`int` or `float` functions. For example, `float(5)` will convert the integer 5 to the floating-point number 5.0. When converting floating-point numbers to integers, Python always truncates the fractional part of the number. For example, `int(3.99999)` will convert the floating-point number 3.99999 to the integer 3.

In summary, we have seen that Python supports several different types of primitive objects in the number family: integers for ordinary simple math; or, when precision is required or when dealing with very large numbers; floating-point numbers, for working with scientific applications or accounting applications where we need to keep track of dollars and cents. We have seen that Python can be used to make simple numerical calculations. However, at this point Python is nothing more than a calculator. In the next section we will add some additional Python primitives that will give us a lot more power.

1.5.2 Naming Objects

Very often we have an object that we would like to remember. Python allows us to **name** objects so that we can refer to them later. For example, we might want to use the name *pi* rather than the value 3.14159 in a mathematical expression. We might also want to give a name to a value that we are going to use over and over again rather than recalculating it each time.

In Python we can name objects using an **assignment statement**. A statement is like an expression except that it does not produce a value for the read–eval–print loop to print. An assignment statement has three parts: (1) the left-hand side, (2) the right-hand side, and (3) the assignment operator (=). The left side contains the name we are assigning to a variable, and the right side can be any Python expression.

```
variableName = python expression
```

When the Python interpreter evaluates an assignment statement, it first evaluates the expression that it finds on the right-hand side of the equals sign. Once the right-hand side expression has been evaluated, the resulting object is referred to using the name found on the left side of the equals sign. In computer science, we call these names **variables**. More formally, we define a variable to be a named reference to a data object. In other words, a variable is simply a name that allows us to locate a Python object.

Suppose we want to calculate the volume of a cylinder where the radius of the base is 8 cm and the height is 16 cm. We will use the formula *volume = area of base * height.* Rather than calculate everything in one big expression, we will divide the work into several assignment statements. First, we will name the numeric objects "pi," "radius," and "height." Next, we will use the named objects to calculate the area of the base and finally the volume of the cylinder. Session 1.5 shows how we use this sequence of assignment statements and Python arithmetic to solve our problem.

```
>>> pi = 3.14159
>>> radius = 8.0
>>> height = 16
>>> baseArea = pi * radius ** 2
>>> cylinderVolume = baseArea * height
>>> baseArea
201.06175999999999
>>> cylinderVolume
3216.9881599999999
>>>
```

Session 1.5 Calculating the volume of a sphere with assignment statements

After studying Session 1.5, you may have some questions:

- How is the use of the equals sign in Python different from what you learned in math class?
- If you change the value for `baseArea` will `cylinderVolume` automatically change?
- Why doesn't Python print out the value of `pi` after the first assignment statement?
- What are the legal values for names in Python?

Let's look at these questions one at a time. The equals sign in Python is very different from what you learned in math class. In fact, you should think of it not in terms of equality but rather as the assignment operator, which has the job of associating a name with an object. Figure 1.5 illustrates how names are associated with objects in Python. All the names and objects in this figure come from Session 1.5. The relationships between names and the objects they reference are indicated by the arrows between them.

Another way of thinking about assignment is to imagine that an assignment statement is like taking a sticky label with a name written on it and attaching it to an object. You know that you can put more than one sticky label on an object in the real world, and the

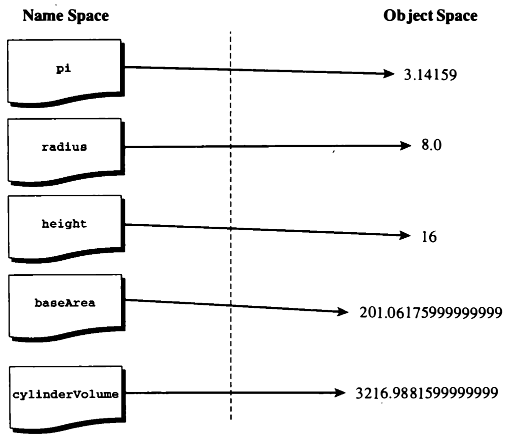

Figure 1.5 A reference diagram illustrating simple assignment in Python

analogy holds true in the Python world as well. A Python object can have more than one name. For example, suppose you made the following additional assignment $x = 8.0$. After executing that statement, you could add another label called `x` with another arrow pointing at the object 8.0, as shown in Figure 1.6.

