Ten reasons not to use a statically typed functional programming language

A rant against something I don't get

Are you fed up with all the hype about functional programming? Me too! I thought I'd rant about some reasons why sensible people like us should stay away from it.

Just to be clear, when I say "statically typed functional programming language", I mean languages that also include things such as type inference, immutability by default, and so on. In practice, this means Haskell and the ML-family (including OCaml and F#).

Reason 1: I don't want to follow the latest fad

Like most programmers, I'm naturally conservative and I dislike learning new things. That's why I picked a career in IT.

I don't jump on the latest bandwagon just because all the "cool kids" are doing it -- I wait until things have matured and I can get some perspective.

To me, functional programming just hasn't been around long enough to convince me that it is here to stay.

Yes, I suppose some pedants will claim that ML and Haskell have been around almost as long as old favorites like Java and PHP, but I only heard of Haskell recently, so that argument doesn't wash with me.

And look at the baby of the bunch, F#. It's only seven years old, for Pete's sake! Sure, that may be a long time to a geologist, but in internet time, seven years is just the blink of an eye.

So, all told, I would definitely take the cautious approach and wait a few decades to see if this functional programming thing sticks around or whether it is just a flash in the pan.

Reason 2: I get paid by the line

I don't know about you, but the more lines of code I write, the more productive I feel. If I can churn out 500 lines of code in a day, that's a job well done. My commits are big, and my boss can see that I've been busy.

But when I compare code written in a functional language with a good old C-like language, there's so much less code that it scares me.

I mean, just look at this code written in a familiar language:

```
public static class SumOfSquaresHelper
{
   public static int Square(int i)
```

```
    {
       return i * i;
    }

    public static int SumOfSquares(int n)
    {
       int sum = 0;
       for (int i = 1; i <= n; i++)
       {
          sum += Square(i);
       }
       return sum;
    }
}
```

and compare it with this:

```
let square x = x * x
let sumOfSquares n = [1..n] |> List.map square |> List.sum
```

That's 17 lines vs. only 2 lines. Imagine that difference multiplied over a whole project!

If I did use this approach, my productivity would drop drastically. I'm sorry -- I just can't afford it.

Reason 3: I love me some curly braces

And that's another thing. What's up with all these languages that get rid of curly braces. How can they call themselves real programming languages?

I'll show you what I mean. Here's a code sample with familiar curly braces.

```
public class Squarer
{
    public int Square(int input)
    {
        var result = input * input;
        return result;
    }

    public void PrintSquare(int input)
    {
        var result = this.Square(input);
        Console.WriteLine("Input={0}. Result={1}", input, result);
    }
}
```

And here's some similar code, but without curly braces.

```
type Squarer() =

    let Square input =
        let result = input * input
        result
    let PrintSquare input =
        let result = Square input
        printf "Input=%i. Result=%i" input result
```

Look at the difference! I don't know about you, but I find the second example a bit disturbing, as if something important is missing.

To be honest, I feel a bit lost without the guidance that curly braces give me.

Reason 4: I like to see explicit types

Proponents of functional languages claim that type inference makes the code cleaner because you don't have to clutter your code with type declarations all the time.

Well, as it happens, I *like* to see type declarations. I feel uncomfortable if I don't know the exact type of every parameter. That's why Java is my favorite language.

Here's a function signature for some ML-ish code. There are no type declarations needed and all types are inferred automatically.

```
let GroupBy source keySelector =
    ...
```

And here's the function signature for similar code in C#, with explicit type declarations.

```
public IEnumerable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>(
    IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector
    )
    ...
```

I may be in the minority here, but I like the second version much better. It's important to me to know that the return is of type `IEnumerable<IGrouping<TKey, TSource>>`.

Sure, the compiler will type check this for you and warn you if there is a type mismatch. But why let the compiler do the work when your brain can do it instead?

Ok, I admit that if you do use generics, and lambdas, and functions that return functions, and all the other newfangled stuff, then yes, your type declarations can get really hairy and complex. And it gets really hard to type them properly.

But I have an easy fix for that -- don't use generics and don't pass around functions. Your signatures will be much simpler.

Reason 5: I like to fix bugs

To me, there's nothing quite like the thrill of the hunt -- finding and killing a nasty bug. And if the bug is in a production system, even better, because I'll be a hero as well.

But I've read that in statically typed functional languages, it is much harder to introduce bugs.

That's a bummer.

