Programming in Python

Part #2

IDLE

An *I*ntegrated *D*eve*L*opment *E*nvironment in and for Python

My ideal environment

- Knows more about Python than you can teach Emacs:
    - Perfect colorization
    - Perfect auto-indent
    - Interactive shell with auto-indent
    - Integrated Python debugger
- Is written in portable Python
    - hence extensible in Python

It's not done yet!

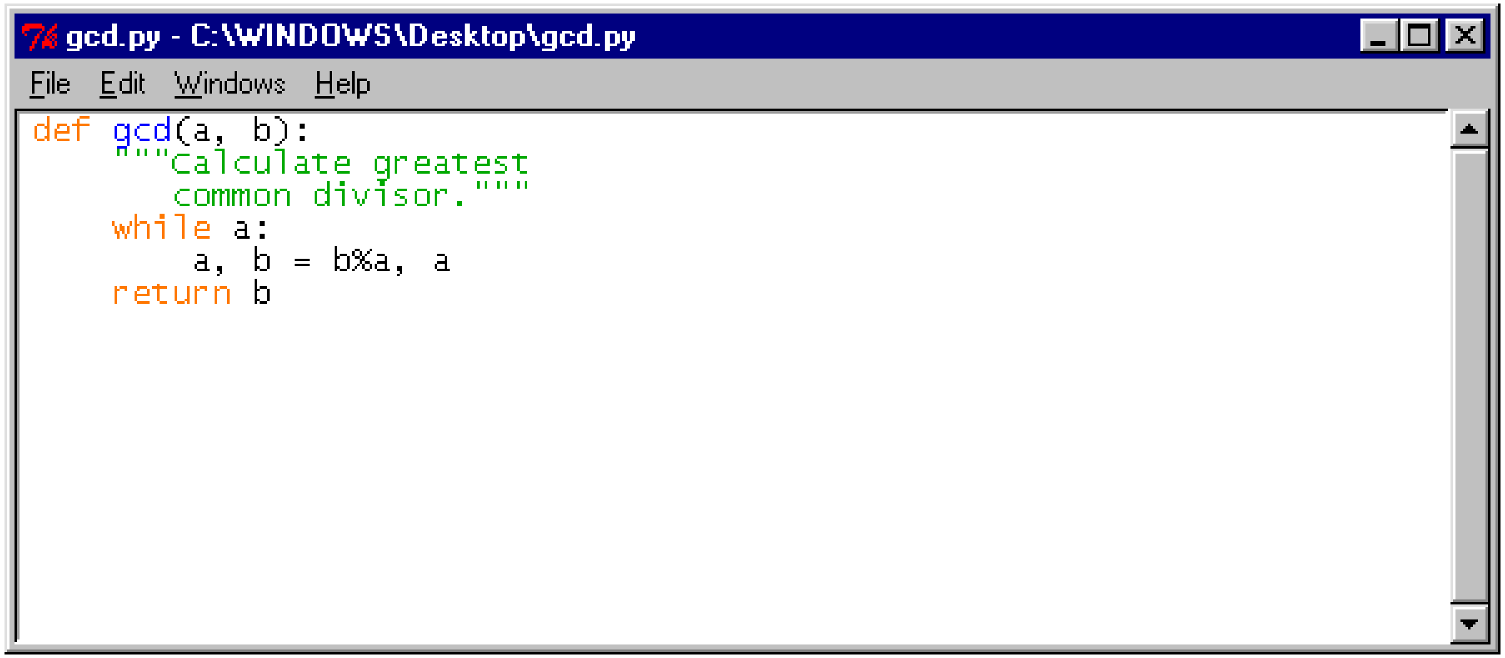
- Debugger unfinished
- Customization
    - beyond editing the source
- Tons of details, e.g.:
    - Emulate more Emacs key bindings
    - Back up files, check if file changed
    - Typing above prompt in shell window
    - Reformat whole buffer

Possible developments

- More newbie-proof
- Project management tools
- Syntax suggestions (templates?)
- Name suggestions (type analysis?)
- Continuous syntax check & lint

