OOP vs Functional Programming vs Procedural [closed]

Asked 15 years, 10 months ago Modified 4 years, 3 months ago

Viewed 207k times

253

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 7 years ago.

Improve this question

What are the differences between these programming paradigms, and are they better suited to particular problems or do any use-cases favour one over the others?

Architecture examples appreciated!

oop functional-programming paradigms procedural-programming

Share edited Sep 16, 2020 at 9:15

Improve this question

Follow

community wiki
10 revs, 2 users 100%
Ashutosh Singh-MVP
SharePoint

This isn't a full answer, but I write a bit about how "functional" affects/contrasts "OO" style (in the context of F#) here: lorgonblog.spaces.live.com/blog/cns!701679AD17B6D310!511.entry – Brian Feb 16, 2009 at 5:39

You might consider reading this! There are many examples of when to use which ant what is the main differences, pros/cons etc – Nikita Ignatov Feb 16, 2009 at 6:14

2 Also see: - What is the difference between procedural programming and functional programming? - Can someone give me examples of functional programming vs imperative/procedural programming? - truly understanding the difference between procedural and functional – WReach Mar 12, 2011 at 5:44

1 **See also:** stackoverflow.com/questions/1530868 – dreftymac Mar 4, 2017 at 19:46

Uncle Bob has tweeted about this. And also here. – jaco0646 Mar 16, 2020 at 21:46

7 Answers

Sorted by: Highest score (default)

142

All of them are good in their own ways - They're simply different approaches to the same problems.

In a purely procedural style, data tends to be highly decoupled from the functions that operate on it.

In an object oriented style, data tends to carry with it a collection of functions.

In a functional style, data and functions tend toward having more in common with each other (as in Lisp and Scheme) while offering more flexibility in terms of how functions are actually used. Algorithms tend also to be defined in terms of recursion and composition rather than loops and iteration.

Of course, the language itself only influences which style is preferred. Even in a pure-functional language like Haskell, you can write in a procedural style (though that is highly discouraged), and even in a procedural language like C, you can program in an object-oriented style (such as in the GTK+ and EFL APIs).

To be clear, the "advantage" of each paradigm is simply in the modeling of your algorithms and data structures. If, for example, your algorithm involves lists and trees, a functional algorithm may be the most sensible. Or, if, for example, your data is highly structured, it may make more sense to compose it as objects if that is the native paradigm of your language - or, it could just as easily be written as a functional abstraction of monads, which is the native paradigm of languages like Haskell or ML.

The choice of which you use is simply what makes more sense for your project and the abstractions your language supports.

Share Improve this answer Follow

edited Jun 30, 2016 at 4:04

Steve Ladavich
3,562 ● 1 ● 21 ● 27

answered Feb 16, 2009 at 7:18

greyfade
25.6k ● 7 ● 66 ● 80

6 What you said, does not seem to reflect what you wrote. You say they don't have "pros and cons", and then say how they are different approaches. Why would someone choose one approach over another, based on any given situation? strengths and weaknesses, pros and cons, whatever you call them they do exist! I'm not saying one is inherently better, and neither did you. I believe that is what you really were trying to say. unless you really believe any chosen approach doesn't have positives and negatives, relative to another approch. – J. M. Becker Oct 19, 2011 at 15:13

1 @TechZilla: I agree my wording was poor, but what I meant is that there's not really a list of features that you can point to and say language X is better than Y without qualifying that language X is better suited to writing algorithm U and language Y might be better suited to writing algorithm V, while both can easily be implemented in either language. – greyfade Oct 19, 2011 at 15:54

2 The difference between procedural and functional is not clear to me. I'm learning Scheme/Rackett at College, but I really can't see the big difference between it and procedural C or PHP, could you give some example? – Leonel Apr 1, 2012 at 2:39

7 @Leonel: The biggest difference most people will cite is that in procedural languages, you might use a for loop, but

functional languages, there's no such thing - instead, you use recursive calls to functions to perform the same task. Functional languages also make functions first-class objects - you can pass them around like you would a number - but you can't do that in C (and PHP's support for it is broken). – greyfade Apr 24, 2012 at 16:38

2 @tastro: When one paradigm makes more sense than the other. That's really all. Sometimes it makes more sense to model your code as the composition of functions, and sometimes it makes more sense to model your data as objects. There are many ways to express algorithms and data structures. OOP and functional happen to be two of those things. – greyfade Jul 14, 2014 at 5:46

25

I think the available libraries, tools, examples, and communities completely trumps the paradigm these days. For example, ML (or whatever) might be the ultimate all-purpose programming *language* but if you can't get any good libraries for what you are doing you're screwed.

