What task is best done in a functional programming style?

Asked 15 years, 8 months ago Modified 9 years, 7 months ago

Viewed 10k times

30

I've just recently discovered the functional programming style and I'm convinced that it will reduce development efforts, make code easier to read, make software more maintainable. However, the problem is I sucked at convincing anyone.

Well, recently I was given a chance to give a talk on how to reduce software development and maintenance efforts, and I wanted to introduce them the concept of functional programming and how it benefit the team. I had this idea of showing people 2 set of code that does exactly the same thing, one coded in a very imperative way, and the other in a very functional way, to show that functional programming can made code way shorter, easier to understand and thus maintainable. Is there such an example, beside the famous sum of squares example by Luca Bolognese?

c# f# c#-3.0 functional-programming

edited Jan 24, 2011 at 18:37

Follow

asked Mar 29, 2009 at 13:52

Hao Wooi Lim

3,988 ● 5 ● 32 ● 35

Would someone be able to provide link to where to find "the famous sum of squares example by Luca Bolognese"? Thanks – dance2die Mar 29, 2009 at 21:20

5 Luca's talk: channel9.msdn.com/pdc2008/TL11 – Brian Mar 31, 2009 at 1:57

16 Answers

Sorted by: Highest score (default)

68

> I've just recently discovered the functional programming style [...] Well, recently I was given a chance to give a talk on how to reduce software development efforts, and I wanted to introduce the concept of functional programming.

If you've only just discovered functional programming, I *do not* recommend trying to speak authoritatively on the subject. I know for the first 6 months while I was learnig F#, all of my code was just C# with a little more awkward syntax. However, after that period of time, I was able to write consistently good code in an idiomatic, functional style.

I recommend that you do the same: wait for 6 months or so until functional programming style comes more naturally, then give your presentation.

> I'm trying to illustrate the benefits of functional programming, and I had the idea of showing people 2 set of code that does the same thing, one coded in a very imperative way, and the other in a very functional way, to show that functional programming can made code way shorter, easier to understand and thus maintain. Is there such example, beside the famous sum of squares example by Luca Bolognese?