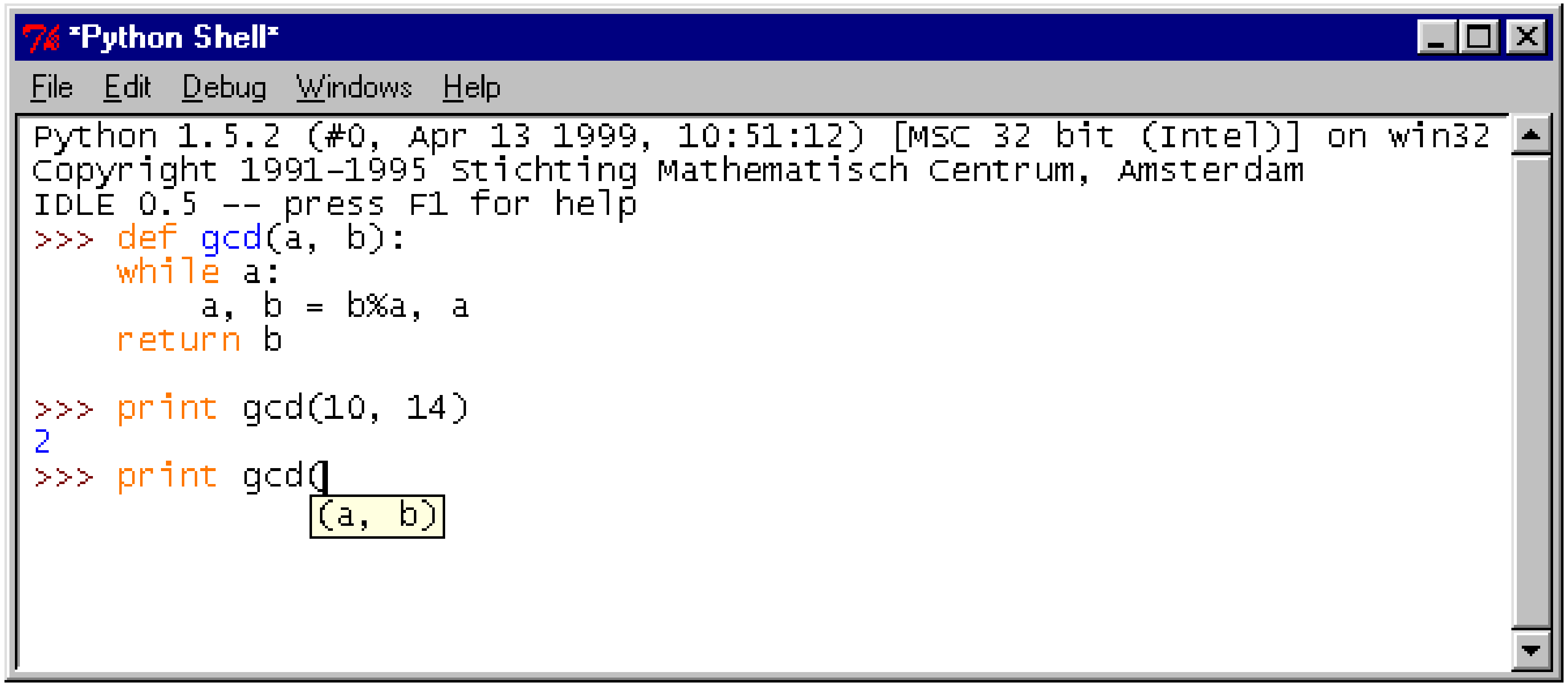
  (like a spell checker)

EditorWindow.py



```
def gcd(a, b):
    """Calculate greatest
       common divisor."""
    while a:
        a, b = b%a, a
    return b
```

PyShell.py



```
Python 1.5.2 (#0, Apr 13 1999, 10:51:12) [MSC 32 bit (Intel)] on win32
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
IDLE 0.5 -- press F1 for help
>>> def gcd(a, b):
        while a:
            a, b = b%a, a
        return b

>>> print gcd(10, 14)
2
>>> print gcd(
              (a, b)
```