For example, if you're making a video game, there are more good code examples and SDKs in C++, so you're probably better off with that. For a small web application, there are some great Python, PHP, and Ruby frameworks that'll get you off and running very quickly. Java is a great choice for larger projects because of the compile-time checking and enterprise libraries and platforms.

It used to be the case that the standard libraries for different languages were pretty small and easily replicated - C, C++, Assembler, ML, LISP, etc.. came with the basics, but tended to chicken out when it came to standardizing on things like network communications, encryption, graphics, data file formats (including XML), even basic data structures like balanced trees and hashtables were left out!

Modern languages like Python, PHP, Ruby, and Java now come with a far more decent standard library and have many good third party libraries you can easily use, thanks in great part to their adoption of namespaces to keep libraries from colliding with one another, and garbage collection to standardize the memory management schemes of the libraries.

Share Improve this answer Follow

answered Feb 16, 2009 at 6:49

community wiki
Dobes

5 Python, ruby,... do not have "standard" libraries like C or LISP, since they are single implementation languages. Python is what Guido says, there is no standard. Any particular C or LISP (or whatever) implementation nowadays comes with a large set of libraries beyond the standard ones. – Dan Andreatta May 13, 2010 at 11:03

9 The question was about the differences between object oriented, functional, and procedural programming. While the languages mentioned in these answers certainly lend themselves to one of these approaches, the answer makes no mention of any of these concepts... Regardless of whether or not "the available libraries [...] [trump] the paradigm", that doesn't answer the question at hand, thereby side-stepping

what is a completely valid question. – rinogo Oct 25, 2013 at 23:46

1 ircmaxell's related rant: blog.ircmaxell.com/2012/07/oop-vs-procedural-code.html – rinogo Oct 26, 2013 at 0:16

23

These paradigms don't have to be mutually exclusive. If you look at python, it supports functions and classes, but at the same time, everything is an object, including functions. You can mix and match functional/oop/procedural style all in one piece of code.

What I mean is, in functional languages (at least in Haskell, the only one I studied) there are no statements! functions are only allowed one expression inside them!! BUT, functions are first-class citizens, you can pass them around as parameters, along with a bunch of other abilities. They can do powerful things with few lines of code.

While in a procedural language like C, the only way you can pass functions around is by using function pointers, and that alone doesn't enable many powerful tasks.

In python, a function is a first-class citizen, but it can contain arbitrary number of statements. So you can have a function that contains procedural code, but you can pass it around just like functional languages.

Same goes for OOP. A language like Java doesn't allow you to write procedures/functions outside of a class. The only way to pass a function around is to wrap it in an object that implements that function, and then pass that object around.

In Python, you don't have this restriction.

Share Improve this answer Follow

edited Feb 18, 2009 at 23:51

community wiki
2 revs
hasen j

I believe you meant "these paradigms don't have to be mutually exclusive". The 3 of them *are* orthogonal in that ideally you could use one, 2 or the 3 of them in a single program (if your language allows it). – Joe Pineda Feb 18, 2009 at 21:45

Yea I guess mutually exclusive is a better term for that! thanks – hasen Feb 18, 2009 at 23:50

16

For GUI I'd say that the Object-Oriented Paradigma is very well suited. The Window is an Object, the Textboxes are Objects, and the Okay-Button is one too. On the other Hand stuff like String Processing can be done with much less overhead and therefore more straightforward with simple procedural paradigma.

I don't think it is a question of the language neither. You can write functional, procedural or object-oriented in almost any popular language, although it might be some additional effort in some.