I gave an F# presentation to the .NET users group in my area, and many people in my group were impressed by F#'s pattern matching. Specifically, I showed how to traverse an abstract syntax tree in C# and F#:

```
using System;

namespace ConsoleApplication1
{
    public interface IExprVisitor<t>
    {
        t Visit(TrueExpr expr);
        t Visit(And expr);
        t Visit(Nand expr);
        t Visit(Or expr);
        t Visit(Xor expr);
        t Visit(Not expr);

    }

    public abstract class Expr
```

```
    {
        public abstract t Accept<t>(IExprVisitor<t> vi
    }

    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 override t Accept<t>(IExprVisitor<t> vi
        {
            return visitor.Visit(this);
        }
    }

    public class And : BinExpr
    {
        public And(Expr first, Expr second) : base(fir
        public override t Accept<t>(IExprVisitor<t> vi
        {
            return visitor.Visit(this);
        }
    }

    public class Nand : BinExpr
    {
```

```
        public Nand(Expr first, Expr second) : base(fi
        public override t Accept<t>(IExprVisitor<t> vi
        {
            return visitor.Visit(this);
        }
    }

    public class Or : BinExpr
    {
        public Or(Expr first, Expr second) : base(firs
        public override t Accept<t>(IExprVisitor<t> vi
        {
            return visitor.Visit(this);
        }
    }

    public class Xor : BinExpr
    {
        public Xor(Expr first, Expr second) : base(fir
        public override t Accept<t>(IExprVisitor<t> vi
        {
            return visitor.Visit(this);
        }
    }

    public class Not : UnaryOp
    {
        public Not(Expr first) : base(first) { }
        public override t Accept<t>(IExprVisitor<t> vi
        {
            return visitor.Visit(this);
        }
    }

    public class EvalVisitor : IExprVisitor<bool>
    {
        public bool Visit(TrueExpr expr)
        {
            return true;
        }

        public bool Visit(And expr)
        {
            return Eval(expr.First) && Eval(expr.Secon
```

```
        }

        public bool Visit(Nand expr)
        {
            return !(Eval(expr.First) && Eval(expr.Sec
        }

        public bool Visit(Or expr)
        {
            return Eval(expr.First) || Eval(expr.Secon
        }

        public bool Visit(Xor expr)
        {
            return Eval(expr.First) ^ Eval(expr.Second
        }

        public bool Visit(Not expr)
        {
            return !Eval(expr.First);
        }

        public bool Eval(Expr expr)
        {
            return expr.Accept(this);
        }
    }

    public class PrettyPrintVisitor : IExprVisitor<str
    {
        public string Visit(TrueExpr expr)
        {
            return "True";
        }

        public string Visit(And expr)
        {
            return string.Format("({0}) AND ({1})", ex
expr.Second.Accept(this));
        }

        public string Visit(Nand expr)
        {
            return string.Format("({0}) NAND ({1})", e
```

```
expr.Second.Accept(this));
        }

        public string Visit(Or expr)
        {
            return string.Format("({0}) OR ({1})", exp
expr.Second.Accept(this));
        }

        public string Visit(Xor expr)
        {
            return string.Format("({0}) XOR ({1})", ex
expr.Second.Accept(this));
        }

        public string Visit(Not expr)
        {
            return string.Format("Not ({0})", expr.Fir
        }

        public string Pretty(Expr expr)
        {
            return expr.Accept(this).Replace("(True)",
        }
    }

    class Program
    {
        static void TestLogicalEquivalence(Expr first,
        {
            var prettyPrinter = new PrettyPrintVisitor
            var eval = new EvalVisitor();
            var evalFirst = eval.Eval(first);
            var evalSecond = eval.Eval(second);

            Console.WriteLine("Testing expressions:");
            Console.WriteLine("    First  = {0}", pret
            Console.WriteLine("        Eval(First):  {
            Console.WriteLine("    Second = {0}",
prettyPrinter.Pretty(second));
            Console.WriteLine("        Eval(Second): {
            Console.WriteLine("    Equivalent? {0}", e
            Console.WriteLine();
        }
```

```
        static void Main(string[] args)
        {
            var P = new TrueExpr();
            var Q = new Not(new TrueExpr());

            TestLogicalEquivalence(P, Q);

            TestLogicalEquivalence(
                new Not(P),
                new Nand(P, P));

            TestLogicalEquivalence(
                new And(P, Q),
                new Nand(new Nand(P, Q), new Nand(P, Q

            TestLogicalEquivalence(
                new Or(P, Q),
                new Nand(new Nand(P, P), new Nand(Q, Q

            TestLogicalEquivalence(
                new Xor(P, Q),
                new Nand(
                    new Nand(P, new Nand(P, Q)),
                    new Nand(Q, new Nand(P, Q)))
                );

            Console.ReadKey(true);
        }
    }
}
```

The code above is written in an idiomatic C# style. It uses the visitor pattern rather than type-testing to guarantee type safety. This is about 218 LOC.

Here's the F# version:

```
#light
open System
```

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

let (^^) p q = not(p && q) && (p || q) // makeshift xo

let rec eval = function
    | True          -> true
    | And(e1, e2)   -> eval(e1) && eval(e2)
    | Nand(e1, e2)  -> not(eval(e1) && eval(e2))
    | Or(e1, e2)    -> eval(e1) || eval(e2)
    | Xor(e1, e2)   -> eval(e1) ^^ eval(e2)
    | Not(e1)       -> not(eval(e1))

