Max(o => o.Amount)
```

```
int maxAmount = 0;
foreach (var order in orders)
{
    if (order.Amount > maxAmount)
    {
        maxAmount = order.Amount;
    }
}
```

Can be applied outside of LINQ

```
myCanvas.AddSquare(Color.Green, 100)
        .AddCircle(Color.Red, 40, Align.Center)
```

Chaining Functions



LINQ pipelines

Chain together many simple functions

Can be applied outside of LINQ

Use extension methods to create fluent interfaces

```
LoadFile("sound1.mp3")
    .WithVolume(0.5)
    .FadeIn(2.0)
    .Take(30)
    .FadeOut(2.0)
    .Concat(LoadFile("sound2.mp3"));
```

Helpful in many problem domains

Higher Order Functions

Take a function as a parameter, or return a function

Many examples in LINQ

```
orders.Where(o => o.Amount > 100).Select(o => o.Id)
```

Can be applied outside of LINQ

```
long Time(Action action)
{
    var s = new Stopwatch();
    s.Start();
    action();
    s.Stop();
    return s.ElapsedMilliseconds;
}
```

```
var duration = Time(() => MyFunc());
```

Being Lazy

LINQ supports being lazy

Deferred execution

Can be applied outside of LINQ

e.g. Lazy<T>

```
var lazyAddress =
    new Lazy<Address>(() => GetBillingAddress());
```

```
// will call GetBillingAddress or return cached value
lazyAddress.Value
```

Avoiding Side Effects

Pure functions

Output depends entirely on input parameters

Don’t cause any “side effects”

Do not access or modify global state

Take immutable parameters

Simply return data

Benefits

Thread-safe

More testable

Easier to reason about

Programming with Immutable Objects

LINQ encourages immutability

```
myList.Where(x => x.Value > 50)
```

The list we are filtering remains unchanged

Anonymous objects are immutable

Select method should return a new object

Don’t modify the object you were passed

“Side effects” are inevitable

e.g. disk & network access, user interface

Isolate methods with side effects

Keep “business logic” in pure functions

Allows them to be covered by unit tests

Course Summary

More Effective LINQ

- Simple declarative code
- Solve complex problems with pipelines
- Create clean and readable code
- Extend LINQ yourself or with MoreLINQ
- Benefits of laziness
- Optimizing performance
- Debug, test and handle exceptions

Bonus Content

More LINQ Challenges

- http://markheath.net/category/linq-challenge
- Solutions provided in C# and F#

Advent of Code Solution Videos

- 25 daily programming challenges
- Can be solved with LINQ & MoreLINQ
- http://adventofcode.com
- My solutions: http://tinyurl.com/aoclinq