Inheritance and Overriding __init__ in python

Asked 11 years, 7 months ago Active 11 months ago Viewed 181k times

131

33

I was reading 'Dive Into Python' and in the chapter on classes it gives this example:

```
class FileInfo(UserDict):
    "store file metadata"
    def __init__(self, filename=None):
        UserDict.__init__(self)
        self["name"] = filename
```

The author then says that if you want to override the `__init__` method, you must explicitly call the parent `__init__` with the correct parameters.

1. What if that `FileInfo` class had more than one ancestor class?
    - Do I have to explicitly call all of the ancestor classes' `__init__` methods?
2. Also, do I have to do this to any other method I want to override?

python overriding superclass Edit tags

edited May 5 '14 at 16:08
Eugene S
5,978 7 49 79

asked Apr 15 '09 at 20:45

liewl
3,769 12 41 60

3 Note that Overloading is a separate concept from Overriding. – Dana the Sane Apr 15 '09 at 21:02

7 Answers

Active | Oldest | Votes

159

The book is a bit dated with respect to subclass-superclass calling. It's also a little dated with respect to subclassing built-in classes.

It looks like this nowadays:

```
class FileInfo(dict):
    """store file metadata"""
    def __init__(self, filename=None):
        super(FileInfo, self).__init__()
        self["name"] = filename
```

Note the following:

1. We can directly subclass built-in classes, like `dict`, `list`, `tuple`, etc.

2. The `super` function handles tracking down this class's superclasses and calling functions in them appropriately.

edited Nov 26 '19 at 19:14

Andrew Palmer
1,170 11 13

answered Apr 15 '09 at 20:49

S.Lott
351k 73 478 750

5 should i look for a better book/tutorial? – liewl Apr 15 '09 at 20:54

2 So in the case of multiple inheritance, does super() track them all down on your behalf? – Dana Apr 15 '09 at 20:54

dict.__init__(self), actually, but nothing's wrong with it - the super(...) call just provides a more consistent syntax. (I'm not sure how it works for multiple inheritance, I think it may only find one superclass **init**) – David Z Apr 15 '09 at 21:47

4 The intention of super() is that it handles multiple inheritance. The disadvantage is, that in practice multiple inheritance still breaks very easily (see <fuhm.net/super-harmful). – sth Apr 15 '09 at 22:06

2 Yes, in case of multiple inheritance and base classes taking constructor arguments, you usually find yourself calling the constructors manually. – Torsten Marek Apr 15 '09 at 22:53

18

In each class that you need to inherit from, you can run a loop of each class that needs init'd upon initiation of the child class...an example that can copied might be better understood...

```
class Female_Grandparent:
    def __init__(self):
        self.grandma_name = 'Grandma'

class Male_Grandparent:
    def __init__(self):
        self.grandpa_name = 'Grandpa'

class Parent(Female_Grandparent, Male_Grandparent):
    def __init__(self):
        Female_Grandparent.__init__(self)
        Male_Grandparent.__init__(self)

        self.parent_name = 'Parent Class'

class Child(Parent):
    def __init__(self):
        Parent.__init__(self)
#--------------------------------------------------------------------------------
-#
        for cls in Parent.__bases__: # This block grabs the classes of the child
             cls.__init__(self)      # class (which is named 'Parent' in this case),
                                     # and iterates through them, initiating each one.
                                     # The result is that each parent, of each child,
                                     # is automatically handled upon initiation of the
                                     # dependent class. WOOT WOOT! :D
#--------------------------------------------------------------------------------
-#



g = Female_Grandparent()
print g.grandma_name
```

```
p = Parent()
print p.grandma_name

child = Child()

print child.grandma_name
```

edited Apr 30 '14 at 22:20

Adam
3,236 5 41 73

answered Jun 15 '12 at 17:43

Code Bug
225 2 4

2 It doesn't seem like the for loop in `Child.__init__` is necessary. When I remove it from the example I child still prints "Grandma". Isn't grandparent init handled by the `Parent` class? – Adam Apr 30 '14 at 22:17

4 I think granparents init's are already handled by Parent's init, aren't they? – johk95 Dec 14 '14 at 16:33

0

This post is hidden. It was deleted 11 years ago by the post author.

more a question than an answer:

```
class A:
    def __init__(self):
        some_code()...
        self.subclass_init(self)

class B(A):
    def subclass_init(self):
        additional_code()...
```

is it "right" (both from python and OOP points of view) ?

answered May 3 '09 at 11:55

Berry Tsakala
10.4k 9 49 67

you might want to ask a separate question and delete this non-answer – SilentGhost May 3 '09 at 12:03

comments disabled on deleted / locked posts / reviews

0

This post is hidden. It was deleted 11 years ago by the post author.

Ok, correct me if I'm wrong, but I've always been under the impression that if you wrote an __init__ method for a a sub class then you would only have to call it's immediate superclass' __init__ , because somewhere in that method would be the call to the __init__ of it's parent superclass, and so on all the way up the hierarchy.

In other words, you'd only have to call it once, because all further calls would be automatic.

answered May 3 '09 at 15:56

dangerouslyfacetious
503 3 11

comments disabled on deleted / locked posts / reviews

15

You don't really *have* to call the `__init__` methods of the base class(es), but you usually *want* to do it because the base classes will do some important initializations there that are needed for rest of the classes methods to work.

For other methods it depends on your intentions. If you just want to add something to the base classes behavior you will want to call the base classes method additionally to your own code. If you want to fundamentally change the behavior, you might not call the base class' method and implement all the functionality directly in the derived class.

answered Apr 15 '09 at 21:00

sth
192k 49 259 350

4 For technical completeness, some classes, like threading.Thread, will throw gigantic errors if you ever try to avoid calling the parent's **init**. – David Berger Apr 15 '09 at 21:39

5 I find all this "you don't have to call the base's constructor" talk extremely irritating. You don't have to call it in any language I know. All of them will screw up (or not) in much the same way, by not initializing members. Suggesting not to initialize base classes is just wrong in so many ways. If a class does not need initialization now it will need it in the future. The constructor is part of the interface of the class structure/language construct,and should be used correctly. It's correct use is to call it sometime in your derived's constructor. So do it. – AndreasT Jun 5 '13 at 21:29

2 "Suggesting not to initialize base classes is just wrong in so many ways." No one has suggested to not initialize the base class. Read the answer carefully. It's all about intention. 1) If you want to leave the base class' init logic as is, you don't override init method in your derived class. 2) If you want to extend the init logic from the base class, you define your own init method and then call base class' init method from it. 3) If you want to replace base class' init logic, you define your own init method while not calling the one from the base class. – wombatonfire Jul 9 '18 at 18:16

2

Yes, you must call `__init__` for each parent class. The same goes for functions, if you are overriding a function that exists in both parents.

answered Apr 15 '09 at 20:50

vezult
4,865 20 40

4

If the FileInfo class has more than one ancestor class then you should definitely call all of their __init__() functions. You should also do the same for the __del__() function, which is a destructor.

answered Apr 15 '09 at 20:49

moinudin
112k 41 182 213