let rec prettyPrint e =
    let rec loop = function
        | True          -> "True"
        | And(e1, e2)   -> sprintf "(%s) AND (%s)" (lo
        | Nand(e1, e2)  -> sprintf "(%s) NAND (%s)" (l
        | Or(e1, e2)    -> sprintf "(%s) OR (%s)" (loo
        | Xor(e1, e2)   -> sprintf "(%s) XOR (%s)" (lo
        | Not(e1)       -> sprintf "NOT (%s)" (loop e1
    (loop e).Replace("(True)", "True")

let testLogicalEquivalence e1 e2 =
    let eval1, eval2 = eval e1, eval e2
    printfn "Testing expressions:"
    printfn "    First  = %s" (prettyPrint e1)
    printfn "        eval(e1): %b" eval1
    printfn "    Second = %s" (prettyPrint e2)
    printfn "        eval(e2): %b" eval2
    printfn "    Equilalent? %b" (eval1 = eval2)
    printfn ""

let p, q = True, Not True
let tests =
    [
        p, q;

        Not(p), Nand(p, p);
```

```
        And(p, q),
            Nand(Nand(p, q), Nand(p, q));

        Or(p, q),
            Nand(Nand(p, p), Nand(q, q));

        Xor(p, q),
            Nand(
                    Nand(p, Nand(p, q)),
                    Nand(q, Nand(p, q))
                )
    ]
tests |> Seq.iter (fun (e1, e2) -> testLogicalEquivale

Console.WriteLine("(press any key)")
Console.ReadKey(true) |> ignore
```

This is 65 LOC. Since it uses pattern matching rather than the visitor pattern, we don't lose any type-safety, and the code is very easy to read.

Any kind of symbolic processing is orders of magnitude easier to write in F# than C#.

[Edit to add:] Oh, and pattern matching isn't just a replacement for the visitor pattern, it also allows you to match against the *shape* of data. For example, here's a function which converts Nand's to their equivalents:

```
let rec simplify = function
    | Nand(p, q) when p = q -> Not(simplify p)
    | Nand(Nand(p1, q1), Nand(p2, q2))
        when equivalent [p1; p2] && equivalent [q1; q2
                    -> And(simplify p1, simplify q1)
    | Nand(Nand(p1, p2), Nand(q1, q2))
        when equivalent [p1; p2] && equivalent [q1; q2
                    -> Or(simplify p1, simplify q1)
```