Variables can take the place of the actual object in a Python expression. When Python evaluates the expression `pi * radius ** 2`, it first looks up `pi` and `radius` to see what objects they refer to and then substitutes the values into the expression. The expression thus becomes `3.14159 * 8.0 ** 2`. Next, Python evaluates `8.0 **2` to get an intermediate result of `64.0`. Python then evaluates `3.14159 * 64.0` to get the value `201.06176`. After the right-hand side of the expression is evaluated, Python assigns the name `baseArea` to `201.06176`.

Let's look at one more example using assignment. Consider the Python statements in Session 1.6.

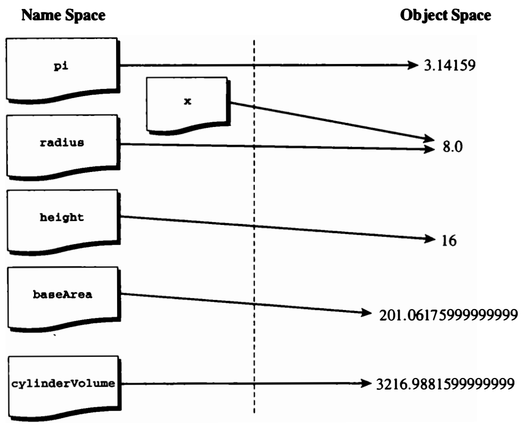

Figure 1.6 Reference diagram after x = 8.0

```
>>> a = 10
>>> b = 20
>>> a = b
```

Session 1.6 Python evaluates assignment statements in sequence

After these three assignment statements, what object does a refer to? As you think about this question it is very important to remember that Python evaluates the statements from top to bottom, one after another. Let's rephrase what is going on in the order that Python performs its work.

1. Assign the name a to the integer object 10.
2. Assign the name b to the integer object 20.
3. Find the object named b, then assign the name a to that object.

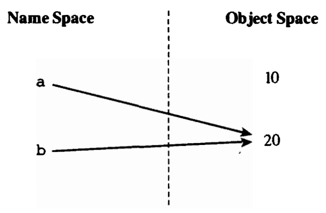

Figure 1.7 Result of assignment `a = b`

The answer to our question is that `a` now refers to the object 20. This is shown in Figure 1.7. In addition, since we have not changed what `b` refers to since the original assignment, `b` continues to refer to the object 20. If you are confused by this example, try to draw the reference diagram yourself one step at a time.

Now that you understand more about the assignment operator you will understand that attaching the name **`baseArea`** to a different object will have no impact on the name `cylinderVolume`. To make the new value for **`baseArea`** change the value of `cylinderVolume`, you will need to ask Python to reevaluate the expression **`cylinderVolume = baseArea * height`**.

Since assignment is a statement rather than an expression, it does not return a value. This means that there is nothing for the read–eval–print loop to print out. That is why you do not see any output following the assignment statements in Sessions 1.5 and 1.6.

A name on a line all by itself is a very simple Python expression. Notice that just typing a name causes Python to find the value for the object and return it as the result of the expression. You can see examples of this in Session 1.5.

There are several important rules to remember about naming things in Python. Names can include any letter, number, or an _ (underscore). Names must start with either a letter or an _. Python is case sensitive, which means that the names `baseArea`, `basearea`, and `BaseArea` are all different. Some names are reserved by Python for its own use. These are called **keywords** and correspond to important Python capabilities that you will learn about. Table 1.1 shows you all of Python's reserved keywords.

and	continue	else	for	import	not	raise
assert	def	except	from	in	or	return
break	del	exec	global	is	pass	try
class	elif	finally	if	lambda	print	while

Table 1.1 Python's reserved words

Exercises

1.17 Given the following Python statements:

```
a = 79
b = a 79
a = 89
```

$$\begin{tikzcd}
a \arrow[r] & 79 \\
b \arrow[ru] & \\
a \arrow[r] & 89
\end{tikzcd}$$

(a) Draw a reference diagram to show the labels and objects after the first two statements.

(b) Draw a reference diagram to show the labels and objects after the last statement.

1.18 Which of the following are legal variable names:

[✓ (a)] abc123
~~(b)~~ 123abc
[✓ (c)] abc123_
[✓ (d)] _123

1.19 Consider the following statements:

```
a = 10
b = 20
c = a * b  200
d = a + b  30
```