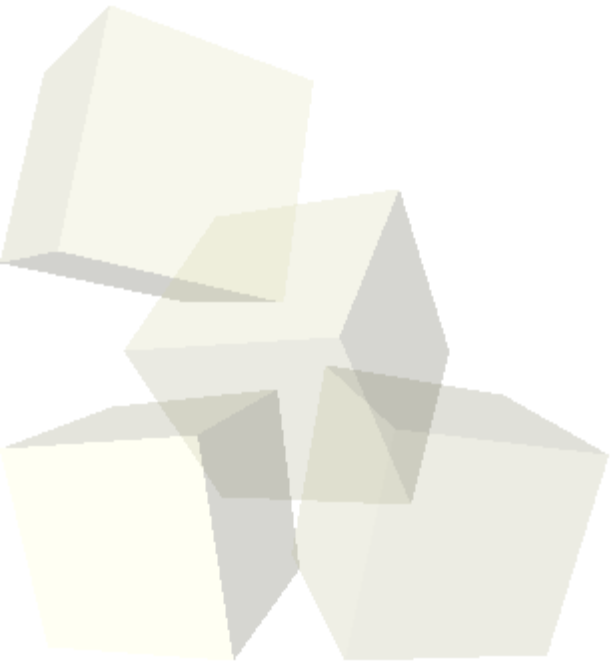
Jython

Python in the JVM

Whitemice Consulting



What is it

- Implements the same language as CPython 2.5
  - Latest release was Jython 2.5.1 on 2009-09-26
- Pure Java (it isn't CPython)
  - Does not, and will not, support CPython modules that include shared libraries (*.so files). Pure Python libraries can be import into the Jython runtime.
    - Of course, Python modules are slower than pure Java assemblies; using Java's Date will be faster than using Python's datetime.
  - co_code not supported.
    - Which might be considered a feature.
- It is Java
  - Uses the JVM GC!
  - Uses the JVM runtime optimizer!

Why

- Dynamic compilation of Java bytecode.
  - Faster than Python
  - Optional static compilation.
- Easy re-use of the massive collection of Java components.
- Easy to use clear syntax of the Python language.

Jython
100% Java
Cross Platform
Mutlthreaded
Advanced GC
Re-use Java components

Python
C
Not Cross Platform *
GIL
Dodgy GC
Wrapped C libraries **

* Isn't Python cross-platform? Copy some non-trivial code from a LINUX host to a Windows server and make it run. Try it. It is so fun!
** Not a fun weekend project, unless you already know C and make and GCC – and desperately need a life.

What doesn't work

- Non-pure Python Python modules
- co_code
  - This might be a feature.
- Multiple inheritence with Java classes
  - You can multi-inherit pure Python classes.
- dir(javaObject)
  - Result may not be complete.
- Access to Java protected static methods.
- The emulated Python “os” module may be limited by Java sandboxing (configuration dependent).
  - This might be a feature.
- Exceptions can be weird
  - Some originate from Jython, other from Java.

What doesn't work

- Python Serialization
  - You cannot pickle/cPickle Java classes
  - Java serialization works.
- Arbitrary attributes on Java classes
  - Python class with no property 'fred'
    - x = pythonClass()
    - x.fred = 'george'
  - Java class with no property 'fred'
    - x = javaClass()
    - x.fred = 'george'
    - Boom!
  - This can be emulated by extending the Java class.
- *Jython is more forgiving about using Python keywords (aka 'print') as methods/attributes.*

What does work

- Python!
  - Straight forward lists and dictionaries.
  - Almost the entire standard library.
- The Python DB API 2.0
  - Courtest of zxJDBC
- There is a port of the Numeric package
  - http://jnumerical.sourceforge.net/index.html
- Compiling scripts to jars for use by Java apps.
- Keyword arguments
- Object attribute access
- Embeedding Jython in Java
  - One of the primary uses of Jython

Can it?

- Run on Google App Engine?
  - Yes
- Host my favorite framework?
  - Django
  - Pylons
  - SQLalchemy
  - TurboGears (w/patches?)
- Run in my application Server?
  - WebLogic
  - WebSphere
  - Tomcat
  - JBoss

Banish getX/setX

- Java code is littered with methods like:
  - javaObject.getName()
  - javaObject.setName(...)
- When Jython sees the code:
  - x = javaObject.name
    - it automatically calls -
  - x = javaObject.getName()
- When Jython sees the code:
  - javaObject.name = 'George
    - it automatically calls -
  - javaObject.setName('George')