```
    | Nand(Nand(p1, Nand(p2, q1)), Nand(q2, Nand(p3, q
      when equivalent [p1; p2; p3] && equivalent [q1
                  -> Xor(simplify p1, simplify q1)
    | Nand(p, q) -> Nand(simplify p, simplify q)
    | True       -> True
    | And(p, q)  -> And(simplify p, simplify q)
    | Or(p, q)   -> Or(simplify p, simplify q)
    | Xor(p, q)  -> Xor(simplify p, simplify q)
    | Not(Not p) -> simplify p
    | Not(p)     -> Not(simplify p)
```

Its not possible to write this code concisely at all in C#.

Share Improve this answer Follow

edited Mar 29, 2009 at 15:51

answered Mar 29, 2009 at 15:37

Juliet

81.4k ● 46 ● 199 ● 229

What does `let rec loop = function` mean? I understand that it is declaration of recursive function, but what does the `function` keyword mean here? Is it some simple way to declare a function that takes single parameter with pattern matching on this single parameter? (hence we omit parameter name and `match` keyword)
– Dmitrii Lobanov Aug 19, 2011 at 9:34

1 @DmitryLobanov I means the same as `let rec loop = match loop with` – user522860 Sep 28, 2011 at 15:01

2 @CheckMate9500 you mean `let rec loop x = match x with...` – Mauricio Scheffer Feb 28, 2012 at 22:46

can someone point me to an example of the practical usage of the c# interface IExprVisitor<t> code above?
– Alex Gordon Feb 13, 2017 at 20:49

The Motivation on Wikipedia's Visitor Pattern article describes exporting objects as images (or rendering them on screen) in a CAD system. You don't have any evaluation/processing specific code in the expression nodes and you also get a compile time error if you "forget" an implementation for a newly created `Expr` subtype, because `Expr` knows `IExprVisitor` and `IExprVisitor` knows every possible `Expr` type. If you were to implement that using `switch/case`, it's easy to forget specialising for a new type. – sunside Feb 18, 2017 at 20:42

14

There are plenty examples out there but none will be as impact full as using a sample relevant to one of your projects at work. Examples like "Sum Of Squares" by Luca are awesome but if someone used that as proof as to how our code base could be written better I would not be convinced. All the example proves is some things are better wrote functionally. What you need to prove is your code base is better written functionally

My advice would be to pick some popular trouble spots and some core spots in the code base, and rewrite them in a functional style. If you can demonstrate a substantially better solution, it will go a long way to winning over co-workers.

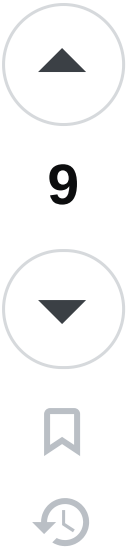
Share Improve this answer

Follow

answered Mar 29, 2009 at 13:57

JaredPar

753k ● 151 ● 1.3k ● 1.5k



Tasks for functional style? Any time you have a common coding pattern and want to reduce it. A while ago I wrote a bit on using C# for functional style, while making sure it's practical: Practical Functional C# (I'm hesitate to link to my own stuff here, but I think it's relevant in this case). If you have a common "enterprise" application, showing, say, how expressions are lovely in pattern matching won't be too convincing.

But in real-world apps, there are TONS of patterns that pop up at a low, coding level. Using higher order functions, you can make them go away. As I show in that set of blog posts, my favourite example is WCF's "try-close/finally-abort" pattern. The "try/finally-dispose" pattern is so common it got turned into a language keyword: using. Same thing for "lock". Those are both trivially represented as higher order functions, and only because C# didn't support them originally do we need hard-coded language keywords to support them. (Quick: switch your "lock" blocks out to use a ReaderWriter lock. Oops, we'll have to write a higher order function first.)

But perhaps convincing just requires looking at Microsoft. Generics aka parametric polymorphism? That's hardly OO, but a nice functional concept that, now, everyone loves. The cute Ninject framework wouldn't work without it. Lambdas? As expression trees, they're how LINQ, Fluent NHibernate, etc. get all their power. Again, that doesn't come from OO or imperative programming. The new Threading library? Pretty ugly without closures.

So, functional programming has been blessing things like .NET over the last decade. The major advances (such as generics, "LINQ") are directly from functional languages. Why not realise there's something to it and get more involved in it? That's how I'd phrase it to skeptics.

The bigger problem is actually getting people to make the jump in understanding to higher order functions. While it's quite easy, if you've never seen it before in your life, it might be shocking an incomprehensible. (Heck, seems like a lot of people think generics are just for type-safe collections, and LINQ is just embedded SQL.)

So, what you should do is go through your codebase, and find places that are an overly-complicated imperative nightmare. Search for the underlying patterns, and use functions to string it together nicely. If you can't find any, you might settle for just demo'ing off lists. For example "find all the Foos in this list and remove them". That's a 1 line thing in functional style "myList.Remove(x=>x.Bla > 0)" versus 7 lines in C# style (create a temp list, loop through and add to-remove items, loop though and remove the items).

The hope is that, even though the syntax is odd, people will recognize "wow, that's a lot simpler". If they can put down the "verbose == more readable" and "that looks confusing" for a bit, you'll have a chance.

Good luck.

Follow

answered Mar 29, 2009 at 21:17

MichaelGG

10k ● 1 ● 41 ● 82

1 web.archive.org/web/20160316073516/http://www.atrevido.net/blog/… – Kristoffer Jälén Aug 9, 2017 at 13:10

3

The best advocacy paper ever written for the functional style is a paper by John Hughes called Why Functional Programming Matters. I suggest you do some examples for yourself until you reach the stage where you can convincingly make the arguments laid out in that paper.

Many of the examples in the paper are numerical and do not resonate with today's audiences. One more contemporary exercise I gave my students was to use the ideas in that paper to pack large media files onto 4.7GB DVDs for backup. They used Michael Mitzenmacher's "bubble search" algorithm to generate alternative packings, and using this algorithm and Hughes's techniques it was easy to get each DVD (except the last) 99.9% full. Very sweet.

Share Improve this answer

edited Jan 21, 2011 at 2:25

Follow

answered Mar 31, 2009 at 1:09

Norman Ramsey

202k ● 62 ● 371 ● 541

1 @compie fixed. I hate linkrot. – Norman Ramsey Jan 21, 2011 at 2:25

2

Essentially, the functional paradigm is highly effective for parallel processing:

> "The really interesting thing I want you to notice, here, is that as soon as you think of map and reduce as functions that everybody can use, and they use them, you only have to get one supergenius to write the hard code to run map and reduce on a global massively parallel array of computers, and all the old code that used to work fine when you just ran a loop still works only it's a zillion times faster which means it can be used to tackle huge problems in an instant.
>
> Lemme repeat that. By abstracting away the very concept of looping, you can implement looping any way you want, including implementing it in a way that scales nicely with extra hardware."
>
> http://www.joelonsoftware.com/items/2006/08/01.html

Share Improve this answer

Follow

answered Mar 30, 2009 at 7:10

ken

3,699 ● 1 ● 31 ● 44

1

Another example would be the Quicksort algorithm. It can be described very briefly in a functional language like Haskell:

