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™ BUZZ ON-LINE* e-mail newsletter at **www.deitel.com/newsletter/subscribeinformIT.html**. To learn more about **Python programming courses** or any of our Deitel instructor-led corporate training courses that can be delivered at your location, visit **www.deitel.com/training** or contact Christi Kelsey, Director of Corporate Training Programs, at (978) 461-5880 or e-mail: **christi.kelsey@deitel.net**.

Note from the Authors: This article is an excerpt from Chapter 4, Section 4.9 of *Python How to Program*. This article discusses how the various forms of the **`import`** statement affect a program's namespace. We explore all the forms of the statement and use function **`dir`** in interactive sessions to demonstrate the **`import`** statement's effects on the namespace. Readers should be familiar with defining functions, importing modules, basic scoping, namespaces and function **`dir`**. The code examples included in this article show readers examples using the Deitel™ signature *LIVE-CODE™ Approach*, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

4.9 Keyword `import` and Namespaces

Python modules are logical blocks of code that contain related definitions (e.g., of functions or classes). For example, the **`math`** module contains function definitions that perform common mathematical operations. To use a module, a Python program must *import* the module using keyword **`import`**. In this article, we explore how importing a module affects a program's namespace and discuss how to import modules into a program.

4.9.1 Importing One or More Modules

Consider a program that needs to perform one of the specialized mathematical operations defined in module **`math`**. The program must first import the module with the line:

```
import math
```

The code that imports the module now has a reference to the **`math`** module in its namespace. After the **`import`** statement, the program may access any identifiers defined in the **`math`** module.

The interactive session in Fig. 4.11 demonstrates how an **`import`** statement affects the session's namespace and how a program can access identifiers defined in a module's namespace. The first line imports the **`math`** module. The next line then calls function **`dir`** to demonstrate that the identifier **`math`** has been inserted in the session's namespace. As the subsequent **`print`** statement shows, the identifier is bound to an object that represents the **`math`** module. If we pass identifier **`math`** to function **`dir`**, the function returns a list of all the identifiers in the **`math`** module's namespace.[3] [*Note*: In this article, we use Python version 2.2. Earlier versions of Python may return different results for function **`dir()`**.]

3. Actually, function **`dir`** returns a list of attributes for the object passed as an argument. In the case of a module, this information amounts to a list of all identifiers (e.g., classes, functions and data) defined in the module.

```
Python 2.2 (#28, Dec 21 2001, 12:21:22) [MSC 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more informa-
tion.
>>> import math
>>> dir()
['__builtins__', '__doc__', '__name__', 'math']
>>> print math
<module 'math' (built-in)>
>>> dir(math)
['__doc__', '__name__', 'acos', 'asin', 'atan', 'atan2', 'ceil',
'cos', 'cosh','e', 'exp', 'fabs', 'floor', 'fmod', 'frexp', 'hy-
pot', 'ldexp', 'log', 'log10','modf', 'pi', 'pow', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
>>> math.sqrt( 9.0 )
3.0
```

Fig. 4.11 Importing a module.

The next command in the session invokes function **sqrt**. To access an identifier in the **math** module's namespace, we must use the dot (**.**) access operator. The line

```
math.sqrt( 9.0 )
```

first accesses (with the dot access operator) function **sqrt** defined in the **math** module's namespace. The line then invokes (with the parentheses operator) the **sqrt** function, passing an argument of **9.0**.

If a program needs to import several modules, the program can include a separate **import** statement for each module. A program can also import multiple modules in one statement, by separating the module names with commas. Each imported module is added to the program's namespace as demonstrated in the interactive session of Fig. 4.12.

```
Python 2.2 (#28, Dec 21 2001, 12:21:22) [MSC 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more informa-
tion.
>>> import math, random
>>> dir()
['__builtins__', '__doc__', '__name__', 'math', 'random']
```

Fig. 4.12 Importing more than one module.

4.9.2 Importing identifiers from a Module

Previously we discussed how to access an identifier defined in another module's namespace. To access such an identifier, the programmer must use the dot (**.**) access operator. Sometimes, a program uses only one identifier or a few identifiers from a module. In this case, it may be useful to import only those identifiers the program needs. Python provides the ***from*/*import*** statement to import one or more identifiers from a module directly into the program's namespace.