The undo engine

- Intercept all buffer-changing operations of widget command (*not* events)
  - rename widget command
  - define new command in its place
    - delegate all ops except insert/delete
- Mechanism allows dynamically stacked interceptors
  - colorization inserted *below* undo

Undo details

- Each command has an inverse
- Some marks also remembered
  - needed by shell (too ad-hoc)
- Grouping option
  - undo/redo several ops at once
  - used by high level cmds e.g. reformat
- Keeps track of "file-changed" flag

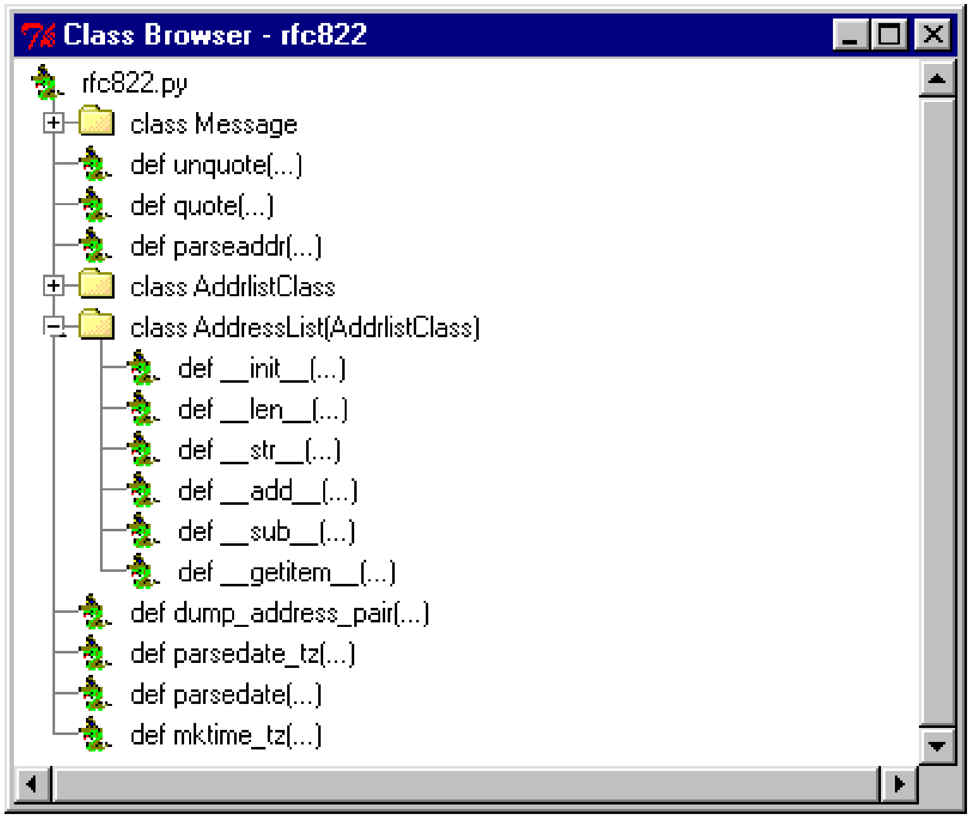
The colorizer

- Insert/delete mark text as dirty
- Colorize while idle, stop on editing
- Perfect Python tokenizer using optimized regular expressions
- >200 lines/sec on 300 MHz P-II

