

## Re: Basic inheritance question

---

*Source:* <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2008-01/msg02429.html>

---

- *From:* Bruno Desthuilliers <[bruno.42.desthuilliers@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:bruno.42.desthuilliers@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Mon, 21 Jan 2008 13:48:45 +0100
- 

Lie a écrit :

On Jan 16, 9:23 pm, Bjoern Schliessmann <[usenet-mail-0306.20.chr0n...@xxxxxxxxxxxxxxxxxxxxx](mailto:usenet-mail-0306.20.chr0n...@xxxxxxxxxxxxxxxxxxxxx)> wrote:

Lie wrote:

[42.desthuilli...@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:42.desthuilli...@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx) wrote:

I used to systematically use it – like I've always systematically used 'this' in C++ and Java.

And that is what reduces readability.

IMHO not, IOPHO not. This is the nth time ( $n \gg 1$ ) this discussion comes up here. If I have learned one thing from those very lengthy discussions, it's that Python's "self" handling is not going to change.

And ah... yes, I don't wish it go away either. In VB, excessive Me reduces readability, but in Python it doesn't.

A proficient VB/C/Java programmer would frown upon the extra, unneeded garbage as they thought it was clear already that the variable refers to a class-level variable.

C programmers surely have no opinion concerning C because it has no native classes.

C-family, ok?

Well... I'm not sure what "C-family" means, but to me it would consist in C, C++, Objective C, and D.

## Re: Basic inheritance question

Please stop taking my words to its letters.

So we're supposed to actually guess what you really mean ???

Personally, I've seen many C++ programs with complex class designs where it definitely helps to consistently use "this->". I cannot remember all local (and global) variables in bigger methods.

In that case, you have the `_option_` to do it.

Make no sens when maintaining code wrote by someone that didn't use this 'option'.

(snip)

it's the first argument of the function –  
which usually happens to be  
the current instance when the function is  
used as a method.

And that's the point, `self` (or anything you name it) is almost  
always  
the current instance

```
# this is a plain function. In this function,  
# 'obj' can be whatever that happens to have a (numeric)  
# 'stuff' attribute  
def func(obj, arg):  
    return (obj.stuff + arg) / 2.0
```

```
# this is a class with an instance attribute 'stuff'  
class Foo(object):  
    def __init__(self, bar):  
        self.stuff = bar + 42
```

```
# this is another (mostly unrelated) class  
# with a class attribute 'stuff'  
class Bar(object):  
    stuff = 42
```

```
# this is a dummy container class:  
class Dummy(object): pass
```

```
# now let's play:  
import new
```

## Re: Basic inheritance question

```
d = Dummy()
d.stuff = 84
print func(d, 1)

d.baaz = new.instancemethod(func, d, type(d))
print d.baaz(2)

f = Foo(33)
print func(f, 3)
Foo.baaz = func
f.baaz(4)

print func(Bar, 5)
Bar.baaz = classmethod(func)
Bar.baaz(6)
```

and that makes it functionally the same as Me and this in VB and Java.

Depends on the context, cf above !-)

Please again, stop taking letters to the words, I don't meant them to be exactly the same, rather the same would meant that they generally can be considered equal,

If you still think that way after having read the above code, then I can't help. We obviously don't share the same mental model here.

exceptions exists of course. And btw, I don't understand what you meant by your example, they seemed to be a completely OK program for me,

Indeed it's ok (even if totally useless by itself). The point is that 'self' (or whatever you name it) is just and only the first argument of a function, period.

even though it's a bit confusing to follow[2].

Nothing in it should be confusing to anyone having a decent knowledge of Python's object model IMHO.

[2] btw, the reason it's a bit confusing to follow is one of my points: It is a Bad Thing(tm) to use the same name for different

## Re: Basic inheritance question

variables

Where do I "use the same name for different variables" here ?

even in a language like Python that enforce explicit naming of classes

Python doesn't "enforce" explicit name of classes – IIRC, there are ways to instantiate anonymous class objects. But I definitely don't see how this relate to this discussion.

Yes, I know, "please guess what I mean" !-) but sorry, there's no sens discussing a technical point without using accurate and appropriate technical naming of technical concepts invlved.