Reason 6: I live in the debugger

And talking of bug fixing, I spend most of my day in the debugger, stepping through code. Yes, I know I should be using unit tests, but easier said than done, OK?

Anyway, apparently with these statically typed functional languages, if your code compiles, it usually works.

I'm told that you do have to spend a lot of time up front getting the types to match up, but once that is done and it compiles successfully, there is nothing to debug. Where's the fun in that?

Which brings me to...

Reason 7: I don't want to think about every little detail

All this matching up types and making sure everything is perfect sounds tiring to me.

In fact, I hear that you are forced to think about all the possible edge cases, and all the possible error conditions, and every other thing that could go wrong. And you have to do this at the beginning -- you can't be lazy and postpone it till later.

I'd much rather get everything (mostly) working for the happy path, and then fix bugs as they come up.

Reason 8: I like to check for nulls

I'm very conscientious about checking for nulls on every method. It gives me great satisfaction to know that my code is completely bulletproof as a result.

```
void someMethod(SomeClass x)
{
    if (x == null) { throw new NullArgumentException(); }

    x.doSomething();
}
```

Haha! Just kidding! Of course I can't be bothered to put null-checking code everywhere. I'd never get any real work done.

But I've only ever had to deal with one bad crash caused by a NPE. And the business didn't lose too much money during the few weeks I spent looking for the problem. So I'm not sure why this is such a big deal.

Reason 9: I like to use design patterns everywhere

I first read about design patterns in the Design Patterns book (for some reason it's referred to as the Gang of Four book, but I'm not sure why), and since then I have been diligent in using them at all times for all problems. It certainly makes my code look serious and "enterprise-y", and it impresses my boss.

But I don't see any mention of patterns in functional design. How can you get useful stuff done without Strategy, AbstractFactory, Decorator, Proxy, and so on?

Perhaps the functional programmers are not aware of them?

Reason 10: It's too mathematical

Here's some more code for calculating the sum of squares. This is *way* too hard to understand because of all the weird symbols in it.

```
ss=: +/ @: *:
```

Oops, sorry! My mistake. That was J code.

But I do hear that functional programs use strange symbols like `<*>` and `>>=` and obscure concepts called "monads" and "functors".

I don't know why the functional people couldn't stick with things I already know -- obvious symbols like `++` and `!=` and easy concepts such as "inheritance" and "polymorphism".

Summary: I don't get it

You know what. I don't get it. I don't get why functional programming is useful.

What I'd really like is for someone to just show me some real benefits on a single page, instead of giving me too much information.

UPDATE: So now I've read the "everything you need to know on one page" page. But it's too short and simplistic for me.

I'm really looking for something with a bit more depth -- something I can get my teeth into.

And no, don't say that I should read tutorials, and play with examples, and write my own code. I just want to grok it without doing all of that work.

I don't want to have to change the way I think just to learn a new paradigm.

Posted by scottw on 12 Apr 2013

Comments

65 Comments F# for fun and profit 1 Login

Recommend 5 Share Sort by Best

Join the discussion…

Simon Morgan • 2 years ago

It's somewhat depressing how many people take this post seriously.

96 • Reply • Share ›

SoulFireMage • 2 years ago

Oh the irony, Fantastic. What a great piece, I have to pass it on :)

30 ^ | ˅ • Reply • Share ›

scottw Mod • 2 years ago

For those with an irony-deficiency, this page is not meant to be taken seriously.

For those who think I am being cruel or trolling, everything here is based on what I have actually read or what people have told me in person (with only a little exaggeration). I have just made *explicit* what is implicit in their reactions.

Also, to be clear, I am not trying to attack (or be snobbish about) those who prefer not to use FP. People can use whatever tools they like. There are plenty of good reasons *not* to use FP, but the ones listed here are not among them. I encounter these kinds of defensive and irrational "arguments" all the time, and they don't deserve to be taken seriously.

Some background on the individual points:

Reason 1: "fad". I cannot tell you how many times I have heard this. Please learn you some CS history. "I'm naturally conservative and I dislike learning new things. That's why I picked a career in IT." -- I thought I was being funny. Oh well.

Reason 2: "paid by the line". No one has said this directly, but if you look at, say, any

see more

17 ^ | ˅ • Reply • Share ›

Maciej Piechotka ➦ scottw • 2 years ago

AD Reason 4: I'm not sure about ML but in Haskell you can explicitly type everything (with 'generics' you need to use ScopedTypeVariables possibly but you can). Most people don't but it is possible.

