How to implement virtual methods in Python?

Asked 12 years ago Modified 3 months ago Viewed 131k times

113

I know virtual methods from PHP or Java.

How can they be implemented in Python?

Or have I to define an empty method in an abstract class and override it?

python virtual-functions

Share Improve this question Follow

edited Aug 17, 2017 at 11:24
Melebius
5,967 4 35 50

asked Jan 17, 2011 at 14:14
Meloun
13.1k 17 63 92

7 Answers

Sorted by:
Highest score (default)

127

Sure, and you don't even have to define a method in the base class. In Python methods are better than virtual - they're completely dynamic, as the typing in Python is *duck typing*.

```
class Dog:
  def say(self):
    print "hau"

class Cat:
  def say(self):
    print "meow"

pet = Dog()
pet.say() # prints "hau"
another_pet = Cat()
another_pet.say() # prints "meow"

my_pets = [pet, another_pet]
for a_pet in my_pets:
  a_pet.say()
```

`Cat` and `Dog` in Python don't even have to derive from a common base class to allow this behavior - you gain it for free. That said, some programmers prefer to define their class hierarchies in a more rigid way to document it better and impose *some* strictness of typing. This is also possible - see for example the abc standard module.

Share Improve this answer Follow

edited Oct 5, 2017 at 22:46
Engineero
12k 5 52 74

answered Jan 17, 2011 at 14:19
Eli Bendersky
258k 88 347 410

52 +1 for an example. In what language do dogs say "hau" by the way? – JeremyP Jan 17, 2011 at 15:05

7 @JeremyP: hmm, good point :-) I guess in languages where "h" is understood as making the sound like the first letter of "hippy", or of "Javier" in Spanish. – Eli Bendersky Jan 17, 2011 at 15:07

7 @Eli: Sorry, but I was seriously interested in the answer to the question. In English they say "woof", well they don't but that is the word we use analogous to "meow" for cats and "moo" for cows. Is "hau" Spanish then? – JeremyP Jan 17, 2011 at 15:11

14 @JeremyP I know for sure that's what Polish dogs say ;) – j_kubik Nov 21, 2016 at 20:25

3 @JeremyP yes I confirm dogs say "Jau" in Spanish and when written in english would be "Hau" :) hth – SkyWalker Jun 9, 2020 at 12:37

102

`raise NotImplementedError()`

This is the recommended exception to raise on "pure virtual methods" of "abstract" base classes that don't implement a method.

https://docs.python.org/3.5/library/exceptions.html#NotImplementedError says:

> This exception is derived from `RuntimeError`. In user defined base classes, abstract methods should raise this exception when they require derived classes to override the method.

As others said, this is mostly a documentation convention and is not required, but this way you get a more meaningful exception than a missing attribute error.

E.g.:

```
class Base(object):
    def virtualMethod(self):
        raise NotImplementedError()
    def usesVirtualMethod(self):
        return self.virtualMethod() + 1

class Derived(Base):
    def virtualMethod(self):
        return 1

print Derived().usesVirtualMethod()
Base().usesVirtualMethod()
```

gives:

```
2
Traceback (most recent call last):
  File "./a.py", line 13, in <module>
    Base().usesVirtualMethod()
  File "./a.py", line 6, in usesVirtualMethod
    return self.virtualMethod() + 1
  File "./a.py", line 4, in virtualMethod
    raise NotImplementedError()
NotImplementedError
```

Related: Is it possible to make abstract classes in Python?

Share Improve this answer Follow

edited Jan 15, 2019 at 11:02

answered Aug 2, 2016 at 10:07

Ciro Santilli
OurBigBook.com
329k 95 1153 945

58

Python methods are always virtual.

