Functional programming vs Object Oriented programming [closed]

Asked 14 years, 11 months ago Modified 7 years, 4 months ago

Viewed 311k times

855

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 7 years ago.

Improve this question

I've been mainly exposed to OO programming so far and am looking forward to learning a functional language. My questions are:

- When do you choose functional programming over object-oriented?
- What are the typical problem definitions where functional programming is a better choice?

oop functional-programming paradigms

Share

Improve this question

Follow

edited Aug 22, 2017 at 12:23

community wiki
5 revs, 4 users 59%
Olivier Lalonde

Duplicate: stackoverflow.com/questions/552336/… – gnovice Jan 16, 2010 at 22:23

this is relevant programmers.stackexchange.com/questions/9730/… – kirill_igum Sep 11, 2013 at 2:14

1 similar question cs.se also closed what is example where functional programming gives better results than imperative style. conventional wisdom seems to be that one is not superior to the other, or they are not comparable on simple criteria, or that they are used for different purposes... functional programming has more scientific/academic origins & uses and is less common in industry, so the question also sets up an "industry vs academia" unresolvable pov/conflict. one classic ref that shows OOP sytle in functional programming, SICP book/MIT – vzn Apr 27, 2014 at 15:19

"*OO makes code understandable by encapsulating moving parts. FP makes code understandable by minimizing moving parts.*" --Micheal Feathers, 2010 – jaco0646 Mar 17, 2020 at 0:19

4 Answers

Sorted by: Highest score (default)

1313

> When do you choose functional programming over object oriented?

When you anticipate a different kind of software evolution:

- Object-oriented languages are good when you have a fixed set of *operations* on *things*, and as your code evolves, you primarily add new things. This can be accomplished by adding new classes which implement existing methods, and the existing classes are left alone.
- Functional languages are good when you have a fixed set of *things*, and as your code evolves, you primarily add new *operations* on existing things. This can be accomplished by adding new functions which compute with existing data types, and the existing functions are left alone.

When evolution goes the wrong way, you have problems:

- Adding a new operation to an object-oriented program may require editing many class definitions to add a new method.
- Adding a new kind of thing to a functional program may require editing many function definitions to add a new case.

This problem has been well known for many years; in 1998, Phil Wadler dubbed it the "expression problem". Although some researchers think that the expression problem can be addressed with such language features as mixins, a widely accepted solution has yet to hit the mainstream.

> What are the typical problem definitions where functional programming is a better choice?

Functional languages excel at manipulating symbolic data in tree form. A favorite example is compilers, where source and intermediate languages change seldom (mostly the same *things*), but compiler writers are always adding new translations and code improvements or optimizations (new operations on things). Compilation and translation more generally are "killer apps" for functional languages.

Share Improve this answer Follow

edited May 9, 2010 at 8:11

community wiki
2 revs, 2 users 97%
Norman Ramsey

139 There is some serious zen behind this answer. I think it illuminates the fact that certain OOP design patterns (Visitor) are actually hacks which attempt to overcome the problem of adding new operations.
– Jacobs Data Solutions Mar 26, 2013 at 16:08

67 In JavaScript, you can have all of the things. – Erik Reppen Nov 15, 2013 at 17:38

70 @ErikReppen at which point the question arises, when do you choose to use the functional features, and when do you choose to use the object-oriented features? – Norman Ramsey Nov 23, 2013 at 0:38

7 @NormanRamsey It's not at all uncommon to mix it up in JS and first-class functions are tied to a lot of JS OOP-related features. JS's array sorting takes functions as an arg which can produce some powerful data structures. Closures + a passed function is used to keep jquery objects very lightweight memory-speaking since most methods are just references. Etc... – Erik Reppen Nov 23, 2013 at 0:55

13 @NormanRamsey: Very good answer, along the lines of SICP. According to this classification, functional and procedural programming are grouped together on the opposite side of object-oriented programming. This could explain the boom of OOP during the late 1980-ies early 1990-ies: when GUIs became mainstream OOP proved to be a good approach for modeling them because you normally have a fixed set of operations (paint, open, close, resize) and a growing number of widgets. Of course, this does not mean that OOP is better than procedural for any application, as you illustrated. – Giorgio Jan 9, 2015 at 17:54

195

You don't necessarily have to choose between the two paradigms. You can write software with an OO architecture using many functional concepts. **FP and OOP are orthogonal in nature**.