Most other languages  
1) automatically assign the  
containing class' object

s/containing class' object/current instance/

in a keyword  
(Java: this, VB: Me) behind  
the screen,

That's not very far from what a Python  
method object does –  
automatically assign the current instance to  
something. The difference  
is that Python uses functions to implement  
methods (instead of having  
two distinct constructs), so the only reliable  
way to "inject" the  
reference to the current instance is to pass it  
as an argument to the  
function (instead of making it pop from pure  
air).

It isn't very far, but Python makes it obvious about the  
assignment  
(not behind the screen).

Exactly. And given both the simplicity of the solution and what it let  
you do, that's a *\*very\** GoodThing(tm) IMHO.

I agree, it's a Good Thing but it doesn't make the point less pointy,  
the difference between Me/this and self is just the explicit  
assignment. Other things that is possible because of the explicit

## Re: Basic inheritance question

assignment is just a "coincidence" of design choice.

Are you sure ? As far as I'm concerned, I think that the design choice somehow results from what it makes possible.

(snip pointless discussion wrt/ # of attributes)

And it is always a Bad Thing(tm) to use the same name for two variable in the class and in function (which is the main and only source of possible ambiguity) in ANY language, even in Python.

Ho, yes.... Like, this would be bad ?

```
class Person(object):
def __init__(self, firstname, lastname, birthdate, gender):
self.firstname = firstname
self.lastname = lastname
self.birthdate = birthdate
self.gender = gender
```

C'mon, be serious. It's often hard enough to come with sensible names, why would one have to find synonyms too ? Try to come with something more readable than the above, and let us know. Seriously, this braindead rule about "not using the same name for an attribute and a local var" obviously comes from languages where the "this" ref is optional, and FWIW it's obviously the wrong solution to a real problem (the good solution being, of course, to use the fully qualified name for attributes so there's no possible ambiguity).

The code fragment you've given way above (about the Foo, Bar, bazz, and func) also suffers from the bad habits of using the same name for different variables.

Where ? And how does this answer the question above ?

And it's not a "braindead" rule

The way you express it, and as far as i'm concerned, it is, definitively.

## Re: Basic inheritance question

The example you've given IS the most readable form since the function is `_simple_`, consider a function that have complex codes, possibly calculations instead of simply assigning initial values I'm sure you'd slip up between the `self.*` variables and the `*` variables once or twice,

`*you*` would perhaps have this problem. And you would indeed have this problem in Java or C++. In Python, this problem just don't exist.

possibly becoming the source of hard-to-find bugs.

Consistant and intelligible naming is quite another problem. And it's too much dependant on the context, language, domain and whatnot for any rule like your one above to be universally applyable.

And in languages that doesn't enforce explicit naming of classes,

I still don't understand how all this relates to the naming of class objects ?

Oops, sorry: you meant "in languages that has implicit instance reference available in methods" ? Python doesn't have it, so any rule deriving from this "feature" is out of scope here.

when

there is the two or more same names, the one with the smallest scope is picked, so in `_simple_` functions, the trick of using full qualified names and overloaded local names is still possible and feasible. In complex functions, the trick fails even in Python, because even if Python and our full-concentration-brain is aware of the difference between `self.*` and `*`, our spreaded-concentration-brain that is scanning the code for the source of bugs might get stumbled on the confusing use of `self.*` and `*`.

Here again, `*you*` may have this problem. I don't, since I always used explicit instance reference.

Anyway, I actually know 3 languages (4 if C# works the same) that has this implicit 'this' (or whatever the name) 'feature', and at least 5 that don't. So I'm not sure that the "most other languages" qualifier really applies to point 2 !-)

What's this 5 languages?

## Re: Basic inheritance question

Smalltalk, Python, PHP, javascript, Ruby. I don't remember how Scheme, CLOS and OCaml handle the case.

Among all them, only Javascript is considerably mainstream.

Is that a joke ? PHP is probably the most used language for web apps (server side of course). And Python is certainly not an obscure unknown geek-only language no more. But anyway: you were talking about "most other languages" – not "most mainstream languages".

Are they a mainstream, high-level languages  
or lesser known, low-level languages? C-family, Java, and  
Basic are  
the Big Three of high-level programming language.

