

Appendix B

Introduction to Jython

This Appendix contains an introduction to the Jython scripting language, along with a collection of Jython scripts that access databases such as MySQL, which is a relational database

What Is Jython?

Jython is a Java-based version of Python, and its homepage is here:

<http://www.jython.org>

Jython scripts can leverage the power of Java and Python. Since Jython is based on Python, Jython enforces the same indentation style that you have already seen in Python. Jython scripts look like Python scripts, and unlike the case with Java, you do not need to declare the type of a variable. Jython (like Python) will dynamically determine the type of a variable by the context in which you refer to that variable.

Jython allows you to use aliases for imported Java classes and for imported Python classes to simplify your code. Jython also allows you to define servlets and Java beans.

Download Jython from this Website:

<http://www.jython.org/downloads.html>

Note: Jython 2.5.3 is the most recent stable release of Jython.

Install Jython by navigating to the directory containing the Jython `JAR` file and then executing the following command:

```
java -jar jython_installer-2.5.3.jar
```

Now follow the prompts in the installer and complete the Jython installation.

Figure B.1 displays the initial screen that is displayed when you type the preceding command in a command shell on a Macbook Pro.



Figure B.1: The Jython initial installation screen

The directory that you selected during the installation process (whose default location is `$HOME/jython-2.5.3`) will contain the following files after you have completed the installation process:

```
ACKNOWLEDGMENTS
CoreExposed.includes
Demo
Doc
LICENSE.txt
LICENSE_Apache.txt
LICENSE_CPython.txt
Lib
NEWS
README.txt
bin
jython
jython.jar
registry
tests
```

Additional installation-related options for Jython are here:

<https://wiki.python.org/jython/InstallationInstructions>

Launching the Jython Interpreter

You can launch Jython by navigating to the directory that contains the Jython JAR file and then executing this command:

```
java -jar jython.jar
```

The first time that you launch Jython you will see output similar to the following:

```
*sys-package-mgr*: processing new jar,
'/Users/ocampesato/jython2.5.3/jython.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/cl
asses.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/ui
.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/js
se.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/j
ce.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/ch
arsets.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/AppleScriptEngine.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/dns_sd.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/j3daudio.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/j3dcore.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/j3dutils.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/jai_codec.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/jai_core.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/mlibwrapper_jai.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/MRJToolkit.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/QTJava.zip'
```

```

*sys-package-mgr*: processing new jar,
'/System/Library/Java/Extensions/vecmath.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/apple_provider.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/dnsns.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/localedata.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/sunjce_provider.jar'
*sys-package-mgr*: processing new jar,
'/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/sunpkcs11.jar'
Jython 2.5.3 (2.5:c56500f08d34+, Aug 13 2012, 14:48:36)
[Java HotSpot(TM) 64-Bit Server VM (Apple Inc.)] on java1.6.0_65
Type "help", "copyright", "credits" or "license" for more information.
>>>

```

If you type the word `quit`, Jython responds as follows:

```

>>> quit
Use quit() or Ctrl-D (i.e. EOF) to exit

```

If you type the word `exit`, Jython responds as follows:

```

>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>>

```

Now press `ctrl-d` and launch the Jython JAR file again with the same command, and this time you will see a much shorter output:

```

Jython 2.5.3 (2.5:c56500f08d34+, Aug 13 2012, 14:48:36)
[Java HotSpot(TM) 64-Bit Server VM (Apple Inc.)] on java1.6.0_65
Type "help", "copyright", "credits" or "license" for more information.
>>>

```

Instead of launching Java with the Jython JAR file, a simpler way is to add the location of the `jython` executable to the `PATH` variable, as shown here for OS X:

```
export PATH=$HOME/jython2.5.3:$PATH
```

The Python code in Jython works without any modification, so we'll focus on some of the Java-like aspects of Jython for the remainder of this chapter. If you have not

yet read the material pertaining to Python, now would be a good time to read (or skim through) the first two or three chapters.

Importing Java Classes in Jython

Jython examines JAR files and class files on its path and scans them for Java packages.