Class browser

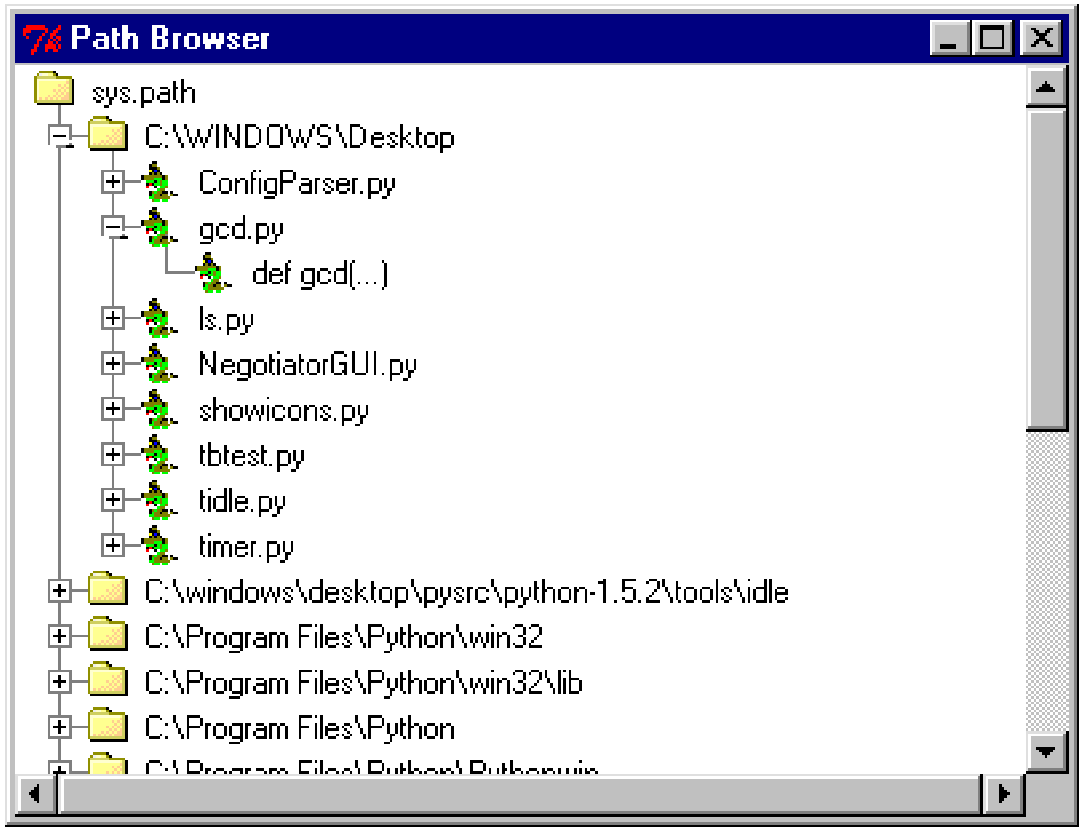
- New Tree widget class
  - uses model/view/controller
  - path browser: change model only
- pyclbr.py parses module source
  - looks for classes and methods
  - modified to find top-level functions

ClassBrowser.py



```
rfc822.py
  class Message
  def unquote(...)
  def quote(...)
  def parseaddr(...)
  class AddrlistClass
  class AddressList(AddrlistClass)
    def __init__(...)
    def __len__(...)
    def __str__(...)
    def __add__(...)
    def __sub__(...)
    def __getitem__(...)
  def dump_address_pair(...)
  def parsedate_tz(...)
  def parsedate(...)
  def mktime_tz(...)
```