None of C, C++, Java nor Basic qualify as "hi-level". C is the lowest possible level above assembly, C++ is often referred to as an "object oriented assembler", Java is way too static, crippled, verbose and unexpressive to qualify as "hi-level" (even if it suffers from some problems usually associated with higher level languages). I won't even comment on basic (is that really a language at all ?).

Your criteria on being high-level is simply just odd.

My criteria on being hi-level seems quite common: automatic memory management, portability, "rich" builtin data types, functions as first class citizens, lexical closures, strong introspection features, strong metaprogramming support, etc...

The rest of the  
world recognizes C-family, Java, and Basic as high-level languages.

C was "hi-level" wrt/ assembler, indeed. And Java is "hi-level" wrt/ C++. Ho, and \_please\_ don't get me started on basic !-)

If I have to say it, Python is actually lower level than Basic.

No comment.

While  
Java is just below Python and C and C++ is just below Java. Why do I consider Basic the highest-level? Because it is the cleanest to scan (no confusing symbols, i.e. no curly braces, no confusing use of

## Re: Basic inheritance question

parens (Python uses (), [], and {}), VB only use ()[3]),

Basic != VB. There are quite a few other basics here.

Now if you choose this criteria, you may want to learn some Lisp dialect.

In what way C++ resembles an assembler?

C++ is some OO stuff bolted on a very close-to-the-metal language itself designed to write operating systems, drivers and other low-level stuff.

Have you ever programmed in assembly?

I did.

How hard is it to create a simple program in assembly? And how hard is it to create a complex program in C++

Roughly the same, thanks to scoping rules, dependencies hell, lack of automatic memory management and overcomplex features.

(which AFAIK is used by hundreds of mega projects including CPython)?

CPython – as the name implies – is written in C.

And have you ever used Basic at all?

I did. And not only VB.

Some programmers would instantly frown upon Basic, simply because they don't know that Basic is "just another language".

I've used at least four different variants of Basic.

## Re: Basic inheritance question

In VB, Me is extremely rarely used,

I used to systematically use it – like I've always systematically used 'this' in C++ and Java.

And that is what reduces readability. A proficient VB/C/Java programmer

There are quite a few proficient C/C++/Java programmers here. As far as I'm concerned, I would not pretend being one – I just have a good enough knowledge of C, Java and (alas) VB to be able to get up to speed in a reasonable time frame.

As a side note, the problem just doesn't exist in C, which has absolutely no support for OO.

When I said C, it might mean C and C-family,

When you say "C", it means "C" to everyone reading you.

so please stop misunderstanding me.

Please learn to express yourself clearly. If you say "X" when you mean "Y", \*you\* are the one responsible for misunderstandings. This is human communication 101 skill.

(snip)

I'm not saying my mindset is better than yours (it has its positives and negatives), in fact I apologize for getting you confused.

My "mindset", as you say, is the one you'll find in each and every technical communication. You can't discuss technical points without a clear, unambiguous naming of technical concepts. And when a community exists, it usually has its own technical idiom, whether highly specific (ie : "member variables" in C++, "attributes" in Python), or more general (ie: C is not the same thing as C++, whenever language you're using).

would frown upon the extra, unneeded garbage as they thought it was clear already that the variable refers to a class-level variable.

In C++, the canonical way to make this "clear" is to use the m\_name convention. There must be some reason C++ programmers feel a need for

## Re: Basic inheritance question

this "extra, unneeded garbage" ?-)

In some cases, an extremely complex class that can't be fragmented any further, the `m_` convention is surely useful, but in most cases you could skip them out.

You "could" skip this convention, but it's really considered bad practice by quite a lot of C++ programmers.

And the canonical way to make this "clear" is not the `m_` convention, it's the name itself. A well-designed class would choose names that is recognizable instantly from the name itself, even without the pseudo-name appended to it (or prepended).

I await for your examples.

btw you must have been memorizing names braindeadly, because the only way you could stumble on that is by memorizing names braindeadly. Names shouldn't be memorized, it should be inferred and memorized. For example, when you met a variable name `firstname` and `lastname` inside a class called `Person`, you'd immediately realize that it is Class Level variable because you know that the function you're currently working on use the name `initialfirstname` and `initiallastname`.