```
qsort []     = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++ qsort
```

but needs some more coding in an iterative language. On the referenced Website you can also find a lot of other examples with comparisons between languages.

Share Improve this answer

Follow

answered Mar 29, 2009 at 14:13

crackmigg

3 On that same link you can also find that this is kind of inefficient, since you're traversing the xs list twice on each iteration. So, although it is a very popular example, I don't particularly like it. – Kurt Schelfthout Mar 29, 2009 at 15:15

2 Also, this doesn't really make code cleaner, this piece of code seems rather "clever" (if you know what I mean), and, aren't we doing alot of copying here? not very efficient I'd say. – hasen Mar 29, 2009 at 15:45

1 @tokland: Quicksort was specifically designed to reduce memory overhead by swapping in-place. This bastardized derivative incurs massive amounts of completely unnecessary copying and, consequently, is orders of magnitude slower than a real quicksort. – J D Feb 13, 2011 at 20:55

1 Historically, this is one of skeletons in the functional programming closet. The other one is the Sieve of Eratosthenes, where prominent figures in the FP community

taught tens of thousands of students about a beautiful purely functional implementation before Melissa O'Neill wrote a paper explaining why they had been wrong for so many years. cs.hmc.edu/~oneill/papers/Sieve-JFP.pdf – J D Feb 13, 2011 at 20:59

1 @Jon: I am no expert on algorithmia, but I am surprised because I've seen this kind of implementation of quicksort in many (and serious) places. The wikipedia states: "Although quicksort is usually not implemented as an in-place sort, it is possible to create such an implementation". Can you provide any URL? [checking the sieve one, any about quicksort?] – tokland Feb 13, 2011 at 21:03

1

In order to achieve what you want, and to communicate it to others in your organisation you need to demonstrate your company's business being built in a better way.

Its no use using a couple of algorithms to demonstrate the power of functional programming if its totally useless for your business domain. So, take some existing code and rewrite it functionally. If you can prove through that, that it is better, people will listen to you - you've shown them a concrete, relevant example. IF you cannot, then perhaps functional programming is not the solution you've been looking for.

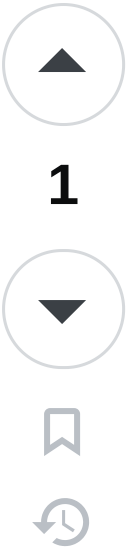
Share Improve this answer

Follow

answered Mar 29, 2009 at 14:20

gbjbaanb

52.6k ● 12 ● 110 ● 154



If by the 'functional style' you mean the usage of concepts like 'map', 'apply', 'reduce', 'filter', lambda functions and list comprehensions, then it should be evident that code that has to deal with operations on lists is almost always more concise when written in the 'functional style'. But if you're mixing 'functional style' with imperative code in a mostly imperative language, it really is just a matter of style.

In python for example, you can reimplement the Haskell qsort crackmigg posted like so:

