Is late binding consistent with the philosophy of "readibility counts"?

Asked 15 years, 11 months ago Modified 15 years, 11 months ago Viewed 852 times

1

I am sorry all - I am not here to blame Python. This is just a reflection on whether what I believe is right. Being a Python devotee for two years, I have been writing only small apps and singing Python's praises wherever I go. I recently had the chance to read Django's code, and have started wondering if Python really follows its "readability counts" philosophy. For example,

```
class A:
    a = 10
    b = "Madhu"

    def somemethod(self, arg1):
        self.c = 20.22
        d = "some local variable"
        # do something
        ....
        ...
    def somemethod2 (self, arg2):
        self.c = "Changed the variable"
        # do something 2
        ...
```

It's difficult to track the flow of code in situations where the instance variables are created upon use (i.e. `self.c` in the above snippet). It's not possible to see which instance variables are defined when reading a substantial amount of code written in this manner. It becomes very frustrating even when reading a class with just 6-8 methods and not more than 100-150 lines of code.

I am interested in knowing if my reading of this code is skewed by C++/Java style, since most other languages follow the same approach as them. Is there a Pythonic way of reading this code more fluently? What made Python developers adopt this strategy keeping "readability counts" in mind?

python **readability**

Share

Improve this question

Follow

edited Jan 12, 2009 at 7:46

cdleary

71.3k ● 55 ● 164 ● 194

asked Jan 11, 2009 at 20:47

Madhusudan CS

I think when it comes to dynamic languaes, and this is not necessarily a bad thing, in reality, reability does not count. Really, you should probably be happy that it is not very readable, otherwise it is unlikely that it would be such a popular language. – BobbyShaftoe Jan 11, 2009 at 21:21

@BobbySaftoe: Although you /can/ code like this, it's generally considered bad practice to do so, because of the readability concerns of the OP. Just because you /can/ code like this in Python doesn't mean that you should. We typically like to be explicit and bind in **init**. – cdleary Jan 12, 2009 at 23:57

8 Answers

Sorted by: Highest score (default)

14

The code fragment you present is fairly atypical (which might also because you probably made it up):

- you wouldn't normally have an instance variable (self.c) that is a floating point number at some point, and a string at a different point. It should be either a number or a string all the time.
- you normally don't bring instance variables into life in an arbitrary method. Instead, you typically have a constructor (__init__) that initializes all variables.
- you typically don't have instance variables named a, b, c. Instead, they have some speaking names.

With these fixed, your example would be much more readable.

Share Improve this answer Follow

answered Jan 11, 2009 at 21:21

Martin v. Löwis

127k ● 20 ● 203 ● 236

Quite agreed with the last point. But we can only hope that point 1 and 2 won't happen normally. If it happens we are messed up. There is no rule enforcing it in Python but there are rules in C++/Java. The second point is right, but I have seen a lot of programs not following the pattern. – Madhusudan CS Jan 11, 2009 at 21:30

1 I think you misunderstand the Zen, then. "Readability counts" does not mean that Python enforces it, at all times. Instead, it means that Python authors should write their code with readability in mind. – Martin v. Löwis Jan 11, 2009 at 21:35

@Madhusudan: FWIW, C++/Java only enforce typing; nothing prevents an integer used for an index being re-used for screen coordinates then file size... Apart from the programmer. Naming helps with this... – Shog9 Jan 11, 2009 at 23:05

+1: The language encourages readability. Fussy static type checking doesn't encourage readability. People write perfectly rotten Java all the time. – S.Lott Jan 12, 2009 at 1:40

5

A sufficiently talented miscreant can write unreadable code in any language. Python attempts to impose some rules on structure and naming to nudge coders in the right direction, but there's no way to force such a thing.

For what it's worth, I try to limit the scope of local variables to the area where they're used in *every* language that i use - for me, not having to maintain a huge mental dictionary makes re-familiarizing myself with a bit of code much, much easier.

Share Improve this answer Follow

answered Jan 11, 2009 at 21:20

Shog9

159k • 36 • 235 • 240

> "A sufficiently talented miscreant can write unreadable code in any language." Amen to that, brother. – Brian Clapper Jan 11, 2009 at 21:33

3

I agree that what you have seen can be confusing and ought to be accompanied by documentation. But confusing things can happen in any language.

In your own code, you should apply whatever conventions make things easiest for you to maintain the code. With respect to this particular issue, there are a number of possible things that can help.

- Using something like Epydoc, you can specify all the instance variables a class will have. Be scrupulous about documenting your code, and be equally scrupulous about ensuring that your code and your documentation remain in sync.
- Adopt coding conventions that encourage the kind of code you find easiest to maintain. There's nothing better than setting a good example.
- Keep your classes and functions small and well-defined. If they get too big, break them up. It's easier to figure out what's going on that way.
- If you **really** want to insist that instance variables be declared before referenced, there are some metaclass tricks you can use. e.g., You can create a common base class that, using metaclass logic, enforces the convention that only variables that are declared when the subclass is declared can later be set.

Share Improve this answer Follow

answered Jan 11, 2009 at 21:24

Brian Clapper

26.2k • 7 • 66 • 66

3

This problem is easily solved by specifying coding standards such as declaring all instance variables in the **init** method of your object. This isn't really a problem with python as much as the programmer.

Share Improve this answer Follow

answered Jan 11, 2009 at 21:39

Yi.

322 ● 3 ● 11

2

If what the code is doing becomes mysterious for some reason .. there should either be comments or the function names should make it obvious.

This is just my opinion though.

Share Improve this answer Follow

answered Jan 11, 2009 at 20:53

hasen

166k ● 66 ● 196 ● 233

2

I personally think not having to declare variables is one of the dangerous things in Python, especially when doing classes. It is all too easy to accidentally create a variable by simple mistyping and then boggle at the code at length, unable to find the mistake.

Share Improve this answer Follow

answered Jan 11, 2009 at 21:00

Gnudiff

4,305 ● 1 ● 25 ● 28

> In Python, there are no "variables" per-se. Names are references bound to objects. It's unhealthy to concern yourself with memory slots. Once your mindset changes, it's easier to cope - use descriptive names, documentation, conventions, standard interfaces, trust the clients of your libs (gasp!)... – Jeremy Brown Jan 12, 2009 at 0:54

2

Adding a property just before you need it will prevent you from using it before it's got a value. Personally, I *always* find classes hard to follow just from reading source - I read the documentation and find out what it's supposed to do, and then it usually makes sense when I read the source again.

Share Improve this answer Follow

answered Jan 11, 2009 at 21:05

too much php

90.8k ● 36 ● 131 ● 139

0

The fact that such stuff is allowed is only useful in rare times for prototyping; while Javascript tends to allow anything and maybe such an example could be considered normal (I don't really know), in Python this is mostly a negative byproduct of omission of type declaration, which can help speeding up development - if you at some point change your mind on the type of a variable, fixing type declarations can take more time than the fixes to actual code, in some cases, including the renaming of a type, but also cases where you use a different type with some similar methods and no superclass/subclass relationship.

Share Improve this answer Follow

answered Jan 11, 2009 at 22:05

Blaisorblade

6,488 ● 1 ● 44 ● 77