Share Improve this answer Follow

answered Jan 17, 2011 at 14:16

Ignacio Vazquez-Abrams
763k 151 1326 1346

1 Except for dunder methods. – Konstantin Dec 29, 2018 at 10:16

3 This answer is not really helping in the objective of implementing interface classes which is one of the main reason for using virtual methods. – Jean-Marc Volle Jul 10, 2020 at 8:38

25

Actually, in version 2.6 python provides something called **abstract base classes** and you can explicitly set virtual methods like this:

```
from abc import ABCMeta
from abc import abstractmethod
...
class C:
    __metaclass__ = ABCMeta
    @abstractmethod
    def my_abstract_method(self, ...):
```

It works very well, provided the class does not inherit from classes that already use metaclasses.

source: http://docs.python.org/2/library/abc.html

Share Improve this answer Follow

edited May 27, 2018 at 13:04

Brian Burns
19.5k 8 82 73

answered Oct 11, 2013 at 10:36

user2795020
245 3 3

Is there a python 3 equivalent for this directive? – locke14 Jun 14, 2018 at 12:03

11

> Python methods are always virtual

like Ignacio said yet Somehow class inheritance may be a better approach to implement what you want.

```
class Animal:
    def __init__(self,name,legs):
        self.name = name
        self.legs = legs

    def getLegs(self):
        return "{0} has {1} legs".format(self.name, self.legs)

    def says(self):
        return "I am an unknown animal"

class Dog(Animal): # <Dog inherits from Animal here (all methods as well)

    def says(self): # <Called instead of Animal says method
        return "I am a dog named {0}".format(self.name)
```

```
    def somethingOnlyADogCanDo(self):
        return "be loyal"

formless = Animal("Animal", 0)
rover = Dog("Rover", 4) #<calls initialization method from animal

print(formless.says()) # <calls animal say method

print(rover.says()) #<calls Dog says method
print(rover.getLegs()) #<calls getLegs method from animal class
```

Results should be:

```
I am an unknown animal
I am a dog named Rover
Rover has 4 legs
```

Share Improve this answer Follow

answered Aug 22, 2015 at 0:44

Jinzo

111 1 2

9

Something like a virtual method in C++ (calling method implementation of a derived class through a reference or pointer to the base class) doesn't make sense in Python, as Python doesn't have typing. (I don't know how virtual methods work in Java and PHP though.)

But if by "virtual" you mean calling the bottom-most implementation in the inheritance hierarchy, then that's what you always get in Python, as several answers point out.

Well, almost always...

As dplamp pointed out, not all methods in Python behave like that. Dunder method don't. And I think that's a not so well known feature.

Consider this artificial example

```
class A:
    def prop_a(self):
        return 1
    def prop_b(self):
        return 10 * self.prop_a()

class B(A):
    def prop_a(self):
        return 2
```

Now

```
>>> B().prop_b()
20
>>> A().prob_b()
10
```

However, consider this one

```
class A:
    def __prop_a(self):
        return 1
    def prop_b(self):
        return 10 * self.__prop_a()

class B(A):
    def __prop_a(self):
        return 2
```

Now

```
>>> B().prop_b()
10
>>> A().prob_b()
10
```

The only thing we've changes was making `prop_a()` a dunder method.

A problem with the first behavior can be that you can't change the behavior of `prop_a()` in the derived class without impacting the behavior of `prop_b()`. This very nice talk by Raymond Hettinger gives an example for a use case where this is inconvenient.

Share Improve this answer Follow

answered Dec 29, 2018 at 10:16

Konstantin
2,271 1 24 26

0

Python 3.6 introduced `__init_subclass__` and this let you simply do this:

```
class A:

    def method(self):
        '''method needs to be overwritten'''
        return NotImplemented

    def __init_subclass__(cls):
        if cls.method is A.method:
            raise NotImplementedError(
                'Subclass has not overwritten method {method}!')
```

The benefit of this solution is that you avoid the abc metaclass and give the user a direct imperative how to do it right. In addition to another answer here that raises `NotImplementedError` when calling the method. This solution is checked on runtime and not only IF the user calls the method.

Share Improve this answer Follow

answered Oct 14, 2022 at 15:49

Thingamabobs
6,294 5 16 46