How can I simply inherit methods from an existing instance?

Asked 16 years, 3 months ago Modified 4 years, 1 month ago Viewed 545 times

1

Below I have a very simple example of what I'm trying to do. I want to be able to use HTMLDecorator with any other class. Ignore the fact it's called decorator, it's just a name.

```
import cgi

class ClassX(object):
  pass # ... with own __repr__

class ClassY(object):
  pass # ... with own __repr__

inst_x=ClassX()

inst_y=ClassY()

inst_z=[ i*i for i in range(25) ]

inst_b=True

class HTMLDecorator(object):
   def html(self): # an "enhanced" version of __repr__
       return cgi.escape(self.__repr__()).join(("<H1>","</H1>"))

print HTMLDecorator(inst_x).html()
print HTMLDecorator(inst_y).html()
wrapped_z = HTMLDecorator(inst_z)
inst_z[0] += 70
wrapped_z[0] += 71
print wrapped_z.html()
print HTMLDecorator(inst_b).html()
```

Output:

```
Traceback (most recent call last):
  File "html.py", line 21, in
    print HTMLDecorator(inst_x).html()
TypeError: default __new__ takes no parameters
```

Is what I'm trying to do possible? If so, what am I doing wrong?

python oop inheritance object

Share

Improve this question

Follow

edited Aug 9, 2017 at 20:13

Peter Featherstone

8,092 ● 6 ● 36 ● 67

asked Sep 1, 2008 at 5:17

Harley Holcombe

182k ● 15 ● 72 ● 63

6 Answers

Sorted by: Highest score (default)

2

> Very close, but then I lose everything from ClassX. Below is something a collegue gave me that does do the trick, but it's hideous. There has to be a better way.

Looks like you're trying to set up some sort of proxy object scheme. That's doable, and there are better solutions than your colleague's, but first consider whether it would be easier to just patch in some extra methods. This won't work for built-in classes like `bool`, but it will for your user-defined classes:

```
def HTMLDecorator (obj):
    def html ():
        sep = cgi.escape (repr (obj))
        return sep.join (("<H1>", "</H1>"))
    obj.html = html
    return obj
```

And here is the proxy version:

```
class HTMLDecorator(object):
    def __init__ (self, wrapped):
        self.__wrapped = wrapped

    def html (self):
        sep = cgi.escape (repr (self.__wrapped))
        return sep.join (("<H1>", "</H1>"))

    def __getattr__ (self, name):
        return getattr (self.__wrapped, name)

    def __setattr__ (self, name, value):
        if not name.startswith ('_HTMLDecorator__'):
            setattr (self.__wrapped, name, value)
            return
        super (HTMLDecorator, self).__setattr__ (name, value)

    def __delattr__ (self, name):
        delattr (self.__wraped, name)
```

Share Improve this answer Follow

answered Sep 1, 2008 at 7:33

John Millikin

201k ● 41 ● 215 ● 227

2

Both of John's solutions would work. Another option that allows HTMLDecorator to remain very simple and clean is to monkey-patch it in as a base class. This also works only for user-defined classes, not builtin types:

```
import cgi

class ClassX(object):
    pass # ... with own __repr__

class ClassY(object):
    pass # ... with own __repr__

inst_x=ClassX()
inst_y=ClassY()

class HTMLDecorator:
    def html(self): # an "enhanced" version of __repr__
        return cgi.escape(self.__repr__()).join(("<H1>","</H1>"))

ClassX.__bases__ += (HTMLDecorator,)
ClassY.__bases__ += (HTMLDecorator,)

print inst_x.html()
print inst_y.html()
```

Be warned, though -- monkey-patching like this comes with a high price in readability and maintainability of your code. When you go back to this code a year later, it can become very difficult to figure out how your ClassX got that html() method, especially if ClassX is defined in some other library.

Share

Improve this answer

Follow

edited Sep 1, 2008 at 8:36

answered Sep 1, 2008 at 8:30

Carl Meyer

126k ● 20 ● 109 ● 117

0

> Is what I'm trying to do possible? If so, what am I doing wrong?

It's certainly possible. What's wrong is that `HTMLDecorator.__init__()` doesn't accept parameters.

Here's a simple example:

```
def decorator (func):
    def new_func ():
        return "new_func %s" % func ()
    return new_func

@decorator
def a ():
    return "a"

def b ():
    return "b"
```

```
print a() # new_func a
print decorator (b)() # new_func b
```

Share Improve this answer Follow

answered Sep 1, 2008 at 5:26

John Millikin
201k ● 41 ● 215 ● 227

0

@John (37448):

Sorry, I might have misled you with the name (bad choice). I'm not really looking for a decorator function, or anything to do with decorators at all. What I'm after is for the html(self) def to use ClassX or ClassY's `__repr__`. I want this to work without modifying ClassX or ClassY.

Share

Improve this answer

Follow

edited Sep 1, 2008 at 6:17

answered Sep 1, 2008 at 6:10

Harley Holcombe
182k ● 15 ● 72 ● 63

0

Ah, in that case, perhaps code like this will be useful? It doesn't really have anything to do with decorators, but demonstrates how to pass arguments to a class's initialization function and to retrieve those arguments for later.

```
import cgi

class ClassX(object):
    def __repr__ (self):
        return "<class X>"

class HTMLDecorator(object):
    def __init__ (self, wrapped):
        self.__wrapped = wrapped

    def html (self):
        sep = cgi.escape (repr (self.__wrapped))
        return sep.join (("<H1>", "</H1>"))

inst_x=ClassX()
inst_b=True

print HTMLDecorator(inst_x).html()
print HTMLDecorator(inst_b).html()
```

Share Improve this answer Follow

answered Sep 1, 2008 at 6:25

John Millikin
201k ● 41 ● 215 ● 227

0

@John (37479):

Very close, but then I lose everything from ClassX. Below is something a collegue gave me that does do the trick, but it's hideous. There has to be a better way.

```
import cgi
from math import sqrt

class ClassX(object):
  def __repr__(self):
    return "Best Guess"

class ClassY(object):
  pass # ... with own __repr__

inst_x=ClassX()

inst_y=ClassY()

inst_z=[ i*i for i in range(25) ]

inst_b=True

avoid="__class__ __init__ __dict__ __weakref__"

class HTMLDecorator(object):
    def __init__(self,master):
        self.master = master
        for attr in dir(self.master):
            if ( not attr.startswith("__") or
                attr not in avoid.split() and "attr" not in attr):
                self.__setattr__(attr, self.master.__getattribute__(attr))

    def html(self): # an "enhanced" version of __repr__
        return cgi.escape(self.__repr__()).join(("<H1>","</H1>"))

    def length(self):
        return sqrt(sum(self.__iter__()))

print HTMLDecorator(inst_x).html()
print HTMLDecorator(inst_y).html()
wrapped_z = HTMLDecorator(inst_z)
print wrapped_z.length()
inst_z[0] += 70
#wrapped_z[0] += 71
wrapped_z.__setitem__(0,wrapped_z.__getitem__(0)+ 71)
print wrapped_z.html()
print HTMLDecorator(inst_b).html()
```

Output:

```
<H1>Best Guess</H1>
<H1><__main__.ClassY object at 0x891df0c></H1>
70.0
<H1>[141, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225,
```

```
256, 289, 324, 361, 400, 441, 484, 529, 576]</H1>
<H1>True</H1>
```

Share Improve this answer Follow

answered Sep 1, 2008 at 6:55

Harley Holcombe
182k ● 15 ● 72 ● 63