2.4. Everything Is an Object

In case you missed it, I just said that Python functions have attributes, and that those attributes are available at runtime.

A function, like everything else in Python, is an object.

Open your favorite Python IDE and follow along:

Example 2.3. Accessing the `buildConnectionString` Function's `doc string`

```
>>> import odbchelper                                    ❶
>>> params = {"server":"mpilgrim", "database":"master", "uid":"sa", "pwd":"secret"}
>>> print odbchelper.buildConnectionString(params)       ❷
server=mpilgrim;uid=sa;database=master;pwd=secret
>>> print odbchelper.buildConnectionString.__doc__      ❸
Build a connection string from a dictionary

Returns string.
```

❶ The first line imports the `odbchelper` program as a module -- a chunk of code that you can use interactively, or from a larger Python program. (You'll see examples of multi-module Python programs in Chapter 4.) Once you import a module, you can reference any of its public functions, classes, or attributes. Modules can do this to access functionality in other modules, and you can do it in the IDE too. This is an important concept, and you'll talk more about it later.

❷ When you want to use functions defined in imported modules, you need to include the module name. So you can't just say `buildConnectionString`; it must be `odbchelper.buildConnectionString`. If you've used classes in Java, this should feel vaguely familiar.

❸ Instead of calling the function as you would expect to, you asked for one of the function's attributes, `__doc__`.

`import` in Python is like `require` in Perl. Once you `import` a Python module, you access its functions with *`module.function`*; once you `require` a Perl module, you access its functions with *`module::function`*.

2.4.1. The Import Search Path

Before you go any further, I want to briefly mention the library search path. Python looks in several places when you try to import a module. Specifically, it looks in all the directories defined in `sys.path`. This is just a list, and you can easily view it or modify it with standard list methods. (You'll learn more about lists later in this chapter.)

Example 2.4. Import Search Path

```
>>> import sys                ❶
>>> sys.path                  ❷
['', '/usr/local/lib/python2.2', '/usr/local/lib/python2.2/plat-linux2',
'/usr/local/lib/python2.2/lib-dynload', '/usr/local/lib/python2.2/site-packages',
'/usr/local/lib/python2.2/site-packages/PIL', '/usr/local/lib/python2.2/site-packages/piddle']
>>> sys                       ❸
<module 'sys' (built-in)>
>>> sys.path.append('/my/new/path') ❹
```

❶ Importing the `sys` module makes all of its functions and attributes available.

❷ `sys.path` is a list of directory names that constitute the current search path. (Yours will look different, depending on your operating system, what version of Python you're running, and where it was originally installed.) Python will look through these directories (in this order) for a `.py` file matching the module name you're trying to import.

❸ Actually, I lied; the truth is more complicated than that, because not all modules are stored as `.py` files. Some, like the `sys` module, are "built-in modules"; they are actually baked right into Python itself. Built-in modules behave just like regular modules, but their Python source code is not available, because they are not written in Python! (The `sys` module is written in C.)

❹ You can add a new directory to Python's search path at runtime by appending the directory name to `sys.path`, and then Python will look in that directory as well, whenever you try to import a module. The effect lasts as long as Python is running. (You'll talk more about `append` and other list methods in Chapter 3.)

2.4.2. What's an Object?

Everything in Python is an object, and almost everything has attributes and methods. All functions have a built-in attribute `__doc__`, which returns the `doc string` defined in the function's source code. The `sys` module is an object which has (among other things) an attribute called `path`. And so forth.

Still, this begs the question. What is an object? Different programming languages define "object" in different ways. In some, it means that *all* objects *must* have attributes and methods; in others, it means that all objects are subclassable. In Python, the definition is looser; some objects have neither attributes nor methods (more on this in Chapter 3), and not all objects are subclassable (more on this in Chapter 5). But everything is an object in the sense that it can be assigned to a variable or passed as an argument to a function (more in this in Chapter 4).

This is so important that I'm going to repeat it in case you missed it the first few times: *everything in Python is an object*. Strings are objects. Lists are objects. Functions are objects. Even modules are objects.

Further Reading on Objects

- *Python Reference Manual* explains exactly what it means to say that everything in Python is an object, because some people are pedantic and like to discuss this sort of thing at great length.
- eff-bot summarizes Python objects.

[Report issues](#)