LBYL vs EAFP in Java?

Asked 15 years, 11 months ago Modified 7 years, 7 months ago Viewed 26k times

81

I was recently teaching myself Python and discovered the LBYL/EAFP idioms with regards to error checking before code execution. In Python, it seems the accepted style is EAFP, and it seems to work well with the language.

LBYL (***L****ook* ***B****efore* ***Y****ou* ***L****eap*):

```
def safe_divide_1(x, y):
    if y == 0:
        print "Divide-by-0 attempt detected"
        return None
    else:
        return x/y
```

EAFP (*it's* ***E****asier to* ***A****sk* ***F****orgiveness than* ***P****ermission*):

```
def safe_divide_2(x, y):
    try:
        return x/y
    except ZeroDivisionError:
        print "Divide-by-0 attempt detected"
        return None
```

My question is this: I had never even heard of using EAFP as the primary data validation construct, coming from a Java and C++ background. Is EAFP something that is wise to use in Java? Or is there too much overhead from exceptions? I know that there is only overhead when an exception is actually thrown, so I'm unsure as to why the simpler method of EAFP is not used. Is it just preference?

java python error-handling idioms

Share
Improve this question
Follow

edited Jul 12, 2015 at 18:23
Eric
97.5k ● 54 ● 254 ● 385

asked Jan 1, 2009 at 9:50
ryeguy
66.8k ● 60 ● 201 ● 263

5 Answers

Sorted by: Highest score (default)

If you are accessing files, EAFP is more reliable than LBYL, because the operations involved in LBYL are not atomic, and the file system might change between the time

149

you look and the time you leap. Actually, the standard name is TOCTOU - Time of Check, Time of Use; bugs caused by inaccurate checking are TOCTOU bugs.

Consider creating a temporary file that must have a unique name. The best way to find out whether the chosen file name exists yet is to try creating it - making sure you use options to ensure that your operation fails if the file does already exist (in POSIX/Unix terms, the O_EXCL flag to `open()`). If you try to test whether the file already exists (probably using `access()`), then between the time when that says "No" and the time you try to create the file, someone or something else may have created the file.

Conversely, suppose that you try to read an existing file. Your check that the file exists (LBYL) may say "it is there", but when you actually open it, you find "it is not there".

In both these cases, you have to check the final operation - and the LBYL didn't automatically help.

(If you are messing with SUID or SGID programs, `access()` asks a different question; it may be relevant to LBYL, but the code still has to take into account the possibility of failure.)

Share Improve this answer Follow

answered Jan 1, 2009 at 17:52
Jonathan Leffler
752k ●145 ●946 ●1.3k

3 Great example, Jonathan. This probably makes a lot of sense to java developers who have dealt with concurrent programming and the double-checked lock idiom. – David Mann May 9, 2014 at 3:32

This is a good point but I don't believe this really has to do with LBYL vs EAFP. Any time you open a file for instance you are basically doing the same thing, one operation that returns an error, you're not checking if you can open it before actually doing it. Such code I argue can be LBYL: multiple function calls, each function call can fail and each is checked on the spot. TOCTOU is a more general problem. Also, your answer addresses a correctness concern whereas I believe the question was about instances where both solutions would be correct. – ctn Jul 11, 2019 at 12:09

60

In addition to the relative cost of exceptions in Python and Java, keep in mind that there's a difference in philosophy / attitude between them. Java tries to be very strict about types (and everything else), requiring explicit, detailed declarations of class/method signatures. It assumes that you should know, at any point, exactly what type of object you're using and what it is capable of doing. In contrast, Python's "duck typing" means that you don't know for sure (and shouldn't care) what the manifest type of an object is, you only need to care that it quacks when you ask it to. In this kind of permissive environment, the only sane attitude is to presume that things will work, but be ready to deal with the consequences if they don't. Java's natural restrictiveness doesn't fit well with such a casual approach. (This is not intended to disparage either approach or language, but rather to say that these attitudes are part of each language's idiom, and copying idioms between different languages can often lead to awkwardness and poor communication...)

Share Improve this answer Follow

answered Jan 2, 2009 at 23:31
Jeff Shannon
10.1k ● 1 ● 17 ● 7

5 Also, oranlooney.com/lbyl-vs-eafp provides a good set of pros and cons for each approach. – Tim Lewis Jun 23, 2011 at 15:24

I disagree that the loose typing in Python makes it more or less sensible to use EAFP. When you ask for forgiveness, you ask to be forgiven for expected cases. For example, if a "cancel" method is going to cancel an object, we will catch for the exceptions we expect to come back. We expect cancel may fail because the object has active relationships that prevent it from being canceled, and we want to communicate that back to the UI. If the cancel method fails because of some unforseen divide by zero scenerio, we want it to fail normally like any other error in the program. – David Baucum Jun 12, 2015 at 18:08

1 the link doesn't work anymore so someone could read the article here instead [web.archive.org/web/20161208191318/http://www.oranlooney.com/…](web.archive.org/web/20161208191318/http://www.oranlooney.com/) – cookiemonster Feb 27, 2020 at 14:07

13

Exceptions are handled more efficiently in Python than in Java, which is at least *partly* why you see that construct in Python. In Java, it's more inefficient (in terms of performance) to use exceptions in that way.