The interactive session in Fig. 4.13 imports the **sqrt** function directly into the session's namespace. When the interpreter executes the line

```
from math import sqrt
```

the interpreter creates a reference to function **math.sqrt** and places the reference directly into the session's namespace. Now, we can call the function directly without using the dot operator. Just as a program can import multiple modules in one statement, a program can import multiple identifiers from a module in one statement by separating the identifiers with commas. The line

```
from math import sin, cos, tan
```

imports **math** functions **sin**, **cos** and **tan** directly into the session's namespace. After the **import** statement, a call to function **dir** reveals references to each of these functions.

```
Python 2.2 (#28, Dec 21 2001, 12:21:22) [MSC 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more informa-
tion.
>>> from math import sqrt
>>> dir()
['__builtins__', '__doc__', '__name__', 'sqrt']
>>> sqrt( 9.0 )
3.0
>>> from math import sin, cos, tan
>>> dir()
['__builtins__', '__doc__', '__name__', 'cos', 'sin', 'sqrt',
'tan']
```

Fig. 4.13 Importing an identifier from a module.

The interactive session in Fig. 4.14 demonstrates that a program also may import *all* identifiers defined in a module. The statement

```
from math import *
```

imports all identifiers that do not start with an underscore from the **math** module into the interactive session's namespace. Now the programmer can invoke any of the functions from the **math** module, without accessing the function through the module name and the dot access operator. However, importing a module's identifiers in this way can lead to serious errors and is considered a dangerous programming practice. Consider a situation in which a program had defined an identifier named **e** and assigned it the string value **"e"**. After executing the preceding **import** statement, identifier **e** is bound to the mathematical floating-point constant *e*, and the previous value for **e** is no longer accessible. In general, a program should never import all identifiers from a module in this way.

Testing and Debugging Tip 4.3

Avoid importing all identifiers from a module into the namespace of another module. This method of importing a module should be used only for modules provided by trusted sources, whose documentation explicitly states that such a statement may be used to import the module.

```
Python 2.2 (#28, Dec 21 2001, 12:21:22) [MSC 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more informa-
tion.
>>> from math import *
>>> dir()
['__builtins__', '__doc__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'e', 'exp', 'fabs', 'floor',
'fmod', 'frexp', 'hypot', 'ldexp','log', 'log10', 'modf', 'pi',
'pow', 'sin', 'sinh', 'sqrt', 'tan', 'tanh']
```

Fig. 4.14 Importing all identifiers from a module.

4.9.3 Binding Names for Modules and Module Identifiers

We have already seen how a program can import a module or specific identifiers from a module. Python's syntax gives the programmer considerable control over how the **import** statement affects a program's namespace. In this section, we discuss this control in more detail and explain further how the programmer can customize the references to imported elements.

The statement

```
import random
```

imports the **random** module and places a reference to the module named **random** in the namespace. In the interactive session in Fig. 4.15, the statement

```
import random as randomModule
```

also imports the **random** module, but the ***as*** clause of the statement allows the programmer to specify the name of the reference to the module. In this case, we create a reference named **randomModule**. Now, if we want to access the **random** module, we use reference **randomModule**..

```
Python 2.2 (#28, Dec 21 2001, 12:21:22) [MSC 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more informa-
tion.
>>> import random as randomModule
>>> dir()
['__builtins__', '__doc__', '__name__', 'randomModule']
>>> randomModule.randrange( 1, 7 )
2
>>> from math import sqrt as squareRoot
>>> dir()
['__builtins__', '__doc__', '__name__', 'randomModule', 'square-
Root']
>>> squareRoot( 9.0 )
3.0
```

Fig. 4.15 Specifying names for imported elements.

A program also can use an **import**/**as** statement to specify a name for an identifier that the program imports from a module. The line

```
from math import sqrt as squareRoot
```

imports the **sqrt** function from module **math** and creates a reference to the function named **squareRoot**. The programmer may now invoke the function with this reference.

Typically, module authors use **import**/**as** statements, because the imported element may define names that conflict with identifiers already defined by the author's module. With the **import**/**as** statement, the module author can specify a new name for the imported elements, thereby avoiding the naming conflict. Programmers also use the **import**/**as** statement for convenience. A programmer may use the statement to rename a particularly long

identifier that the program uses extensively. The programmer specifies a shorter name for the identifier, thus increasing readability and decreasing the amount of typing.

Python's capabilities for importing elements into a program support component-based programming. The programmer should choose Python syntax appropriate for each situation, keeping in mind that the goal of component-based programming is to create programs that are easier to construct and maintain.