Sloppy Type Auto Casting

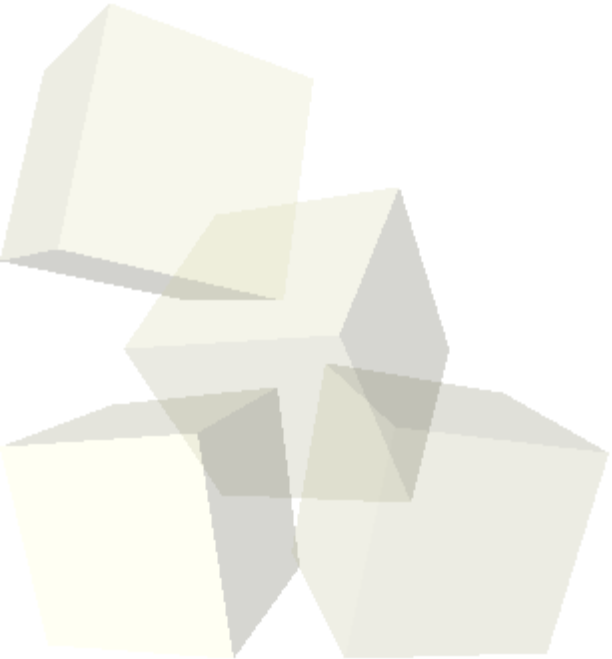
- A difference in type:
  - Java is virtuously statically typed.
  - Python is sloppy typed.
- Jython will, 99.44% of the time, manage your types automatically.
  - So 0.56% of the time it can't.
- For example: [1, 2, 3]
  - Can automatically be cast to an array of integers.
  - If the Java class actually wanted ['1','2','3']
    - → Fail!
  - You can specifically use Java collection classes if you have a tricky type issue.

Supporting Arbitrary Attributes

- If you want/need to make a class more Pythonic...

```
class MyJythonClass(org.Whitemice.MyJavaClass)
        pass
```

- It now supports Pythonic attributes
  - x = MyJythonClass()
  - x.fred = 'george'



Iterating XML

```
import org.jdom as jdom
import java.io as io

output_raw = ""
counter = ""
builder = jdom.input.SAXBuilder()
doc = builder.build(io.FileInputStream("TranslatedDocument.txt"))
rootElement = doc.getRootElement()
for node in rootElement.getChildren("row"):
   output_raw = output_raw + (node.getChild("c01").getTextTrim()) + ","
   output_raw = output_raw + (node.getChild("c02").getTextTrim()) + ","
   output_raw = output_raw + (node.getChild("c03").getTextTrim()) + ","
   output_raw = output_raw + (node.getChild("c04").getTextTrim()) + ","
   output_raw = output_raw + (node.getChild("c05").getTextTrim()) + ","
   if ( (node.getChild("c01").getTextTrim()) == "2" ):
     counter = counter + 1
     output_raw = output_raw + str(counter) + ","
   else:
     counter = 0
```

Jython using Python XML-RPC

```
import xmlrpclib

class Transport(xmlrpclib.Transport):
 .... 34 lines of Python-is-a-scripting-language compensation codel;
       which has nothing to do with this example, it only makes the
       Pythong XML-RPC library have feature parity with the very
       excellent Java librarues....

transport = Transport()
transport.credentials = (_user, _password)
client = xmlrpclib.Server('http://{0}/zidestore/so/{1}/'.format(_host, _user),
                transport=transport)
client.execute('zogi.getObjectById', [10100, 65535])
print 'Got contact {0}'.format(contact['objectId'])
```

XML-RPC using Java Jars

```
import org.apache.xmlrpc.client as xmlrpc
from java.util import Vector, Hashtable
from java.net  import URL

url = "http://%s/zidestore/so/%s/" % ( _host, _username )
client = xmlrpc.XmlRpcClient()
config = xmlrpc.XmlRpcClientConfigImpl()
config.setServerURL(URL(url))
config.basicUserName = _username
config.basicPassword = _password
client.setConfig(config)

params = Vector()
params.add(10100)
params.add(65535);
contact = client.execute('zogi.getObjectById', params)
print 'Got contact {0}'.format(contact['objectId'])
```

```
from com.ziclix.python.sql import zxJDBC

sqlServer = zxJDBC.connect("jdbc:informix-
sqli://192.168.1.60:1526/testdb:Informixserver=BARNET",
                    _username,
                    _password,
                    "com.informix.jdbc.IfxDriver")

outerSQL = 'SELECT DISTINCT xr_stock_no FROM testdb:xrefr ' \
          'WHERE xr_vend_code = ? AND xr_net_price > 0' \
          '  AND (xr_supersede IS NULL OR xr_supersede != \'S\') ' \
          '  AND (xr_stock_no IS NOT NULL)'
outerCursor = sqlServer.cursor()
outerCursor.execute(outerSQL, ['200'])
skus = outerCursor.fetchall()
outerCursor.close()
sqlServer.close()
```

http://seanmcgrath.blogspot.com/JythonWebAppTutorialPart1.html

```
import sys,calendar,time
from java.io import * '
from javax.servlet.http
import HttpServlet from JythonServletUtils import *

class JythonServlet3 (HttpServlet):
        def doGet(self,request,response):
                self.doPost (request,response)

        def doPost(self,request,response):
                toClient = response.getWriter()
                response.setContentType ("text/html")
                toClient.println ("<html><head><title>Servlet Test 3</title>")
                toClient.println ("<body><h1>Calendar</h1><pre>%s</pre></body></html>" %
                        calendar.calendar(time.localtime()[0]))

if __name__ == "__main__":
        JS3 = JythonServlet3()
        dummyRequest = DummyHttpRequest()
        dummyResponse = DummyHttpResponse()
        JS3.doPost (dummyRequest,dummyResponse)
```