1 ^ | ˅ • Reply • Share ›

IM ➦ Maciej Piechotka • 2 years ago

It's convention in Haskell to provide types for all top-level values. I find it makes code significantly more readable and miss it in MLs.

2 ^ | ˅ • Reply • Share ›

Stephen Ramsay • 2 years ago

I agree with every single thing on this page -- except maybe the last one, because I for damn sure "get it." And I think he forgot " . . . I completely hate clever code, and think, in the end, that it kills productivity" -- which perhaps explains why people mostly write intricate, half-baked parsers in these languages (a fact noted derisively by Knuth). Despite being mature languages, their libraries are nowhere *near* the breadth and scope of almost any other serious mainstream language. The fact that the communities are full of cocky, better-than-though, insular practitioners (amply demonstrated above) might be a factor as well.

These languages are not the promised land of software development. They have flaws and disadvantages like everything else. And the mania for functional programming reminds me a lot of the mania for OO (and structured languages before that) -- "revolutions" in programming that now seem as much a series of compromises as anything else.

This is my tenth time through the latest thing. And like most seasoned developers, I can sense the con from ten miles away. There are great things about functional languages, and bad things. Like any other language, there are ten reasons not to use it -- and ten more reasons why they make sense for a particular problem.

13 ^ | ˅ • Reply • Share ›

Nicolas Alejandro Spirdal-Jaco • 2 years ago

Haha, but he could've stuck to 26 year old Perl, best from all worlds ;)

```
sub square { $_[0] * $_[0] }
sub sum_of_squares { List::Util::sum( map { square($_) } (1..shift); }
```

3 ^ | ˅ • Reply • Share ›

Jhg • 2 years ago

Are you kidding! I hope so.

3 ^ | ˅ • Reply • Share ›

scottw Mod ➦ Jhg • 2 years ago

Alas, no. This is based on some true stories. The names have been changed to protect the guilty.

4 ^ | ˅ • Reply • Share ›

Frustrated old programmer • 2 years ago

I never understood why all of the pretty little teaching languages (Pascal, C, Fortran, etc) require variable typing. If the compiler (or even interpreter) can't figure out that "Helllo World" isn't a number, shouldn't the language developer put down the Jolt and start making improvements.

5 ^ | ˅ • Reply • Share ›

dylan ➦ Frustrated old programmer • a year ago

Calling C a "pretty little teaching language" is so far off the mark I don't know where to start.

2 ^ | ˅ • Reply • Share ›

bypasser ➦ dylan • 10 months ago

Don't start! Regardless if the Frustrated old programmer is old enough to call C a "pretty little teaching language" it doesn't change the meat of the post: C doesn't have what is native to languages like F#.

^ | ˅ • Reply • Share ›

scottw Mod ↪ Frustrated old programmer • 2 years ago

Yes, I agree that type inference is a key part of providing a good experience for beginners. See also
http://blogs.endjin.com/2013/0...

1 ^ | ˅ • Reply • Share ›

Jay ↪ Frustrated old programmer • 3 months ago

I never understood why all the script kiddies use "Hello World" as an example for why their toy languages which are hardly used in serious programs (Haskell, ML, F#, Scala, Python, Ruby etc) are so much better than (Pascal, C, Fortran, C++, Java, etc).

"Hello World programs" are used to introduce complete newbies to programming i.e. those who have never even seen a line code before in their lives (ironic you're calling Pascal, C, Fortran pretty little teaching languages yet they make it harder for the learner). I always laugh when some noob programmer is over excited about how short and sweet it is to write a "Hello World" program in some toy language, and how horrible and verbose it is in C or Java, really man the latter two are for real programmers who have real work to do not play around with "Hello World" scripts all day.

Anyway enough with the insults (used them to make a point why your comment is ridiculous, don't take them too seriously LOL), "Hello World" is not a good example, lets see how type inference handles more complicated situations.

^ | ˅ • Reply • Share ›

Paul • 12 days ago

You forgot "real programmers take out their own garbage". Its fading in these days of Java, but I still hear claims that GC is a crutch for bad programmers who can't keep track of the memory they are using.

2 ^ | ˅ • Reply • Share ›

Mowat • 2 years ago

@scottw

You lured me into investigating F# with this aggressive and funny post. A truly awesome language!

Thanks.