Sometimes Jython does not find packages that you need for your scripts, so you can use an alternative syntax.

For example, usually you would use this code snippet:

```
import javax.swing as swing
```

Try using this statement instead:

```
from javax.swing import swing
```

Use this statement if you want to import multiple packages:

```
from javax.swing import *
```

More information about package scanning is here:

<https://wiki.python.org/jython/PackageScanning>

Launching a Swing Frame in Jython

The following example shows you how to render a Java Swing-based frame in four lines without compiling any code or even defining a class. As a comparison, try writing the code in Listing B.1 with the equivalent GUI code using Java Swing.

<http://www.jython.org/jythonbook/en/1.0/GUIApplications.html>

Listing B.1 displays the contents of the Jython file `SwingFrame1.py` that launches a Java swing-based frame.

Listing B.1 SwingFrame1.py

```
import javax.swing as swing

myWindow = swing.JFrame("Hello from Jython!")
myWindow.size = (400, 400)
```

```
myWindow.show()
```

Open a command shell and type the following:

```
jython SwingFrame1.py
```

Although the `SwingFrame` does absolutely nothing, the important point is that you can launch this frame with only four lines of code! Now that you have been impressed with the capabilities of Jython, let's look at the details of the code.

The first line in Listing B.1 is an import statement that identifies the Java package

`javax.swing` by the alias `swing`.

The second line declares the variable `myWindow` as a `Swing JFrame`, followed by the third line that sets the dimensions of `myWindow` as `400x400`. The final line of code renders the `JFrame` referenced by the variable `myWindow`.

Accessing MySQL from Jython

You need to perform the following seven steps to access data in a MySQL database from a Jython script.

High-level summary of setup steps

Step 1: Install and start MySQL

Step 2: Create a database

Step 3: Create and populate a table

Step 4: Update `PATH` with the Jython directory

Step 5: Update `CLASSPATH` with the Jython JAR file

Step 6: Update `CLASSPATH` with a suitable MySQL JAR file

Step 7: Write a Jython script to query the MySQL data

If you are not sure of the details for each step in the preceding list, the instructions are provided in the following detailed explanation. Make sure that you correctly spell all directories and filenames as you complete each of the steps. Misspellings occur more often than you think; not only do they waste your time, they also make simple things

seem complicated.

The first step involves downloading MySQL from this link and then performing the installation:

<https://dev.mysql.com/downloads/mysql/>

The second and third steps (creating a database and a table and then populating the table with data) are discussed in detail in Chapter 9, so they will not be repeated here. Steps 4 and 5 were discussed at the beginning of this section. (Perform them now if you haven't done so already.) Step 6 involves copying the JAR file `mysql-connector-java-5.1.19-bin.jar` from the companion CD into a convenient directory and then updating `CLASSPATH` appropriately.

Step 7 is the actual Jython script that extracts the data from the table `simple1`, and it is presented in the following section.

Querying Data from a MySQL Table via Jython

Listing B.2 displays the contents of the Jython script `JythonQuery1.py` that demonstrates how to query the data that is stored in the table `simple1` that resides in the database `simplifiedb1`.

Listing B.2 JythonQuery1.py

```
import java

# instantiate a mySQL database driver
java.lang.Class.forName("org.gjt.mm.mysql.Driver")

# update these values appropriately
database = "simplifiedb1"
userName = "root"
password = ""

urlString = "jdbc:mysql://localhost/%s" % database
dbConnection = java.sql.DriverManager.getConnection(
    urlString,
    userName,
```

```

password)

sqlStatement = dbConnection.createStatement()
sqlQuery     = "SELECT * FROM simple1"
resultSet     = sqlStatement.executeQuery(sqlQuery)

while resultSet.next():
    print resultSet.getInt('id'), \
          resultSet.getString('name'), \
          resultSet.getString('userid')

```

Open a command shell and type the following command:

```
jython JythonQuery1.py
```

If everything is set up correctly, you will see the following output from the preceding command:

```

1000 John Smith jsmith
2000 Jane Edwards jedwards
3000 Tom Jones tjones

```

The data might be slightly different if you have updated the contents of the table `simple1` (or if you are using your own database and table).

