Python Data vs. Non-data Descriptors

Summary: in this tutorial, you'll learn the differences between data and non-data descriptors.

Descriptors (https://www.pythontutorial.net/python-oop/python-descriptors/) have two types:

- Data descriptors are objects of a class that implements `__set__` method (and/or `__delete__` method)
- Non-data descriptors are objects of a class that have the `__get__` method only.

Both descriptor types can optionally implement the `__set_name__` method. The `__set_name__` method doesn't affect the classification of the descriptors.

The descriptor types determine how Python resolves object's attributes lookup.

Non-data descriptor

If a class uses a non-data descriptor, Python will search the attribute in instance attributes first (`instance.__dict__`). If Python doesn't find the attribute in the instance attributes, it'll use the data descriptor.

Let's take a look at the following example.

First, define a non-data descriptor class `FileCount` that has the `__get__` method which returns the number of files in a folder:

```
class FileCount:
    def __get__(self, instance, owner):
        print('The __get__ was called')
        return len(os.listdir(instance.path))
```

Second, define a `Folder` class that uses the `FileCount` descriptor:

```
class Folder:
    count = FileCount()

    def __init__(self, path):
        self.path = path
```

Third, create an instance of the `Folder` class and access the `count` attribute:

```
folder = Folder('/')
print('file count: ', folder.count)
```

Python called the `__get__` descriptor:

```
The __get__ was called
file count:  32
```

After that, set the `count` attribute of the `folder` instance to `100` and access the count attribute:

```
folder.__dict__['count'] = 100
print('file count: ', folder.count)
```

Output:

```
file count:  100
```

In this example, Python can find the `count` attribute in the instance dictionary `__dict__`. Therefore, it does not use data descriptors.

Data descriptor

When a class has a data descriptor, Python will look for an instance's attribute in the data descriptor first. If Python doesn't find the attribute, it'll look for the attribute in the instance dictionary (`__dict__`). For example:

First, define a `Coordinate` descriptor class:

```
class Coordinate:
    def __get__(self, instance, owner):
        print('The __get__ was called')

    def __set__(self, instance, value):
        print('The __set__ was called')
```

Second, define a `Point` class that uses the `Coordinate` descriptor:

```
class Point:
    x = Coordinate()
    y = Coordinate()
```

Third, create a new instance of the `Point` class and assign a value to the `x` attribute of the `p` instance:

```
p = Point()
p.x = 10
```

Output:

```
The __set__ was called
```

Python called the `__set__` method of the `x` descriptor.

Finally, access the `x` attribute of the `p` instance:

```
p.x
```

Output:

```
The __get__ was called
```

Python called the `__get__` method of the `x` descriptor.

Summary

- Data descriptors are objects of a class that implements `__set__` method (and/or `__delete__` method)
- Non-data descriptors are objects of a class that have the `__get__` method only.
- When accessing object's attributes, data descriptors override the instance's attributes and instance's attributes override non-data descriptors.