```
def qsort(list):
    if list == []:
        return []
    else:
        x = list[0]; xs = list[1:]
        return qsort(filter(lambda y: y<x, xs)) + [x]
y>= x, xs))
```

Although writing the last line as

```
return qsort([y for y in xs if y<x]) + [x] + qsort([y
```

is arguably more 'pythonic'.

But this is obviously more concise than the implementation here:

http://hetland.org/coding/python/quicksort.html

Which, incidentally, is how I would have thought about implementing it before I learned Haskell.

The functional version is extremely clear *if* and *only if* you are acclimated to functional programming and grok what `filter` is going to do as easily as the well-worn C++ programmer groks a `for` loop without even having to think much about it. And that's clearly what the real issue at stake here: functional 'style' programming is a *completely different mindset*. If the people you work with aren't used to thinking recursively, and aren't the type to get excited about, not just a new technology, but a whole other way of thinking about solving their problems, then any amount of code comparisons isn't going to win them over.

Share Improve this answer

Follow

answered Mar 29, 2009 at 14:47

Zachary Hamm

628 ● 3 ● 16

2 What you posted is not technically an implementation of the quicksort algorithm, because it will go through the list twice to partition it. See augustss.blogspot.com/2007/08/… – RossFabricant Mar 29, 2009 at 15:09

You're correct. The lack of O(n) array operations is the biggest drawback to functional programming (although you can do them in Lisp with vectors!) – Zachary Hamm Mar 29, 2009 at 15:20

Splitting a list in two can easily be done in a single pass. I think that F# has such a function built in. Two passes would also be O(n) though. You can also have something equivalent to array operations. Check out Chris Okasaki's "Purely

Functional Data Structures." – Jørgen Fogh Jul 20, 2009 at 13:24

That is not quicksort but not because of the double pass. – J D Jan 23, 2011 at 18:50

0

A good example cood be creating your own programming language using existing one, where you will have to use Monads.

With F# it's much much simplier to write parsing logic than with C#.

Take a look at this article: Functional .NET - LINQ or Language Integrated Monads?

Share Improve this answer Follow

edited Mar 29, 2009 at 15:02

answered Mar 29, 2009 at 14:54

Konstantin Tarkus

38.3k ● 14 ● 136 ● 121

0

Algorithms involving backtracking search and simplifying undo support in GUIs are two places I've used functional style in practice.

Share Improve this answer

Follow

answered Mar 29, 2009 at 15:08

Pete Kirkham

49.3k ● 5 ● 94 ● 173

0

A simple example of a task that is often easiest done in a functional style is the transformation of data from one form to another. "Sum of squares" is a trivial example of data transformation. Luca's talk at last year's PDC showed how to use this sort of data transformation for something more practical, downloading and analyzing stock quotes. The demo is done in F#, but the concepts are the same and can be applied to C# or most other programming languages.

http://channel9.msdn.com/pdc2008/TL11/

Share Improve this answer

Follow

answered Mar 29, 2009 at 15:26

Dustin Campbell

9,855 ● 2 ● 32 ● 33

0

Show them jQuery's way of iterating over DOM elements:

```
$(".magic-divs").click(function(){
    // FYI, in this context, "this" will be the elemen
    alert("somebody clicked on " + this.id);
    this.hide();