Note that the variables `dbConnection`, `sqlStatement`, `sqlQuery`, and `resultSet` are often abbreviated as `conn`, `stmt`, `query`, and `rs`, respectively. Although these variable names do not follow good naming conventions, they have become practically standard usage because “everybody” understands them. (This convention is similar to using the variable `i` instead of `index` as a loop counter.)

Inserting a Row into a Table with Jython

Listing B.3 displays the contents of the Jython script `JythonInsertRow1.py` that demonstrates how to insert a row into a table.

Listing B.3 JythonInsertRow1.py

```

import java

# instantiate a MySQL database driver
java.lang.Class.forName("org.gjt.mm.mysql.Driver")

```



```
# update these values appropriately
database = "simplifiedb1"
userName = "root"
password = ""

urlString      = "jdbc:mysql://localhost/%s" % database
dbConnection = java.sql.DriverManager.getConnection(
                                                    urlString,
                                                    userName,
                                                    password)

sqlStatement = dbConnection.createStatement()
insertRow    =
    "INSERT simple1 VALUES(4000,'Samuel James','sjames')";
sqlStatement.executeUpdate(insertRow)
```

Open a command shell and type the following command:

```
jython insertRow1.py
```

Now issue the following statement in `mysql` to confirm the insertion:

```
SELECT * FROM simple1;

mysql> select * from simple1;
+-----+-----+-----+
| id    | name          | userid  |
+-----+-----+-----+
| 1000  | John Smith    | jsmith  |
| 2000  | Jane Edwards  | jedwards|
| 3000  | Tom Jones     | tjones  |
| 4000  | Samuel James  | sjames  |
+-----+-----+-----+
4 rows in set (0.01 sec)
```

Listing B.3 contains two lines in bold that consist of the string variable `insertRow` that contains a SQL `insert` statement, followed by the invocation of the `insert` command, as shown below:

```
insertRow    =
    "INSERT simple1 VALUES(4000,'Samuel James','sjames')";
sqlStatement.executeUpdate(insertRow)
```

For performance reasons, the preceding technique can be very inefficient, especially when there are many (hundreds or thousands) of simultaneous concurrent users. In the latter scenario, the preferred approach is to use prepared statements, as you will see in the next section.

Prepared Statements and Jython

Listing B.4 displays the contents of the Jython script `InsertRow2.py` that demonstrates

how to use prepared statements to insert a row into a database table.

Listing B.4 InsertRow2.py

```
import java

# instantiate a mySQL database driver
java.lang.Class.forName("org.gjt.mm.mysql.Driver")

# update these values appropriately
database = "simplifiedb1"
userName = "root"
password = ""

urlString      = "jdbc:mysql://localhost/%s" % database
dbConnection = java.sql.DriverManager.getConnection(
                                                    urlString,
                                                    userName,
                                                    password)

insertRow      = "INSERT INTO simple1 VALUES(?,?,?)"
pStatement = dbConnection.prepareStatement(insertRow)

pStatement.clearParameters()
pStatement.setInt(1,5000);
pStatement.setString(2,'Lady Gaga');
pStatement.setString(3, 'lgaga');

pStatement.executeUpdate()
pStatement.close()
dbConnection.close()
```

Open a command shell and type the following command:

```
jython InsertRow2.py
```

Now issue the following statement in `mysql` to confirm the insertion:

```
SELECT * FROM simple1;
```

If everything executed correctly, you will see the following output:

```
mysql> select * from simple1;
+-----+-----+-----+
| id    | name          | userid |
+-----+-----+-----+
| 1000  | John Smith    | jsmith |
| 2000  | Jane Edwards  | jedwards |
| 3000  | Tom Jones     | tjones |
```

```
| 4000 | Samuel James | sjames |
| 5000 | Lady Gaga    | lgaga  |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

As you can see, using prepared statements requires more work than in the previous example. You need to replace the following line of code:

