3.4.2. Customizing attribute access

The following methods can be defined to customize the meaning of attribute access (use of, assignment to, or deletion of `x.name`) for class instances.

`object.`**`__getattr__`**(*self*, *name*)

Called when an attribute lookup has not found the attribute in the usual places (i.e. it is not an instance attribute nor is it found in the class tree for `self`). `name` is the attribute name. This method should return the (computed) attribute value or raise an`AttributeError` exception.

Note that if the attribute is found through the normal mechanism,`__getattr__()` is not called. (This is an intentional asymmetry between `__getattr__()` and`__setattr__()`.) This is done both for efficiency reasons and because otherwise `__getattr__()`would have no way to access other attributes of the instance. Note that at least for instance variables, you can fake total control by not inserting any values in the instance attribute dictionary (but instead inserting them in another object). See the`__getattribute__()` method below for a way to actually get total control in new-style classes.

`object.`**`__setattr__`**(*self*, *name*,*value*) ¶

Called when an attribute assignment is attempted. This is called instead of the normal mechanism (i.e. store the value in the instance dictionary). *name*is the attribute name, *value* is the value to be assigned to it.

If `__setattr__()` wants to assign to an instance attribute, it should not simply execute `self.name =value` — this would cause a recursive call to itself. Instead, it should insert the value in the dictionary of instance attributes, e.g., `self.__dict__[name] = value`. For new-style classes, rather than accessing the instance dictionary, it should call the base class method with the same name, for example,`object.__setattr__(self, name,value)`.

`object.`**`__delattr__`**(*self*, *name*)

Like `__setattr__()` but for attribute deletion instead of assignment. This should only be implemented if `del obj.name` is meaningful for the object.

3.4.2.1. More attribute access for new-style classes

The following methods only apply to new-style classes.

`object.`**`__getattribute__`**(*self*,*name*)

Called unconditionally to implement attribute accesses for instances of the class. If the class also defines `__getattr__()`, the latter will not be called unless`__getattribute__()` either calls it explicitly or raises an`AttributeError`. This method should return the (computed) attribute value or raise an`AttributeError` exception. In order to avoid infinite recursion in this method, its implementation should always call the base class method with the same name to access any attributes it needs, for example,`object.__getattribute__(self,name)`.

> **Note** This method may still be bypassed when looking up special methods as the result of implicit invocation via language syntax or built-in functions. See Special method lookup for new-

style classes.