Property Objects

Python 2.x 2.3–2.7 , Python 3.x 3.0–3.6

Examples

Using the @property decorator for read-write properties

If you want to use @property to implement custom behavior for setting and getting, use this pattern:

```
class Cash(object):
    def __init__(self, value):
        self.value = value
    @property
    def formatted(self):
        return '${:.2f}'.format(self.value)
    @formatted.setter
    def formatted(self, new):
        self.value = float(new[1:])
```

To use this:

```
>>> wallet = Cash(2.50)
>>> print(wallet.formatted)
$2.50
>>> print(wallet.value)
2.5
>>> wallet.formatted = '$123.45'
>>> print(wallet.formatted)
$123.45
>>> print(wallet.value)
123.45
```

Using the @property decorator

The @property decorator can be used to define methods in a class which act like attributes. One example where this can be useful is when exposing information which may require an initial (expensive) lookup and simple retrieval thereafter.

Given some module foobar.py :

```
class Foo(object):
    def __init__(self):
        self.__bar = None

    @property
    def bar(self):
        if self.__bar is None:
            self.__bar = some_expensive_lookup_operation()
        return self.__bar
```

Then

```
>>> from foobar import Foo
>>> foo = Foo()
>>> print(foo.bar)  # This will take some time since bar is None after initialization
42
>>> print(foo.bar)  # This is much faster since bar has a value now
42
```

Overriding just a getter, setter or a deleter of a property object

When you inherit from a class with a property, you can provide a new implementation for one or more of the property getter , setter or deleter functions, by referencing the property object *on the parent class* :

```
class BaseClass(object):
    @property
    def foo(self):
        return some_calculated_value()

    @foo.setter
    def foo(self, value):
        do_something_with_value(value)
```

```
class DerivedClass(BaseClass):
    @BaseClass.foo.setter
    def foo(self, value):
        do_something_different_with_value(value)
```

You can also add a setter or deleter where there was not one on the base class before.

Using properties without decorators

While using decorator syntax (with the @) is convenient, it also a bit concealing. You can use properties directly, without decorators. The following Python 3.x example shows this:

```
class A:
    p = 1234
    def getX (self):
        return self._x

    def setX (self, value):
        self._x = value

    def getY (self):
        return self._y

    def setY (self, value):
        self._y = 1000 + value    # Weird but possible

    def getY2 (self):
        return self._y

    def setY2 (self, value):
        self._y = value

    def getT    (self):
        return self._t

    def setT (self, value):
        self._t = value

    def getU (self):
        return self._u + 10000

    def setU (self, value):
        self._u = value - 5000

    x, y, y2 = property (getX, setX), property (getY, setY), property (getY2, setY2)
    t = property (getT, setT)
    u = property (getU, setU)

A.q = 5678
```

Syntax

Parameters

Remarks

Note : In Python 2, make sure that your class inherits from object (making it a new-style class) in order for all features of properties to be available.