```
insertRow =
    "INSERT simple1 VALUES(4000,'Samuel James','sjames')";
```

with the line shown below:

```
insertRow = "INSERT INTO simple1 VALUES(?,?,?)";
```

Think of each “question mark” (commonly called a *bind variable*) as a “placeholder” for an actual value that you must assign in your code, as shown here:

```
pStatement.setInt(1,5000);
pStatement.setString(2,'Lady Gaga');
pStatement.setString(3, 'lgaga');
```

Notice that three preceding statements specify three things: first the location of the bind variable, next the type of the bind variable, and then the actual value of the bind variable. Thus, the first line of code indicates that the first bind variable has the integer value of 5000; the second statement specifies that the second bind variable is a string with the value Lady Gaga, and the third statement indicates that the third bind variable is a string whose value is lgaga.

While it might be tempting to avoid using prepared statements, they can yield a significant performance improvement in applications that have a large number of concurrent users.

Creating HTML Pages with Jython

Listing B.5 displays the contents of the Jython script `GenHTML1.py` that demonstrates

how to query the data that is stored in the table `simple1` that resides in the database

`simplifiedb1` and then create an HTML page with an HTML table containing the retrieved data.

Listing B.5 GenHTML1.py

```
import java

# instantiate a mySQL database driver
java.lang.Class.forName("org.gjt.mm.mysql.Driver")

# update these appropriately
database = "simplifiedb1"
userName = "root"
password = ""

urlString      = "jdbc:mysql://localhost/%s" % database
dbConnection = java.sql.DriverManager.getConnection(
                                                    urlString,
                                                    userName,
                                                    password)

sqlStatement = dbConnection.createStatement()
sqlQuery     = "SELECT * FROM simple1"
resultSet     = sqlStatement.executeQuery(sqlQuery)

print "<html>"
print "<body>"
print "<table>"

while resultSet.next():
    print "<tr>"
    print "<td>" + str(resultSet.getInt('id')) + "</td>"
    print "<td>" + resultSet.getString('name') + "</td>"
    print "<td>" + resultSet.getString('userid') + "</td>"
    print "</tr>"

print "<table>"
print "</body>"
print "</html>"
```

Open a command shell and type the following command:

```
jython GenHTML1.py
```

and you will see the following output:

```
<html>
<body>
<table>
<tr>
<td>1000</td>
```

```

<td>John Smith</td>
<td>jsmith</td>
</tr>
<tr>
<td>2000</td>
<td>Jane Edwards</td>
<td>jedwards</td>
</tr>
<tr>
<td>3000</td>
<td>Tom Jones</td>
<td>tjones</td>
</tr>
<tr>
<td>4000</td>
<td>Samuel James</td>
<td>sjames</td>
</tr>
<tr>
<td>5000</td>
<td>Lady Gaga</td>
<td>lgaga</td>
</tr>
<table>
</body>
</html>

```

The data might be slightly different if you have updated the contents of the table
`simple1` (or if you are using your own database and table).

You can find additional code samples in the Jython distribution in the `Demo`
subdirectory, which contains the following code samples:

```

./applet/deprecated/ButtonDemo.py
./applet/deprecated/ButtonFontDemo.py
./applet/deprecated/CheckboxDemo.py
./applet/deprecated/ChoiceDemo.py
./applet/deprecated/Converter.py
./applet/deprecated/CoordinatesDemo.py
./applet/deprecated/HelloApplet.py
./applet/deprecated/HelloWorld.py
./applet/deprecated/LabelDemo.py
./applet/deprecated/ListDemo.py
./awt/Colors.py
./awt/Graph.py
./awt/simple.py
./bean/TempConverter.py
./javaclasses/deprecated/Graph.py
./modjy_webapp/demo_app.py
./swing/JythonConsole/Action.py
./swing/JythonConsole/Console.py
./swing/JythonConsole/Keymap.py

```

```
./swing/JythonConsole/Styles.py  
./swing/ListDemo.py  
./swing/ObjectTree.py  
./swing/simple.py  
./swing/TreeDemo.py
```