PathBrowser.py

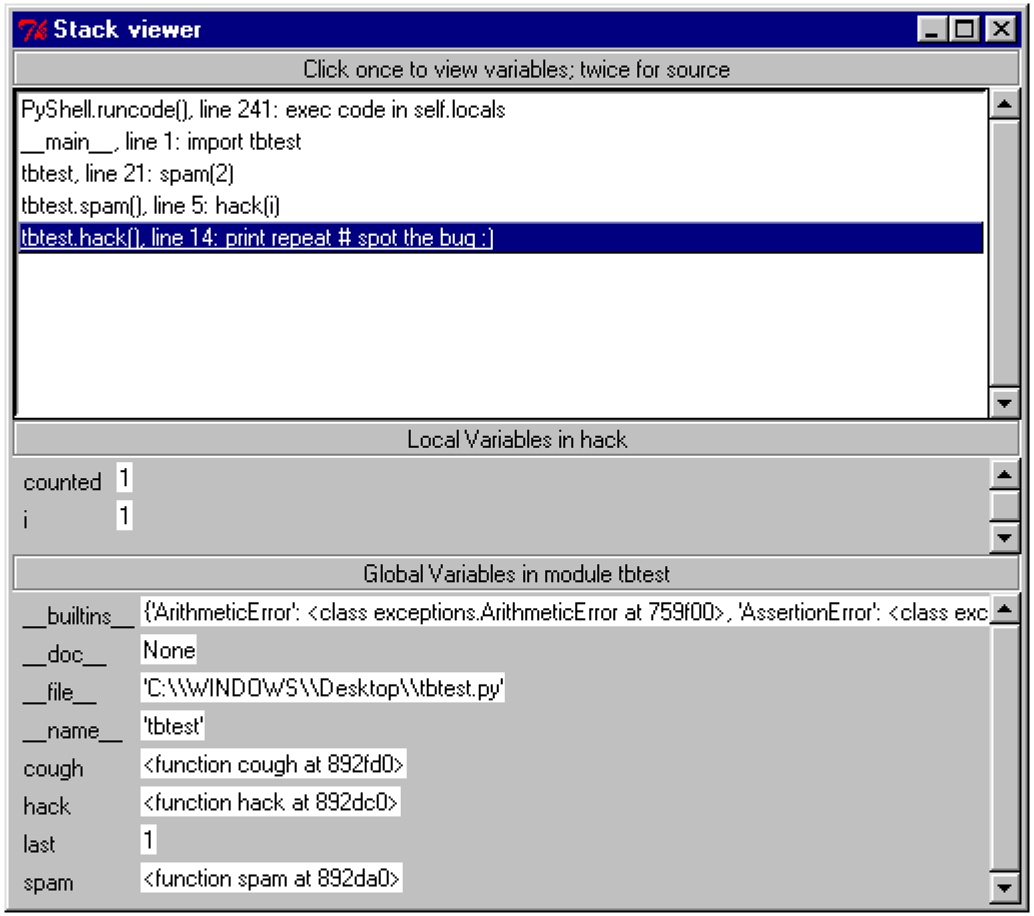


```
sys.path
  C:\WINDOWS\Desktop
    ConfigParser.py
    gcd.py
      def gcd(...)
    ls.py
    NegotiatorGUI.py
    showicons.py
    tbtest.py
    tidle.py
    timer.py
  C:\windows\desktop\pysrc\python-1.5.2\tools\idle
  C:\Program Files\Python\win32
  C:\Program Files\Python\win32\lib
  C:\Program Files\Python
```

Debugger

- Currently sucks :(
    - Go, step, over, out commands
    - Set break points in source window
    - view call stack, locals, globals
        - and source, in editor window
- Stack viewer separately usable
    - post-mortem inspection
- Right-click in stack trace: go to src

StackViewer.py



Click once to view variables; twice for source

```
PyShell.runcode(), line 241: exec code in self.locals
__main__, line 1: import tbtest
tbtest, line 21: spam(2)
tbtest.spam(), line 5: hack(i)
tbtest.hack(), line 14: print repeat # spot the bug :)
```