Take for example C#. You could say it's mostly OOP, but there are many FP concepts and constructs. If you consider **Linq**, the most important constructs that permit Linq to exist are functional in nature: **lambda expressions**.

Another example, F#. You could say it's mostly FP, but there are many OOP concepts and constructs available. You can define classes, abstract classes, interfaces, deal with inheritance. You can even use mutability when it makes your code clearer or when it dramatically increases performance.

Many modern languages are multi-paradigm.

Recommended readings

As I'm in the same boat (OOP background, learning FP), I'd suggest you some readings I've really appreciated:

- **Functional Programming for Everyday .NET Development**, by Jeremy Miller. A great article (although poorly formatted) showing many techniques and practical, real-world examples of FP on C#.
- **Real-World Functional Programming**, by Tomas Petricek. A great book that deals mainly with FP concepts, trying to explain what they are, when they should be used. There are many examples in both F# and C#. Also, Petricek's blog is a great source of information.

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12

community wiki
5 revs, 2 users 97%
Bruno Reis

1 Not forgetting you can mix F# and C# code together so you can leverage the best of each – Paolo Jan 16, 2010 at 21:44

1 Your answer is a great example of the power of .Net. It shows it is able to leverage the powers of both paradigms. – Dykam Jan 16, 2010 at 21:53

5 Heh, sorry if I started some flaming. I didn't mean to say other platforms are less powerfull, just that .NET doesn't only support OOP. For example it has tail call optimization. – Dykam Jan 16, 2010 at 22:26

3 I dont think FP and OOP are orthogonal. Using functional constructs is not the same as functional programming. Sure using functional constructs like lambdas in Linq makes it less OO, but it still isn't functional programming. FP is where functions are first class citizens where as OOP is when classes are first class building blocks (or something to that effect - I do realise there are many kinds of OOP). Imo, the right phrasing is "there are multi-paradigm languages which lets you write both OOP constructs as well as FP constructs making both less OOP and less FP in the process". – nawfal Jul 1, 2015 at 16:59

7 @nawfal, until you can point at some intrinsic features in the two paradigms and say that they're incompatible, they're orthogonal: that is the heart of the discussion. FP is incompatible with imperative programming by definition, but OOP is not limited to imperative programming. The reason we have different words for these concepts is so that we can

talk about them: when you lump them together, you just force us to needlessly come up with new words. – user201891 Feb 10, 2016 at 6:16

33

Object Oriented Programming offers:

1. Encapsulation, to
    - control mutation of internal state
    - limit coupling to internal representation
2. Subtyping, allowing:
    - substitution of compatible types (polymorphism)
    - a crude means of sharing implementation between classes (implementation inheritance)

Functional Programming, in Haskell or even in Scala, can allow substitution through more general mechanism of type classes. Mutable internal state is either discouraged or forbidden. Encapsulation of internal representation can also be achieved. See Haskell vs OOP for a good comparison.

Norman's assertion that "Adding a new kind of thing to a functional program may require editing many function definitions to add a new case." depends on how well the functional code has employed type classes. If Pattern Matching on a particular Abstract Data Type is spread throughout a codebase, you will indeed suffer from this problem, but it is perhaps a poor design to start with.

EDITED Removed reference to implicit conversions when discussing type classes. In Scala, type classes are encoded with implicit parameters, not conversions, although implicit conversions are another means to acheiving substitution of compatible types.

Share Improve this answer Follow

edited Feb 18, 2010 at 14:08

community wiki
2 revs
retronym

3 Typeclasses are not a mechanism for implicit conversion to other types. They're a description of a set of functions defined for a type so as to provide a form of polymorphism. The closest thing from Java-style OOP would be interfaces, though Haskell typeclasses have some important differences. – Zak Feb 18, 2010 at 8:48

28

1. If you're in a heavily concurrent environment, then pure functional programming is useful. The lack of mutable state makes concurrency almost trivial. See Erlang.
2. In a multiparadigm language, you may want to model some things functionally if the existence of mutable state is must an implementation detail, and thus FP is a good model for the problem domain. For example, see list comprehensions in Python or

std.range in the D programming language. These are inspired by functional programming.

Share Improve this answer

Follow

answered Jan 16, 2010 at 21:41

dsimcha

68.6k ● 55 ● 219 ● 340

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.