CHAPTER 5

Conditionals, Loops, and Some Other Statements

By now, I’m sure you are getting a bit impatient. All right—all these data types are just dandy, but you can’t really do much with them, can you?

Let’s crank up the pace a bit. You’ve already encountered a few statement types (print statements, import statements, and assignments). Let’s first take a look at some more ways of using these before diving into the world of conditionals and loops. Then you’ll see how list comprehensions work almost like conditionals and loops, even though they are expressions, and finally you’ll take a look at pass, del, and exec.

More About print and import

As you learn more about Python, you may notice that some aspects of Python that you thought you knew have hidden features just waiting to pleasantly surprise you. Let’s take a look at a couple of such nice features in print¹ and import.

Tip For many applications, logging (using the logging module) will be more appropriate than using print. See Chapter 19 for more details.

Printing with Commas

You’ve seen how print can be used to print an expression, which is either a string or automatically converted to one. But you can actually print more than one expression, as long as you separate them with commas:

```python
>>> print 'Age:', 42
Age: 42
```

As you can see, a space character is inserted between each argument.

¹ In Python 3.0, print is no longer a statement at all—it’s a function (with essentially the same functionality).
Note The arguments of print do not form a tuple, as one might expect:

```python
>>> 1, 2, 3
(1, 2, 3)
>>> print 1, 2, 3
1 2 3
>>> print (1, 2, 3)
(1, 2, 3)
```

This behavior can be very useful if you want to combine text and variable values without using the full power of string formatting:

```python
>>> name = 'Gumby'
>>> salutation = 'Mr.'
>>> greeting = 'Hello,'
>>> print greeting, salutation, name
Hello, Mr. Gumby
```

If the greeting string had no comma, how would you get the comma in the result? You couldn’t just use

```python
print greeting, ',', salutation, name
```

because that would introduce a space before the comma. One solution would be the following:

```python
print greeting + ',', salutation, name
```

which simply adds the comma to the greeting.

If you add a comma at the end, your next print statement will continue printing on the same line. For example, the statements

```python
print 'Hello,,'
print 'world!'
```

print out Hello, world!.

**Importing Something As Something Else**

Usually, when you import something from a module, you either use

```python
import somemodule
```
or

```python
from somemodule import somefunction
```
or

---

2. This will work only in a script, and not in an interactive Python session. In the interactive session, each statement will be executed (and print its contents) separately.