Fine, I now have four names to handle, each of them being possibly an argument, a local variable, a member variable or a global variable. Great.

Sorry but I won't buy this.

(snip)

As I've pointed out, there is little harm in class-level variable's implicit reference.

Have some working experience on any non-trivial C++ project ?

No

I would have been surprised if you had answer otherwise.

## Re: Basic inheritance question

(you could say I'm a student so I've never "worked"[1]). But I've done some medium-sized projects in other languages.

[1] If you understand the irony, you'd realized I was deliberately misunderstanding you

Not sure.

Compare the following codes:

VB.NET:

Public Class A

Dim var

Public Function aFunction()

return var

Add three levels of inheritance and a couple globals and you'll find out that readability count !-)

It's the mental model that have to be adapted here, if the current class is inheriting from another class, you've got to think it as names from parent class as it is a native names, so you don't actually need to know where the variable comes from

In C++ (and VB IIRC), it might as well be a global So sorry but yes, I have to know where it comes from.

How many times would you use globals, it is a Bad Thing(tm) to use globals in the first case.

It is, most of the time, indeed.

The problem is that you rarely start a project from scratch – most of the time, you have to work on legacy code. And you really seldom have the possibility to do a major rewrite to fix all warts.

In some exceptional cases globals might be unavoidable, but it is trivial to work out that you have to reduce the amount of globals to a minimum, in almost any cases to a number you can use a hand to count with.

That very nice, from a theoretical POV. That's alas just not how it works in real life.

## Re: Basic inheritance question

And applying the hacks mentioned, why don't you use the `m_` convention for globals, and retains the convenience of `m_`-free variables in your class variable. You use class variable much more often than globals, and in most projects class-level variable is used just as often as local-variable.

The problem is not what `*I*` (would) do, but how is the existing code.

since knowing where it  
comes from is breaking the encapsulation

Nope, it's knowing what you're doing and how the piece of software at hand is working. And FWIW, while data hiding is one possible mean of encapsulation, it's by no way a synonym for encapsulation.

I agree that knowing an underlying class's implementation is useful (in fact, very useful) but what I'm talking is about should-ness,

we  
shouldn't `_need_` to know the underlying implementation,

How can you hope to extend a class without `_any_` knowledge of it's implementation ?

(snip)

(which, in Python is very  
weakly implemented, which favors flexibility in many  
cases[1]).

[1] In Python, it is impossible to create a completely private variable, which is the reason why the mental model of these other languages doesn't fit Python.

Never heard about the infamous `'#define private public'` hack in C++ ?  
And don't worry, there are also ways to get at so called 'private' vars in Java.

No, but it's violating the language's rule.

Nope. It's just making use of some part of the language's rules.

## Re: Basic inheritance question

Python OTOH, provides formal ways to get to private vars.

Python doesn't have "private vars" at all.

In any non-trivial piece of C++ code, and unless the author either used the explicit 'this' reference or the 'm\_XXX' naming convention, you'll have hard time figuring out where a given name comes from when browsing a function's code.

If you're used to the implicit naming scheme it's easy to know where a variable came from, if not the current scope, it's the class' scope

You forgot the global scope.

How many global variables do you have in your projects on average? If you always have a pageful list of globals in your projects, then you should consider unlearning and relearning the basic programming concepts.

I'll come to you when I'll need more training, don't worry.

It's easy to keep track of globals, as you shouldn't have a lot of them even in a huge project.

Can't you understand that starting a new project afresh is *\*not\** the common case ?

As a final note:  
I don't think implicit class reference is superior to explicit class reference, neither

...

I'm sure you don't believe it since I'm talking on implicit's side, but that's the fact, I just pointed you out that implicits do have its positive side (even if you don't consider them positive in your book) but that doesn't mean I believe it is better than the other.

## Re: Basic inheritance question

To clear things up:

As a final note:

I don't think implicit class reference is superior to explicit class reference, but I don't think the vice versa is true either.

Once again : what classes have to do with this ?

Seriously, how can you hope to be taken seriously when you're obviously confusing such important concepts as instance and class ?

.