Share Improve this answer Follow

answered Jan 1, 2009 at 14:30
mipadi
410k ● 90 ● 531 ● 487

2 mipadi, do you have any insight as to how python accomplishes this? – duffymo Jan 1, 2009 at 15:13

3 @duffymo I had the same question, and found it here: [stackoverflow.com/questions/598157/…](stackoverflow.com/questions/598157/) – Tim Lewis Jun 23, 2011 at 15:01

2 @duffymo most of the discussion centers around LBYL vs EAFP, but one of the answers linked to how exceptions are actually implemented in CPython: docs.python.org/c-api/intro.html#exceptions – Tim Lewis Jun 23, 2011 at 15:16

Consider these code snippets:

10

```
def int_or_default(x, default=0):
    if x.isdigit():
        return int(x)
    else:
        return default

def int_or_default(x, default=0):
    try:
        return int(x)
    except ValueError:
        return default
```

They both look correct, right? But one of them isn't.

The former, using LBYL, fails because of a subtle distinction between `isdigit` and `isdecimal`; when called with the string "①^{23}4,$_{5}$", it will throw an error rather than correctly return the default value.

The later, using EAFTP, results in correct handling, by definition. There is no scope for a behavioural mismatch, because the code that needs the requirement *is* the code that asserts that requirement.

Using LBYL means taking internal logic and copying them into *every* call-site. Rather than having one canonical encoding of your requirements, you get a free chance to mess up every single time you call the function.

It's worth noting that EAFTP *isn't* about exceptions, and Java code especially should not be using exceptions pervasively. It is about giving the right job to the right block of code. As an example, using `Optional` return values is a perfectly valid way of writing EAFTP code, and is far more effective for ensuring correctness than LBYL.

Share Improve this answer Follow

answered Apr 28, 2017 at 18:26

Veedrac
60k ● 15 ● 116 ● 174

That's just because in the first case you did not choose a suitable function for your case. Had you chosen a function that checks exactly what you want, the problem that's given in your argument would not exist. – Silidrone Jun 25, 2022 at 9:52

@Silidron *All* programming errors are from not writing exactly correct code! That's the point! Why have the opportunity to make this mistake, every time the function is called from every programmer who uses it, when the function is already perfectly happy to integrate this check? – Veedrac Jun 25, 2022 at 13:59

Well there's an opportunity to forget throwing ValueError inside the `int()` casting, and then the second example breaks too. – Silidrone Jun 25, 2022 at 14:24

@Silidrone Yes, but that opportunity happens once, inside the code that's already responsible for knowing and dealing with all the particular details of `int` parsing. Versus having that opportunity at every call site, by people with potentially much less specialized knowledge about these details. The person writing `int` will naturally know the difference between digits and decimals, because that's their responsibility, and the code is already parsing it. Most people calling `int` won't, because they're solving some other problem, and they just want it to work and get out of the way. – Veedrac Jun 26, 2022 at 4:48

1 @Silidrone LBYL is fundamentally **broken**. You're doing action X to check if action Y will work. But action X is **not** action Y, so it **can't** definitively tell you action Y will or won't work - that's in addition to LBYL also being broken because it's a TOCTOU bug. This question is a perfect example of the fundamental flaw in LBYL. – Andrew Henle Oct 29, 2022 at 12:38

8

Personally, and I think this is backed up by convention, EAFP is never a good way to go. You can look at it as an equivalent to the following:

```
if (o != null)
    o.doSomething();
else
    // handle
```

as opposed to:

```
try {
    o.doSomething()
}
catch (NullPointerException npe) {
    // handle
}
```

Moreover, consider the following:

```
if (a != null)
    if (b != null)
        if (c != null)
            a.getB().getC().doSomething();
        else
            // handle c null
    else
        // handle b null
else
    // handle a null
```

This may look a lot less elegant (and yes this is a crude example - bear with me), but it gives you much greater granularity in handling the error, as opposed to wrapping it all in a try-catch to get that `NullPointerException`, and then try to figure out where and why you got it.

The way I see it EAFP should never be used, except for rare situations. Also, since you raised the issue: **yes, the try-catch block does incur some overhead** even if the exception is not thrown.

Share

Improve this answer

Follow

edited Jan 1, 2009 at 11:17

answered Jan 1, 2009 at 9:57

Yuval Adam
165k ● 95 ● 315 ● 404

41 This sort of EAFP depends in part on whether the exceptions you test for are going to occur very often. If they are unlikely, then EAFP is reasonable. If they are common, then LBYL may be better. The answer probably also depends on the exception handling paradigm available. In C, LBYL is necessary. – Jonathan Leffler Jan 1, 2009 at 21:26

11 a "common exception" is not an exception at all, so of course LBYL will be preferred in that case, don't you think? – Fredy Treboux Apr 28, 2011 at 18:33

9 Saying "the try-catch block does incur some overhead even if the exception is not thrown" is like saying "the method definition does incur some overhead even if the method is not called". In other words, the overhead is negligible until the method/exception is invoked. Erickson explained it best when I asked about this. – Iain Samuel McLean Elder Nov 8, 2012 at 21:19

13 I utterly and completely disagree. Logically, EAFP *is* a good way to do validation. Because if the check that throws the exception is already there anyway, additional check can only get out of sync and saves nothing. However due to comparably slow exceptions in C++, Java and C#, LBYL is needed when the failure is expected to happen but rarely. – Jan Hudec Jan 11, 2013 at 12:24

21 EAFP can also have better security properties (eliminating some race conditions) in multi-threaded environments. – Mark E. Haase Jul 2, 2013 at 17:12