2 ^ | ˅ • Reply • Share ›

scottw Mod ↪ Mowat • 2 years ago

Thanks. I hope you find F# fun as well as useful. I really enjoy using it.

^ | ˅ • Reply • Share ›

Vasily Kirichenko • 2 years ago

Brilliant! Thanks for your excellent work on this resource.
2 • Reply • Share ›

capcom1116 • a year ago
It took me an embarrassing long amount of time to figure out you were being sarcastic. Most of that was spent wondering why someone writing on a site called "F# for fun and profit" would be blasting statically typed functional languages.
1 • Reply • Share ›

fjrg76 • 2 years ago
I love such a wonderful and brilliant sarcasm !!! I just have started learning FP upon Haskell and I couldn't stopped laughing for a while while reading the post. I'll buy the Scott's book when it's released.
1 • Reply • Share ›

NN • 2 years ago
Try Nemerle language: http://nemerle.org
It both functional and imperative, it supports both syntax with braces and without.
It has local type inference but obliges you to write types in global contract.
And it is much easier to read and understand than F#
1 • Reply • Share ›

Guest • 2 years ago
What role does Automatic Generalization play in this argument?
1 • Reply • Share ›

Mikael Guggenheim • 2 years ago
Awesome, just pure awesome...
1 • Reply • Share ›

Guest • 2 years ago
oh god, what an idiotic post.

> Summary: I don't get it

you have proven your point.
4 • Reply • Share ›

ccdan • 2 years ago
The most ironic thing about this article is that the in-your-face irony is invalid. Fewer lines of code doesn't imply clearer code or improved productivity and quite often it's the very opposite of clarity and efficiency. On the other hand functional languages tend to bring the most awful parts of math into programming: ambiguous, cryptic, incoherent notations and unnecessary, tiresome and highly inefficient (in practice) theoretical concepts that don't translate well into computer instructions (which is why functional languages have awful performance on average)

The often claimed improvement in productivity is a ridiculous myth. The funny truth comes to surface whenever you ask a an FP advocate to implement a real-wold useful application. Or to add functionality to an existing application written in a functional language by another programmer. They start moaning about a multitude of things and finally conclude that FP is probably not very well suited for that particular task, which is pretty much any useful application. :D

3 ^ | v • Reply • Share ›

erik ➦ ccdan • 2 years ago

I respectfully disagree. I've used F# to write 2 real world commercial apps, WcfStorm.Rest and WcfStorm.Server (http://www.wcfstorm.com) and in my experience the productivity improvement is not a myth. With regards to the performance it has the same perf profile as C#.

9 ^ | v • Reply • Share ›

david ➦ ccdan • 2 years ago

I've seen your heavily downvoted answer on Stackoverflow here http://stackoverflow.com/quest.... I'm going to reasonably assume that you must have failed math in college. Please accept my commiserations.

3 ^ | v • Reply • Share ›

Ray ➦ david • 2 years ago

I think that answer will get more and more *votes after you post it here.

^ | v • Reply • Share ›

geraldfnord ➦ ccdan • 24 days ago

I work to make code as explicit and comprehensible for the next person who'll have to fix and to maintain it, which very well might be me(t_0 + 6mos); terseness can be a bad thing for that.

^ | v • Reply • Share ›

nybble41 ➦ geraldfnord • 11 days ago

Languages like Haskell are great for comprehension. Less space taken up in satisfying an overly-simplistic compiler means more space to explain your code to human readers.

1 ^ | v • Reply • Share ›

Mikael Grön • 2 years ago

As a programmer, I have extreme problems relating to these two sentences; "Like most programmers, I'm naturally conservative and I dislike learning new things. That's why I picked a career in IT."... How, in the name of code, do you survive as a developer without loving to learn new languages? That's totally beyond my comprehension. You must be programming traffic lights or something..

2 ^ | v • Reply • Share ›

bypasser ➦ Mikael Grön • 10 months ago

It puzzled me for a moment too, but when we start talking about geology ("It's only seven years old, for Pete's sake! Sure, that may be a long time to a geologist, but in internet time, seven years is just the blink of an eye") I got the sarcasm in this post. Come on, ppl, how hard is it to get the joke?! :)

2 ^ | v • Reply • Share ›

Matej Odaloš • 2 years ago

I too don't really see enterprise solutions build on functional languages (I do not say i can't happened, just that it would surprise me). But these languages have some beauty in their concept and they should be learned by any programmer who take it seriously (as a new perspective to solve problems).

