3.4.2.2. Implementing Descriptors

The following methods only apply when an instance of the class containing the method (a so-called*descriptor* class) appears in an *owner*class (the descriptor must be in either the owner's class dictionary or in the class dictionary for one of its parents). In the examples below, "the attribute" refers to the attribute whose name is the key of the property in the owner class' `__dict__`.

`object.`**`__get__`**(*self*, *instance*,*owner*)

Called to get the attribute of the owner class (class attribute access) or of an instance of that class (instance attribute access).*owner* is always the owner class, while *instance* is the instance that the attribute was accessed through, or `None` when the attribute is accessed through the*owner*. This method should return the (computed) attribute value or raise an `AttributeError`exception.

`object.`**`__set__`**(*self*, *instance*,*value*)

Called to set the attribute on an instance *instance* of the owner class to a new value, *value*.

`object.`**`__delete__`**(*self*, *instance*)

Called to delete the attribute on an instance *instance* of the owner class.

3.4.2.3. Invoking Descriptors

In general, a descriptor is an object attribute with "binding behavior", one whose attribute access has been overridden by methods in the descriptor protocol: `__get__()`,`__set__()`, and `__delete__()`. If any of those methods are defined for an object, it is said to be a descriptor.

The default behavior for attribute access is to get, set, or delete the attribute from an object's dictionary. For instance, `a.x` has a lookup chain starting with `a.__dict__['x']`, then`type(a).__dict__['x']`, and continuing through the base classes of `type(a)`excluding metaclasses.

However, if the looked-up value is an object defining one of the descriptor methods, then Python may override the default behavior and invoke the descriptor method instead. Where this occurs in the precedence chain depends on which descriptor methods were defined and how they were called. Note that descriptors are only invoked for new style objects or classes (ones that subclass `object()`or `type()`).

The starting point for descriptor invocation is a binding, `a.x`. How the arguments are assembled depends on `a`:

Direct Call

The simplest and least common call is when user code directly invokes a descriptor method:`x.__get__(a)`.

Instance Binding

If binding to a new-style object instance, `a.x` is transformed into the call:`type(a).__dict__['x'].__get__(a,type(a))`.

Class Binding

If binding to a new-style class,`A.x` is transformed into the call:`A.__dict__['x'].__get__(None, A)`.

Super Binding

If `a` is an instance of `super`, then the binding `super(B, obj).m()` searches `obj.__class__.__mro__` for the base class `A` immediately preceding `B` and then invokes the descriptor with the call: `A.__dict__['m'].__get__(obj,obj.__class__)`.

For instance bindings, the precedence of descriptor invocation depends on the which descriptor methods are defined. A descriptor can define any combination of `__get__()`, `__set__()` and `__delete__()`. If it does not define `__get__()`, then accessing the attribute will return the descriptor object itself unless there is a value in the object's instance dictionary. If the descriptor defines `__set__()` and/or `__delete__()`, it is a data descriptor; if it defines neither, it is a non-data descriptor. Normally, data descriptors define both `__get__()` and `__set__()`, while non-data descriptors have just the `__get__()` method. Data descriptors with `__set__()` and `__get__()` defined always override a redefinition in an instance dictionary. In contrast, non-data descriptors can be overridden by instances.

Python methods (including `staticmethod()` and `classmethod()`) are implemented as non-data descriptors. Accordingly, instances can redefine and override methods. This allows individual instances to acquire behaviors that differ from other instances of the same class.

The `property()` function is implemented as a data descriptor. Accordingly, instances cannot override the behavior of a property.