});

$(".magic-divs").show();
```

vs. how most google results for "javascript element by classname" do it:

```
var arrayOfElements = // this is filled with the eleme
for(var i=0,j=arrayOfElements.length; i<j; i++) {
   alert("now I get to add an onclick event somehow "
}
// i dont even want to type the ugly for-loop stuff to
```

Functional programming is useful in everyday stuff like the above!

(note: I don't know if my example fits the exact definition of functional programming, but if it does, than functional programming is awesome)

Share Improve this answer Follow

edited Mar 29, 2009 at 21:25

answered Mar 29, 2009 at 15:37

Cory R. King

2,796 ● 1 ● 24 ● 22

0

I came up with a little trick recently to make lambdas, passed into my extension methods look more F# ish. Here it goes:

What I wanted to do was something like:

```
3.Times(() => Console.WriteLine("Doin' it"));
```

Now the extension method for that is easily implemented:

```
    public static void Times(this int times, Action ac
    {
        Enumerable.Range(1, times).ToList().ForEach(in
    }
```

What I didn't like is that I am specifying the index here: `ForEach(index => action())` although it never is used, so I replaced it with a `_` and got `ForEach(_ => action())`

That's nice, but now I was motivated to have my calling code look similar

(I never liked the "()" at the beginning of the lambda expression), so instead of: `3.Times(() => ...);` I wanted `3.Times(_ => ...);` The only way to implement this was to pass a fake parameter to the extension method, which never gets used and so I modified it like so:

```
    public static void Times(this int times, Action<by
    {
        Enumerable.Range(1, times).ToList().ForEach(_
action(byte.MinValue));
    }
```

This allows me to call it like this:

```
3.Times(_ => Console.WriteLine("Doin' it"));
```

Not much of a change, but I still enjoyed, that it was possible to make that little tweak so easily and at the same time removing the "()" noise makes it much more readable.

Share Improve this answer Follow

edited Dec 21, 2009 at 21:55

answered Dec 21, 2009 at 21:41

Thorsten Lorenz
11.8k ● 8 ● 53 ● 62

3 I'd argue that this doesn't make it more readable in C#. Knowing C#, I'd look at this code and go 'WTF?'.
– Robert Jeppesen Oct 23, 2010 at 21:48

-1

1. Show how to code a distinct of an array. Distinct is very easy in SQL but was hard on a memory array. Now it is easy to distinct an array with LINQ.
2. You can explain them that there will be parralel LINQ (PLINQ) in the future. When you start writing functional code it will be easier to parralelize your application. Google uses MapReduce extensively.
3. Explain to them that LINQ is a query language to manipulate al different kinds of data. In memory, in a database, excell, web services, xml files, JSON files. It is some kind of universal SQL. However people who don't like SQL will be less convinced.

I wouldn't talk to much about functional programming, I would explain how LINQ can help developers.

Share Improve this answer

Follow

answered Mar 29, 2009 at 15:13

tuinstoel

7,306 ● 29 ● 28

-1

Not really answering the question, but this is a very good link for those who want to know about functional programming in C#

http://blogs.msdn.com/b/ericwhite/archive/2006/10/04/fp-tutorial.aspx

Share Improve this answer

Follow

answered May 25, 2015 at 22:25

community wiki

Rashmi Pandit

Link is dead (403). – Marc L. Mar 31, 2023 at 19:49

-3

It's interesting noone has really answered the question: what task is best done in a "functional style"?

A program/algorithm consists of 2 parts: logic control and data structure. I think the tasks best done in a functional style are those involved lists or arrays in cases where they behave like list (e.g. qsort). It's no coincidence that

functional programming languages have very good support for lists.

When the data structures deviate from lists, I think the use of a functional programming style become a little "unnatural".

Share Improve this answer Follow

answered Mar 29, 2009 at 16:54

vph