Justin Etheredge – www.codethinked.com

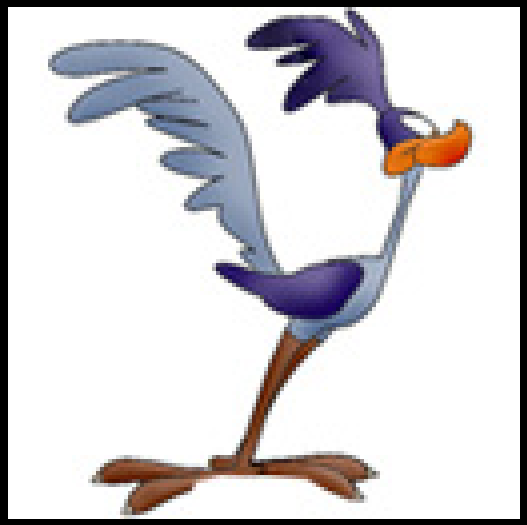
# Functional Programming Features in C# 3.0

# Disclaimer:

I am not an expert in anything, nor do I claim to be. Please use critical thinking when listening to anything I am saying.

Also….

I talk fast…sometimes absurdly so.



# What are we not going to talk about?

- Functional programming in C#
- What??? I thought you said we were going to talk about functional programming!
- Go check out F# (http://research.microsoft.com/fsharp/)

# What are we going to talk about?

- What is functional programming?
- Features of C# 3.0 that were borrowed from functional programming
- Delegates, Lambdas, and Closures
- Higher Order Functions
- Memoization
- Lazy Evaluation
- Maybe Currying if we have time

# What is functional programming?

“a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data.”

-Paul Hudak

Head of Yale Haskell Group

More simply:

# Variables don’t vary.

http://www.flickr.com/photos/quiddle/153663839/in/set-72157594144890007/

What's that you say?

How do you program without variables?

Very carefully.

~~Very carefully.~~

Just kidding…

Instead of modifying state we compose functions:

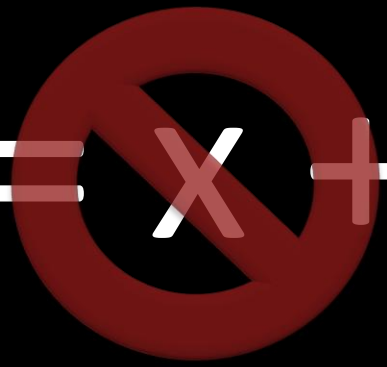
$$f(g(x))$$

And use recursion:

$$f(x) = f(x) + 1$$

And no variables, only constant values:

$$x = x + 1$$



$$y = x + 1$$

# Demo:

Sum integers from 1 to 100
Imperative vs. Functional

And we create Higher Order functions:

```
f(func, g){
  return func
}
```

Functions must be first class objects.

# What support does C# has for first class functions?

- Delegates (C# 1.0)
- Anonymous Methods (C# 2.0)
- Lambdas (C# 3.0)

# Demo:

Delegates, Anonymous Methods, and Lambdas

# Before we go further:

- Lambdas (closer look)
- Implicitly Typed Variables
- Func, Action, and Predicate delegates
- Extension Methods

# Demo:

Lambdas, Implicitly Typed Variables, Func/Action/Predicate delegates, and Extension Methods

# Closure:

A function that is bound to one or more variables in its enclosing scope.

Let’s think about that…

# What is the enclosing scope of a function in C#?

```
public class Scope
{
    private string privateValue = "value";
    public void DoSomething(string a)
    {
        int someValue = 0;
    }
}
```

# What is the enclosing scope of an anonymous function (or lambda) in C#?

```
public class Scope
{
    private string privateValue = "value";
    public void DoSomething(string a)
    {
        int someValue = 0;

        Func<int, int> func = delegate(int value)
                                {
                                    return value + 5;
                                };
    }
}
```

# Demo:

Delegates, Anonymous Methods, and Lambdas **as closures**

So, what can we do with closures and first class functions?

One thing we can do…

# Higher Order Functions

Map, Fold, Filter

# Demo:

Higher Order Functions

Another thing we can do…

# Memoization

# Demo:

Memoization

Any questions before we talk about Lazy Evaluation?

Switching gears…

# Lazy Evaluation
# (aka Delayed Execution)

# Demo:

Lazy Evaluation with Linq

What we looked at:

- Functional Programming
- Lambdas
- Closures
- Higher Order Functions
- Memoization
- Lazy Evaluation

# Questions?

justin@etheredge.us
http://www.codethinked.com