Python: Inherit the superclass __init__

Asked 9 years, 4 months ago Active 7 months ago Viewed 64k times

69

14

I have a base class with a lot of `__init__` arguments:

```
class BaseClass(object):
    def __init__(self, a, b, c, d, e, f, ...):
        self._a=a+b
        self._b=b if b else a
        ...
```

All the inheriting classes should run `__init__` method of the base class.

I can write a `__init__()` method in each of the inheriting classes that would call the superclass `__init__`, but that would be a serious code duplication:

```
class A(BaseClass):
    def __init__(self, a, b, c, d, e, f, ...):
        super(A, self).__init__(a, b, c, d, e, f, ...)

class B(BaseClass):
    def __init__(self, a, b, c, d, e, f, ...):
        super(A, self).__init__(a, b, c, d, e, f, ...)

class C(BaseClass):
    def __init__(self, a, b, c, d, e, f, ...):
        super(A, self).__init__(a, b, c, d, e, f, ...)

...
```

What's the most Pythonic way to automatically call the superclass `__init__`?

python inheritance constructor init Edit tags

edited Apr 5 at 6:17
James Carter
651 1 9 18

asked Jun 30 '11 at 13:51
Adam Matan
100k 114 323 492

7 Answers

Active Oldest Votes

55

```
super(SubClass, self).__init__(...)
```

Consider using *args and **kw if it helps solving your variable nightmare.

edited Jun 26 '13 at 14:25
felippo
5 2

answered Jun 30 '11 at 13:55
Andreas Jung
1

It helps, but is there a way to completely avoid explicitly calling the superclass **init**? – Adam Matan Jun 30 '11 at 13:58

4 @Adam Matan No, there isn't. See here why: Why aren't Python's superclass **init** methods automatically invoked? – Paolo Moretti Jun 30 '11 at 14:01

@Cat Plus Plus correct, but it still requires the annoying code duplication for every base class. – Adam Matan Jun 30 '11 at 14:02

1 @Dogeatcatworld yes. If you called `BaseClass.__init__` inside of `BaseClass(): __init__:` you would get infinite recursion. `super` calls the parent class. – Nick Humrich Jul 22 '14 at 15:34

2 @CatPlusPlus Sorry, I have a very stupid question. In Python 3, how can I use the ellipsis to feed the __init__()? I kept getting errors. When you guys used "...", did you really mean ellipsis in the code or it was just meant to save typing? – astroboylrx Feb 13 '17 at 20:03

|

12

Perhaps a clearer implementation for your case is using **kwargs **combined with new added arguments** in your derived class as in:

```
class Parent:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c


class Child(Parent):
    def __init__(self, d, **kwargs):
        super(Child, self).__init__(**kwargs)
        self.d = d
```

By this method you avoid the code duplication but preserve the implicit addition of arguments in your derived class.

answered Mar 31 '18 at 12:13
avielbl
163 1 7

1 all solution should looks like this one. – Kai Wang Nov 16 '18 at 20:46

1 Running `Child(1, 2, 3, 4)` I get `TypeError: __init__() takes 2 positional arguments but 5 were given` . I suspect this should be `*args` instead of `**kwargs` – JavNoor Aug 20 '19 at 19:33

by using args, you might mix between your arguments when calling to your child classes. This is why it is better practice in such situation to use **kw and named arguments – avielbl Aug 22 '19 at 4:41

2

In 2.6 and lower version to inherit init from base class, there is no super function, You can inherit below way:

```
class NewClass():
     def __init__():
         BaseClass.__init__(self, *args)
```

answered Jan 12 '17 at 11:58

Aashutosh jha
366 2 6

0

Adding a Pythonic implementation. Assuming you want all attributes passed in, you can use the code below. (Can also keep/remove specific kwargs keys if you want a subset).

```
def A(BaseClass):
    def __init__(self, *args, **kwargs):
        for key, value in kwargs.items():
            setattr(self, key, value)

base = BaseClass(...)
new = A( **base.__dict__ )
```

edited Jan 29 '16 at 16:24

Adam Matan
100k 114 323 492

answered Jan 29 '16 at 2:44

Ben
351 1 3 14

38

You have to write it explicitly, but on the other hand, if you have lots of args, you should probably use *args for positional args and **kwargs for keyword args.

```
class SubClass(BaseClass):
    def __init__(self, *args, **kwargs):
        super(SubClass, self).__init__(*args, **kwargs)
        # SubClass initialization code
```

Another technique you could use is to minimize the code in **init** and then at the end of **init** function, call another custom function. Then in the subclass, you just need to override the custom function

```
class BaseClass(object):
    def __init__(self, *args, **kwargs):
        # initialization code
        self._a = kwargs.get('a')
        ...
        # custom code for subclass to override
        self.load()

    def load():
        pass

class SubClass(BaseClass)
    def load():
        # SubClass initialization code
        ...
```

edited Jun 26 '13 at 14:28

felippo
5 2

answered Jun 30 '11 at 14:01

Ryan Ye
2,489 16 23

---

1 This is a technique I've used, in some cases going so far as to provide `preinit()` and `postinit()` hooks. The return value from `preinit()` can be saved in a local variable in `__init__` and passed to `postinit()`, which is convenient sometimes. – kindall Sep 21 '11 at 22:29

@kindall I want to write a superclass that does some modest, uniform member checking of subclass instances (viz., that needed member variables exist and have sane contents), as part of their `__init__` process. I think this sort of `postinit()` hook should work perfectly. – hBy2Py Sep 26 '15 at 16:40

1 improvement: kwargs.get('a', defaultvaluehere) instead of kwargs.get('a') – Massimo Variolo Dec 2 '15 at 10:28

19

If the derived classes don't implement anything beyond what the base class `__init__()` already does, just omit the derived classes `__init__()` methods - the base class `__init__()` is then called automatically.

If, OTOH, your derived classes add some extra work in their `__init__()`, and you don't want them to explicitly call their base class `__init__()`, you can do this:

```
class BaseClass(object):
    def __new__(cls, a, b, c, d, e, f, ...):
        new = object.__new__(cls)
        new._a=a+b
        new._b=b if b else a
        ...
        return new

class A(BaseClass):
    ''' no __init__() at all here '''

class B(BaseClass):
    def __init__(self, a, b, c, d, e, f, ...):
        ''' do stuff with init params specific to B objects '''
```

Since `__new__()` is always called automatically, no further work is required in the derived classes.

edited Sep 21 '11 at 21:45

answered Jun 30 '11 at 14:12

pillmuncher
9,278 2 31 31

Fine method, usefull to know. +1. - By the way, as in the question, the code needs to be corrected: *class BaseClass(object)* instead of *def BaseClass(object)* – eyquem Sep 21 '11 at 12:43

14

Unless you are doing something useful in the subclass `__init__()` methods, you don't have to override it.

```
def BaseClass(object):
    def __init__(self, a, b, c, d, e, f, ...):
        self._a=a+b
        self._b=b if b else a
        ...

def A(BaseClass):
    def some_other_method(self):
        pass

def B(BaseClass):
    pass
```

answered Jun 30 '11 at 14:04

Rob Cowie

20.5k 6 57 56