This article is bunch of bullshits or a sarcastic attempt from the author. Either way it incriminates there language with wrongful statements and thus preventing new people to study this subject (as I expect to be quite easy to google it).

2 ^ | v • Reply • Share ›

Why you care? • 2 years ago

Moron

3 ^ | v • Reply • Share ›

Programmer • 2 years ago

This is a stupid post. Are you trying to look like a fool? Was it intentionally dumb?

2 ^ | v • Reply • Share ›

Guest ➦ Programmer • 2 years ago

It's a satire... you really don't get that?

23 ^ | v • Reply • Share ›

Jay • 3 months ago

This is why I love functional programming:

Reason 1: I am hippie who jumps on every fad train that comes along, I jumped on the OOP fad train, made sure I use Multiple inheritance, polymorphism and deep-nested class hierarchies everywhere, but OOP is not cool and sexy anymore, yes I know functional programming is actually VERY old but it is the current hype, so who cares most people think its the hip new thing anyway.

Reason 2: I love to bash OOP, like I bashed procedural programming in the past. I love to make fallacious statements that OOP and non-functional paradigms leads to more lines of code, of course I always make sure to use "Hello World" or some factorial thingy as an example and never something actually complicated such as an Interactive Graphical User Interface because I am scared the OOP example may actually look cleaner, more elegant and more readable whereas my FP example may end up looking like line noise, or even worse I may end up doing OOP disguised as Functional programming by tossing in fancy words such as Monads, or what not else.

Reason 3: My favorite justification for why functional programming is sexy is syntax

Reason 3: My favorite justification for why functional programming is sexy is syntax, because obviously the paradigm dictates the syntax of a language I mean how can

see more

1 • Reply • Share ›

WraithGlade → Jay • 3 months ago

Nice counterargument overall. It's good to see not everyone is drinking the koolaid so blindly. :-)

Functional programming languages have a number of useful features that are less common in "imperative" programming languages (e.g. C, C++, Java, Python, etc). However, functional programmers are often too quick to assume that many of these useful features are somehow genuinely unique to functional languages. If you create a general purpose query data structure of some kind then you still have most of the expressive capabilities for manipulating data arbitrarily and concisely that you'd use in real world practical applications (e.g. SQL, MongoDB, and custom data structures for the applications).

Some of the fundamental premises of functional programming purists seem fundamentally flawed. Foremost among the invalid assumptions is that removing all "mutable state" is somehow ideal. They like to conveniently sweep under the rug the fact that it is literally physically impossible to do things like file input/output, interacting with hardware, graphics rendering, etc without having mutable state. Computers are fundamentally composed of mutable

see more

• Reply • Share ›

Jay → WraithGlade • 3 months ago

"Even in functional programming there's still mutable state, it is merely hidden in a different form. "

You can add "tail recursion" to your example of hidden mutable state. In my opinion structuring loops as tail recursion, which will be optimized as a loop anyway, just for the sake of being functional and pretending there is no side effects, is the most ridiculous idea that is pushed by FP proponents. If a certain problem is more clearly expressed and more readable as recursion then it should be written as recursion however if a problem is fundamentally a loop (e.g. an event loop) then there is no point in writing it as a tail-recursive function with 20+ arguments representing the local variables, especially if only 1 or 2 change out of all them on each iteration. Anyway mutable isolated local variables aren't the problem, the problem is unrestrained use of mutable global variables and mutable state shared across multiple threads but that is discouraged in nearly every modern language.

Another feature which I still cannot understand why anyone would see more

Reply · Share ›

WraithGlade WraithGlade · 3 months ago

Oh, and by the way, even C++ now has support for some functional programming stuff. The mainstream imperative languages are very pragmatic like that: they take anything that's useful and try to balance theory with business constraints and social momentum.

Here's a link:

http://blog.madhukaraphatak.co...

On a different note: In my opinion no functional language (whether pure or mixed) will manage to dominate unless it treats the concept of mutable state as equally important as immutable state, rather than as a second class "dirty" citizen. The idea of "immutable purity" is essentially based on a logical fallacy.