Share Improve this answer

Follow

edited Jan 7, 2014 at 10:49

BenMorel

36.4k ● 51 ● 202 ● 334

answered Feb 16, 2009 at 10:47

panschk

3,272 ● 3 ● 26 ● 20

19 Tempted to downvote for perpetuating the misconception that "Object = GUI widget", but I'll refrain. OOP works just as well for representing abstract concepts, such as "UserAccount" or "PendingSale", as for visible interface elements like "Window" and "Button". – Dave Sherohman Feb 16, 2009 at 11:19

6 I wrote that a window can be an object. How do you draw the conclusion that every Object is a window from there? It was just an example. Of course OOP can be used just as well to model abstract entities and their relationship. Anyway, thanks for not downvoting. I dont have many points anyway:D – panschk Feb 16, 2009 at 13:26

6 -1. OOP doesn't have anything to do with GUI's. Ideally, the best method for designing guis is using an external text file (e.g. HTML). While things like string-processing are actually much better done with objects. (think about strings in C)!! – hasen Feb 17, 2009 at 17:33

1 I don't know, maybe I'm just used to programming with objects to realise it. But how would you do some interactive stuff like changing the value in TextBox X when the value of TextBox Y has been altered without using objects? Okay, you could simply use global vars for everything... – panschk Feb 17, 2009 at 21:58

1 String processing is awesomely done in perl (100x better than in Java, C++ or C#) yet the string functionality of the

language is absolutely NOT object oriented. C's string handling was terrible, but then C is not the only procedural language (nor the best). – Joe Pineda Feb 18, 2009 at 21:43

6

In order to answer your question, we need two elements:

1. Understanding of the characteristics of different architecture styles/patterns.
2. Understanding of the characteristics of different programming paradigms.

A list of software architecture styles/pattern is shown on the software architecture article on Wikipeida. And you can research on them easily on the web.

In short and general, Procedural is good for a model that follows a procedure, OOP is good for design, and Functional is good for high level programming.

I think you should try reading the history on each paradigm and see why people create it and you can understand them easily.

After understanding them both, you can link the items of architecture styles/patterns to programming paradigms.

Share Improve this answer Follow

answered Feb 16, 2009 at 8:40

Gabriel Chung
1,567 ● 1 ● 20 ● 31

2

I think that they are often not "versus", but you can combine them. I also think that oftentimes, the words you mention are just buzzwords. There are few people who actually know what "object-oriented" means, even if they are the fiercest evangelists of it.

Share Improve this answer Follow

answered Feb 16, 2009 at 13:58

community wiki
Svante

1

One of my friends is writing a graphics app using NVIDIA CUDA. Application fits in very nicely with OOP paradigm and the problem can be decomposed into modules neatly. However, to use CUDA you need to use C, which doesn't support inheritance. Therefore, you need to be clever.

a) You devise a clever system which will emulate inheritance to a certain extent. It can be done!

i) You can use a hook system, which expects every child C of parent P to have a certain override for function F. You can make children register their overrides, which will be stored and called when required.

ii) You can use struct memory alignment feature to cast children into parents.

This can be neat but it's not easy to come up with future-proof, reliable solution. You will spend lots of time designing the system and there is no guarantee that you won't run into problems half-way through the project. Implementing multiple inheritance is even harder, if not almost impossible.

b) You can use consistent naming policy and use divide and conquer approach to create a program. It won't have any inheritance but because your functions are small, easy-to-understand and consistently formatted you don't need it. The amount of code you need to write goes up, it's very hard to stay focused and not succumb to easy solutions (hacks). However, this ninja way of coding is the C way of coding. Staying in balance between low-level freedom and writing good code. Good way to achieve this is to write prototypes using a functional language. For example, Haskell is extremely good for prototyping algorithms.

I tend towards approach b. I wrote a possible solution using approach a, and I will be honest, it felt very unnatural using that code.

Share Improve this answer Follow

edited Feb 20, 2009 at 12:20

community wiki
3 revs
sneg

The first c++ copmpilers were nothing more than pre-processors that generated C code. As such, ALL features of c++ - including multiple inheritance, can be implemented using C. (emulating c++ exception handling in C would require some kind of platform support for exceptions, but c++ implementations require that too, so I dont think it makes the basic idea invalid). – Chris Becke May 23, 2009 at 13:37

2 @Chris Becke your answer is too philosophical. For one, C has multiple standards (through the years), barely any of them completely adopted by C compilers let alone c++. To say that C++ is a superset of C because it uses C syntax doesn't mean much because you can't even write code for one C compiler which compiles in another C compiler without significant effort. Furthermore, there are language features (type systems, OOD support) offered in other languages which would be impossible to implement in C without using C to design a new language (which is exactly why there are 'new languages') – Sprague Jan 5, 2011 at 21:02

You know. I cannot see any more what relevance my comment has to this post at all. :P – Chris Becke Jan 6, 2011 at 5:54

Cuda has supported C++ for some time now.
– Aryeh Leib Taurog Mar 2, 2015 at 17:07