## Inheritance and Overriding \_\_init\_\_ in python

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



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

131



X

33

def \_\_init\_\_(self, filename=None):
 UserDict.\_\_init\_\_(self)
 self["name"] = filename

class FileInfo(UserDict):
 "store file metadata"

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



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

## 7 Answers





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.

159



It looks like this nowadays:



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

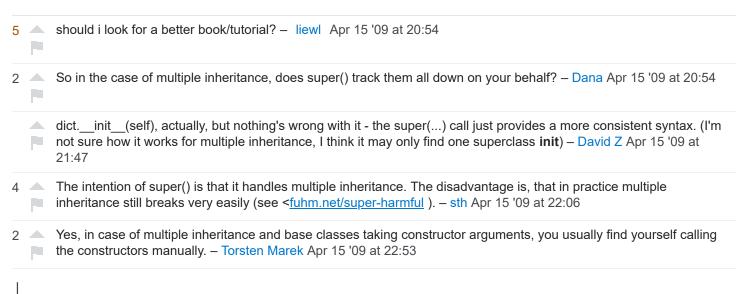
Note the following:

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.







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...

```
18
```



43

```
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



answered Jun 15 '12 at 17:43

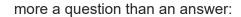


- 2 Lt doesn't seem like the for loop in Child.\_\_init\_\_ is necessary. When I remove it from the example I child still prints "Grandma". Isn't grandparent init handled by the Parent class? Adam Apr 30 '14 at 22:17
- I think granparents init's are already handled by Parent's init, aren't they? johk95 Dec 14 '14 at 16:33



**11** This post is hidden. It was <u>deleted</u> 11 years ago by the post author.

0



**(**)

```
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



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



**7** This post is hidden. It was <u>deleted</u> 11 years ago by the post author.

1

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.

comments disabled on deleted / locked posts / reviews



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



- 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 A 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
- "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



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.

2



answered Apr 15 '09 at 20:50



**4.865** 20



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

182 213