The next big language should also not allow inheritance at all (e.g. disallowing inheritance like Google's Go programming language or Rust do). There's nothing that inheritance can do that things like interfaces, duck typing, component based architectures (e.g. see

see more

Reply · Share ›

Jay WraithGlade · 3 months ago

"Also, simplicity of expression is important and too many functional languages
ignore human constraints like having clear descriptive names"

Right on the mark, I think it stems from the "verbosity" arguments and lot of people seem to think that semicolons, braces or names spelt out in full e.g. "length" instead of awkward abbreviations such as "len" make writing code a pain. In my opinion removing semicolons and abbreviating are a micro optimization when it comes to typing, if typing is that much of problem there is always copy-and-paste and autocomplete. The FP proponents fall in this group (just look at reasons 3 & 4) and hence name there functions with highly abbreviated names. Lisp is not the worst at least it calls its reduction function reduce which makes sense, reduce a sequence to a single value, whereas Haskell and the like call it foldl, foldr not only does fold make less sense but foldl, really? Would it hurt to just call it fold_left (or just fold for the left

see more

^ | ⌄ • Reply • Share ›

Paul Vixie • 10 days ago

<< This is my tenth time through the latest thing. And like most seasoned developers, I can sense the con from ten miles away. >>

yes. this.

letting a compiler silently adjust to the type of an expression is something i never want. i'd turn off integer promotions in C if i could. having written way more than my share of bugs, anything DWIM-ish gets a thumbs down from me because the area under the curve in this case is "total cost of operation" and the X axis is the number of developers who will touch the code, and the Y axis is the code's lifetime.

(DWIM = Do What I Mean)

i think that at its personal best, programming is art, and creating and learning new programming languages are forms of creating and appreciating art. and i must admit, F# source code is very pretty. (disclaimer: i also thought PDP-11 assembly language was very pretty, so you can keep on ignoring me.)

^ | ⌄ • Reply • Share ›

Zax • 11 days ago

I was reading the first couple of lines and was thinking to close the tab and when I skimmed over in the article I quickly realized what's up. I continued reading happily. Thanks :)

^ | ⌄ • Reply • Share ›

scottw Mod ↪ Zax • 11 days ago

Thanks for actually reading it! I thought that the first sentence ("Like most programmers, I'm naturally conservative and I dislike learning new things. That's why I picked a career in IT.") would be a clear giveaway that this is a parody, but apparently not!

2 ^ | ⌄ • Reply • Share ›

Zax ↪ scottw • 11 days ago

To tell you the truth, experience has showed that some developers are actually like that. They find their niche where they create something really essential and after 5 years they are irreplaceable. They never actually move on and the company doesn't make them. It's a stalemate.

^ | ⌄ • Reply • Share ›

scottw Mod ↪ Zax • 11 days ago

Exactly! This is explained well in "The Expert Beginner" - http://www.daedtech.com/how-de... and http://www.daedtech.com/how-so...

^ | ⌄ • Reply • Share ›

Matthias Felleisen • 13 days ago

If this were the description of a regular off-the-street programmer, fine. Sadly it also describes Ivy League computer science departments. Been there, presented the idea, got the above responses.

• Reply • Share ›

Michael C. Gates • 13 days ago

I just like to get the job done, and make money. Then I sit by the fire, and drink a nice cold IPA, or chocolate stout... Maybe even a Java Stout if I'm feeling daring.

• Reply • Share ›

SHARE THE LOVE

F# |> I ❤

SLIDES/VIDEOS

Functional Design Patterns
A functional approach to Domain Driven Design
A functional approach to error handling (Railway Oriented Programming)
Property-based testing
Enterprise Tic-Tac-Toe

SERIES

Why use F#?
Thinking functionally
Expressions and syntax
Understanding F# types
Object-oriented programming in F#
Porting from C#
Designing with types
Computation Expressions
A recipe for a functional app
Dependency cycles
Understanding monoids
Low-risk ways to use F# at work
Annotated walkthroughs
A functional approach to authorization
Handling State
Map and Bind and Apply, Oh my!
Recursive types and folds

RECENT POSTS

Trees in the real world
Generic recursive types
Understanding Folds
Introducing Folds

Catamorphism examples
Introduction to recursive types
» Archives

CATEGORIES

F# vs C# • Conciseness • Types • Folds • Functions • Patterns • Convenience • Partial Application • Correctness • Immutability • Worked Examples • Concurrency • Completeness • Currying • Combinators • Modules • Object-oriented • Classes • Interfaces • DDD • Design • Math • TDD •

MORE

RSS Posts
About
Installing and using F#
Learning F#
Troubleshooting F#
Newsletter

Back to top