Descriptor All Versions

Examples

Simple descriptor

There are two different types of descriptors. Data descriptors are defined as objects that define both a __get__() and a __set__() method, whereas non-data descriptors only define a __get__() method. This distinction is important when considering overrides and the namespace of an instance's dictionary. If a data descriptor and an entry in an instance's dictionary share the same name, the data descriptor will take precedence. However, if instead a non-data descriptor and an entry in an instance's dictionary share the same name, the instance dictionary's entry will take precedence.

To make a read-only data descriptor, define both **get** () and **set** () with the **set** () raising an AttributeError when called. Defining the **set** () method with an exception raising placeholder is enough to make it a data descriptor.

```
descr.__get__(self, obj, type=None) --> value
descr.__set__(self, obj, value) --> None
descr.__delete__(self, obj) --> None
```

An implemented example:

```
class DescPrinter(object):
    """A data descriptor that logs activity."""
    _val = 7

    def __get__(self, obj, objtype=None):
        print('Getting ...')
        return self._val

    def __set__(self, obj, val):
        print('Setting', val)
        self._val = val

    def __delete__(self, obj):
        print('Deleting ...')
        del self._val


class Foo():
    x = DescPrinter()

i = Foo()
i.x
# Getting ...
# 7

i.x = 100
# Setting 100
i.x
# Getting ...
# 100

del i.x
# Deleting ...
i.x
# Getting ...
# 7
```

Two-way conversions

Descriptor objects can allow related object attributes to react to changes automatically.

Suppose we want to model an oscillator with a given frequency (in Hertz) and period (in seconds). When we update the frequency we want the period to update, and when we update the period we want the frequency to update:

```
 >>> oscillator = Oscillator(freq=100.0)  # Set frequency to 100.0 Hz
>>> oscillator.period  # Period is 1 / frequency, i.e. 0.01 seconds
0.01
>>> oscillator.period = 0.02  # Set period to 0.02 seconds
>>> oscillator.freq # The frequency is automatically adjusted
50.0
>>> oscillator.freq = 200.0  # Set the frequency to 200.0 Hz
>>> oscillator.period  # The period is automatically adjusted
0.005
```

We pick one of the values (frequency, in Hertz) as the "anchor", i.e. the one that can be set with no

We pick one of the values (frequency, in Hertz) as the "anchor", i.e. the one that can be set with no conversion, and write a descriptor class for it:

```
class Hertz(object):
    def __get__(self, instance, owner):
        return self.value

    def __set__(self, instance, value):
        self.value = float(value)
```

The "other" value (period, in seconds) is defined in terms of the anchor. We write a descriptor class that does our conversions:

```
class Second(object):
    def __get__(self, instance, owner):
        # When reading period, convert from frequency
        return 1 / instance.freq

    def __set__(self, instance, value):
        # When setting period, update the frequency
        instance.freq = 1 / float(value)
```

Now we can write the Oscillator class:

```
class Oscillator(object):
    period = Second()  # Set the other value as a class attribute

    def __init__(self, freq):
        self.freq = Hertz()  # Set the anchor value as an instance attribute
        self.freq = freq  # Assign the passed value - self.period will be adjusted
```

Syntax

Parameters

Remarks