

Ordering Information: **Python How to Program**

- View the complete **Table of Contents**
- Read the **Preface**
- Download the **Code Examples**

To view all the Deitel products and services available, visit the Deitel Kiosk on InformIT at www.informIT.com/deitel.

To follow the Deitel publishing program, sign-up now for the *DEITEL*TM *BUZZ ON-LINE* e-mail newsletter at www.deitel.com/newsletter/subscribeinformIT.html
To learn more about Deitel instructor-led corporate training delivered at your location, visit www.deitel.com/training or contact Christi Kelsey at (978) 461-5880 or e-mail: christi.kelsey@deitel.net.

Note from the Authors: This article is an excerpt from Chapter 9, Section 9.12.5 of Python How to Program. This article describes how to implement properties for a class. Properties are new to the Python language—they were added in version 2.2. Properties enable the programmer to provide access methods on a per-attribute basis. We discuss how to define get, set and delete methods for a private attribute and how to create properties that use these methods to manipulate the attributes. Readers should be familiar OOP (constructors, the object reference argument, private attributes, class customization) and raising exceptions. The code examples included in this article show readers examples using the DeitelTM signature LIVE-CODETM Approach, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.





476 Object-Oriented Programming: Inheritance

Chapter 9

9.13 Properties

In versions of Python before 2.2, classes and types were two distinct programming elements. The differences between types and classes contradicts the notion that classes are programmer-defined types. Many Python programmers, as well as the developers of the language, also disliked the limitations of this needless difference between classes and types. For example, because types are not classes, programmers cannot inherit from builtin types to take advantage of Python's high-level data manipulation capabilities provided by lists, dictionaries and other objects.

Beginning with Python 2.2, the nature and behavior of classes will change, to remove the difference between types and classes. In all future 2.x releases, a programmer can distinguish between two kinds of classes—so-called "classic" classes that behave in the same manner as the classes in previous versions of Python, and "new" classes that exhibit new behavior. Python 2.2 provides type **object** to define new classes. Any class that directly or indirectly inherits from **object** exhibits all the behaviors defined for a new class, which include many advanced object-oriented features. In this article, we investigate properties—a new ability that allows programmers to define access methods on a per-attribute basis.

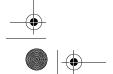
Properties describe object attributes. A program accesses an object's properties using object-attribute syntax. However, a class definition creates a property by specifying up to four components—a get method that executes when a program accesses the property's value, a set method that executes when a program sets the property's value, a delete method that executes when a program deletes the value (e.g., with keyword **del**) and a docstring that describes the property. The get, set and delete methods can perform the tasks that maintain an object's data in a consistent state. Thus, properties provide an additional way for programmers to control access to an object's data¹.

Figure 9.17 defines class **Time** to contain attributes **hour**, **minute** and **second** as properties. The constructor (lines 7–12) creates private attributes __hour, __minute and __second. Typically, classes that use properties define their attributes to be private, to hide the data from clients of the class. The clients of the class then access the public properties of that class, which *get* and *set* the values of the private attributes.

```
# Fig. 9.17: TimeProperty.py
 2
    # Class Time with properties
 3
    class Time( object ):
 5
       """Class Time with hour, minute and second properties"""
 6
7
           __init__( self, hourValue, minuteValue, secondValue ):
8
          """Time constructor, takes hour, minute and second"""
10
          self.__hour = hourValue
11
          self. minute = minuteValue
12
          self.__second = secondValue
```

Fig. 9.17 Properties—class **Time**. (Part 1 of 3.)

1. Other ways to control attribute access include programmer-defined access methods, special meth-__delattr__ and special method __getattr__, __setattr__ and _getattribute__ (for classes that inherit from object). Chapters 7-9 of Python How to Program cover these, and other, object-oriented subjects.











