Descriptor Examples

At this point, you may be confused how you would even use a descriptor. I always find it helpful when I am learning a new concept if I have a few examples that demonstrate how it works. So in this section, we will look at some examples so you will know how to use descriptors in your own code!

Let's start by writing a really simple data descriptor and then use it in a class. This example is based on one from Python's documentation:

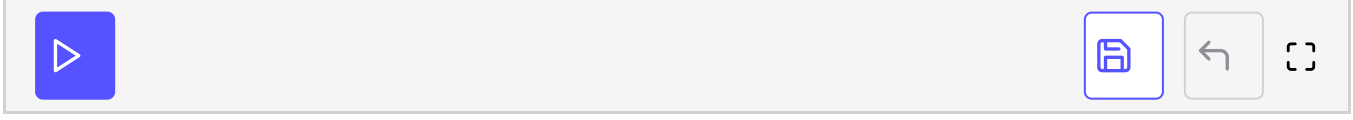
```
class MyDescriptor():
    """
    A simple demo descriptor
    """
    def __init__(self, initial_value=None, name='my_var'):
        self.var_name = name
        self.value = initial_value

    def __get__(self, obj, objtype):
        print('Getting', self.var_name)
        return self.value

    def __set__(self, obj, value):
        msg = 'Setting {name} to {value}'
        print(msg.format(name=self.var_name, value=value))
        self.value = value

class MyClass():
    desc = MyDescriptor(initial_value='Mike', name='desc')
    normal = 10

if __name__ == '__main__':
    c = MyClass()
    print(c.desc)
    print(c.normal)
    c.desc = 100
    print(c.desc)
```



Here we create a class and define three magic methods:

- __init__ - our constructor which takes a value and the name of our variable
- __get__ - prints out the current variable name and returns the value
- __set__ - prints out the name of our variable and the value we just assigned and sets the value itself

Then we create a class that creates an instance of our descriptor as a class attribute and also creates a normal class attribute. Then we run a few “tests” by creating an instance of our normal class and accessing our class attributes. Here is the output:

```
Getting desc
Mike
10
Setting desc to 100
Getting desc
100
```

As you can see, when we access **c.desc**, it prints out our "Getting" message and we print out what it returns, which is “Mike”. Next we print out the regular class attribute’s value. Finally we change the descriptor variable’s value, which causes our “Setting” message to be printed. We also double-check the current value to make sure that it was actually set, which is why you see that last “Getting” message.

Python uses descriptors underneath the covers to build properties, bound / unbound methods and class methods. If you look up the property class in Python’s documentation, you will see that it follows the descriptor protocol very closely:

> property(fget=None, fset=None, fdel=None, doc=None)

It clearly shows that the property class has a getter, setter and a deleting method.

Let’s look at another example where we use a descriptor to do validation:

```
from weakref import WeakKeyDictionary
```

```
class Drinker:
    def __init__(self):
        self.req_age = 21
        self.age = WeakKeyDictionary()

    def __get__(self, instance_obj, objtype):
        return self.age.get(instance_obj, self.req_age)

    def __set__(self, instance, new_age):
        if new_age < 21:
            msg = '{name} is too young to legally imbibe'
            raise Exception(msg.format(name=instance.name))
        self.age[instance] = new_age
        print('{name} can legally drink in the USA'.format(
            name=instance.name))

    def __delete__(self, instance):
        del self.age[instance]


class Person:
    drinker_age = Drinker()

    def __init__(self, name, age):
        self.name = name
        self.drinker_age = age


p = Person('Miguel', 30)
p = Person('Niki', 13)
```

Once again, we create a descriptor class. In this case, we use Python's **weakref** library's **WeakKeyDictionary**, which is a neat class that creates a dictionary that maps keys weakly. What this means is that when there are no strong references to a key in the dictionary, that key and its value will be discarded. We are using that in this example to prevent our Person instances from hanging around indefinitely.

Anyway, the part of the descriptor that we care most about is in our __set__ method. Here we check to see that the instance's **age** parameter is greater than 21, which is what you would need to be in the USA if you wanted to drink an alcoholic beverage. If you're age is lower, then it will raise an exception. Otherwise it will print out the name of the person and a message. To test out our descriptor, we create two instances with one that is greater than 21 in age and one that is less. If you run this code you should see the following output:

```
Miguel can legally drink in the USA
```

```
Traceback (most recent call last):
  File "desc_validator.py", line 32, in <module>
    p = Person('Niki', 13)

  File "desc_validator.py", line 28, in __init__
    self.drinker_age = age
  File "desc_validator.py", line 14, in __set__
    raise Exception(msg.format(name=instance.name))
Exception: Niki is too young to legally imbibe
```

That obviously worked the way it was supposed to, but it's not really obvious how it worked. The reason this works the way it does is that when we go to set **drinker_age**, Python notices that it is a descriptor. Python knows that **drinker_age** is a descriptor because we defined it as such when we created it as a class attribute:

```
drinker_age = Drinker()
```

So when we go to set it, we actually call our descriptor's __set__ method which passes in the instance and the age that we are trying to set. If the age is less than 21, then we raise an exception with a custom message. Otherwise we print out a message that you are old enough.

Getting back to how this all works, if we were to try to print out the drinker_age, Python would execute Person.drinker_age.__get__. Since drinker_age is a descriptor, its __get__ is what actually gets called. If you wanted to set the drinker_age, you would do this:

```
p.drinker_age = 32
```

Python would then call **Person.drinker_age.__set__** and since that method is also implemented in our descriptor, then the descriptor method is the one that gets called. Once you trace your way through the code execution a few times, you will quickly see how this all works.

The main thing to remember is that descriptors are linked to classes and not to instances.

Wrapping Up

Descriptors are pretty important because of all the places they are used in Python's source code. They can be really useful to you too if you understand

how they work. However, their use cases are pretty limited and you probably won't be using them very often. Hopefully this chapter will have helped you see the descriptor's usefulness and when you might want to use one yourself.