477

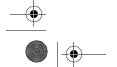


Chapter 9

Object-Oriented Programming: Inheritance

```
13
14
       def __str__( self ):
15
           """String representation of an object of class Time"""
16
17
          return "%.2d:%.2d:%.2d" % \
18
              ( self.__hour, self.__minute, self.__second )
19
20
       def deleteValue( self ):
21
          """Delete method for Time properties"""
22
23
          raise TypeError, "Cannot delete attribute"
24
25
       def setHour( self, value ):
           """Set method for hour attribute"""
26
27
28
          if 0 <= value < 24:</pre>
29
              self.__hour = value
30
          else:
31
             raise ValueError, \
                 "hour (%d) must be in range 0-23, inclusive" % value
32
33
34
       def getHour( self ):
35
            ""Get method for hour attribute"""
36
37
          return self. hour
38
39
       # create hour property
40
       hour = property( getHour, setHour, deleteValue, "hour" )
41
42
       def setMinute( self, value ):
43
           """Set method for minute attribute"""
44
          if 0 <= value < 60:</pre>
45
46
              self.__minute = value
47
          else:
48
             raise ValueError, \
49
                 "minute (%d) must be in range 0-59, inclusive" % value
50
51
       def getMinute( self ):
52
           """Get method for minute attribute"""
53
54
          return self.__minute
55
56
       # create minute property
57
       minute = property( getMinute, setMinute, deleteValue, "minute" )
58
59
       def setSecond( self, value ):
60
           """Set method for second attribute"""
61
62
          if 0 <= value < 60:</pre>
63
              self.__second = value
```

Fig. 9.17 Properties—class **Time**. (Part 2 of 3.)













478 Object-Oriented Programming: Inheritance

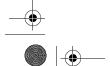
Chapter 9

```
64
          else:
65
             raise ValueError, \
                 "second (%d) must be in range 0-59, inclusive" % value
66
67
68
       def getSecond( self ):
69
          """Get method for second attribute"""
70
71
          return self.__second
72
73
       # create second property
74
       second = property( getSecond, setSecond, deleteValue, "second" )
```

```
Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from TimeProperty import Time
>>>
>>> time1 = Time( 5, 27, 19 )
>>> print time1
05:27:19
>>> print time1.hour, time1.minute, time1.second
5 27 19
>>> time1.hour, time1.minute, time1.second = 16, 1, 59
>>> print time1
16:01:59
>>>
>>> time1.hour = 25
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "TimeProperty.py", line 31, in setHour
    raise ValueError, \
ValueError: hour (25) must be in range 0-23, inclusive
>>> time1.minute = -3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "TimeProperty.py", line 48, in setMinute
    raise ValueError, \
ValueError: minute (-3) must be in range 0-59, inclusive
>>> time1.second = 99
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "TimeProperty.py", line 65, in setSecond
    raise ValueError, \
ValueError: second (99) must be in range 0-59, inclusive
```

Fig. 9.17 Properties—class **Time**. (Part 3 of 3.)

Method **deleteValue** (lines 20–23) raises an exception to prevent a client from deleting an attribute. We use this method to create properties that the client cannot delete. Each property (hour, minute and second) defines corresponding get and set methods.









479



Chapter 9

Object-Oriented Programming: Inheritance

Each get method takes only the object reference (self) as an argument and returns the property's value. Each set method takes two arguments—the object-reference argument and the new value for the property. Lines 25–32 define the set method (setHour) for the **hour** property. If the new value is within the appropriate range, the method assigns the new value to the property; otherwise, the method raises an exception. Method getHour (lines 34–37) is the **hour** property's *get* method, which simply returns the value of the corresponding private attribute (__hour)Built-in function property (line 40) takes as arguments a get method, a set method, a delete method and a docstring and returns a property for the class. Line 40 creates the hour property by passing to function property methods getHour, setHour and deleteValue and the string "hour". Clients access properties, using the dot (.) access operator. When the client uses a property as an rvalue, the property's get method executes. When the client uses the property as an lvalue, the property's set method executes. When the client deletes the property with keyword del, the property's delete method executes. The remainder of the class definition (lines 42–74) defines get and set methods for properties **minute** (created in line 57) and **second** (created in line 74).

Software Engineering Observation 9.1

Function **property** does not require that the caller pass all four arguments. Instead, the caller can pass values for keyword arguments **fget**, **fset**, **fdel** and **doc** to specify the property's get, set and delete methods and the docstring, respectively.

The interactive session in Fig. 9.17 highlights the benefits of properties. A client of the class can access an object's attributes, using the dot access operator, but the class author also can ensure data integrity. Properties have added advantages over implementing methods __setattr__, __getattr__ and __delattr__. For example, class authors can state explicitly the attributes for which the client may use the dot access notation. Additionally, the class author can write separate *get*, *set* and *delete* methods for each attribute, rather than using if/else logic to determine which attribute to access.