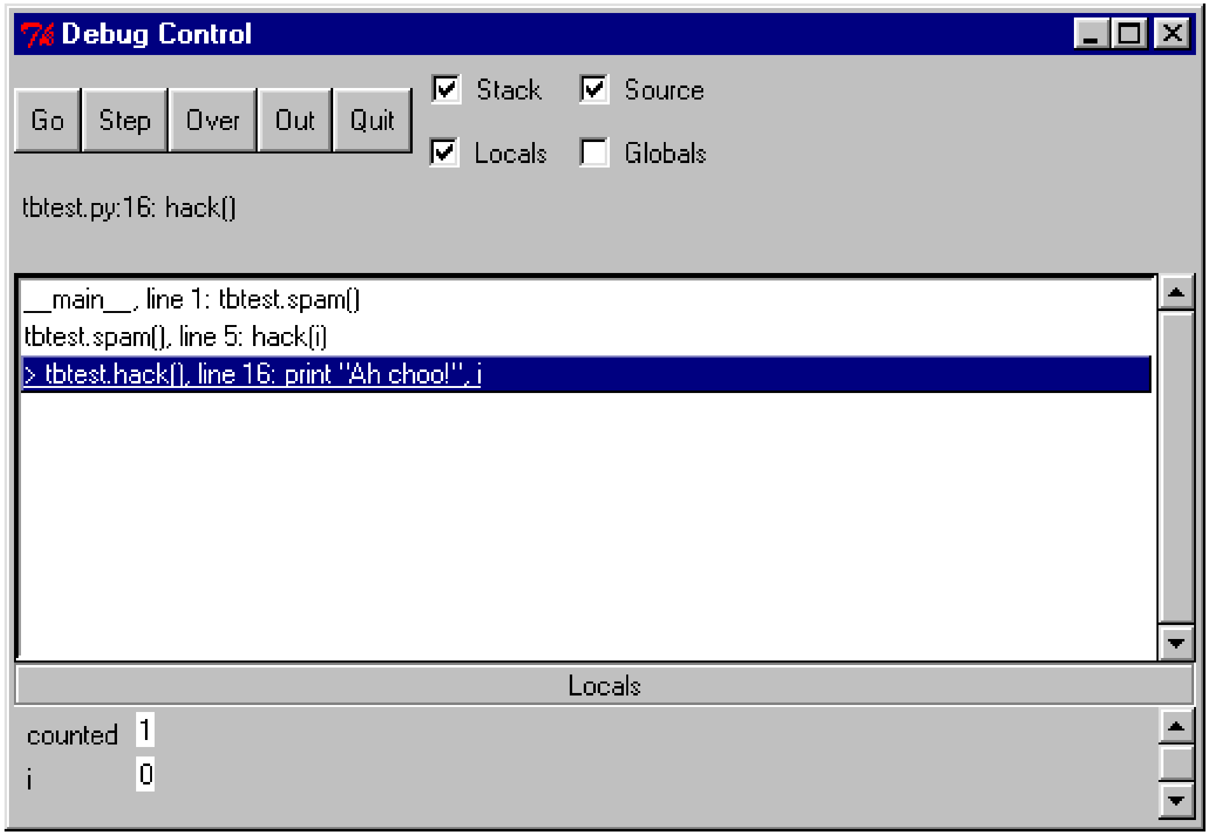
Local Variables in hack

counted	1
i	1

Global Variables in module tbtest

__builtins__	{'ArithmeticError': <class exceptions.ArithmeticError at 759f00>, 'AssertionError': <class exc
__doc__	None
__file__	'C:\\WINDOWS\\Desktop\\tbtest.py'
__name__	'tbtest'
cough	<function cough at 892fd0>
hack	<function hack at 892dc0>
last	1
spam	<function spam at 892da0>

Debugger.py



```
tbtest.py:16: hack()

__main__, line 1: tbtest.spam()
tbtest.spam(), line 5: hack(i)
> tbtest.hack(), line 16: print "Ah choo!", i
```

counted	1
i	0

Extending Python

What is Jython?

- Jython is a **100 % Java version** of the Python scripting language that allows you to compile Python scripts to Java byte code that **runs on any Java Virtual Machine.**
- Jython offers seamless and smooth Java integration: from Python you can **access all Java libraries**, you can build applets or Java beans, you can derive from Java classes in Python and vice versa.
- Like Python, and unlike Java, you can **use Jython interactively**: just type in some Jython code at the prompt and see the story unfold immediately.

Scripting vs. Systems (Hard-Core) Programming / Python vs. Java

- Python does **not** replace Java. Python complements Java and doesn't compete head-on. Python is a scripting language. Java is a hard-core programming language (systems language).
- **No compilation** (Fast Build-Cycle Turnaround)
- **Dynamic Typing** (No Need to Declare Variables For Use)
- **Easy Syntax** (No Curly Braces, No Semicolons, No new, and more)
- **Embeddable** (Scripting Power for Your Apps)
- **Interactive** (Create,View, Change Objects At Run-Time)
- **50 % less code**

Example

- **Java**

```
public class doSomething
 {
 public static void main( String args[] )
   {
     System.out.println( “Do Something" );
   }
 }
```

- **Python**

```
>>> print “Do Something”
```

Python Goodies Missing In Java

- syntactic for lists
- syntactic for maps/dictionaries
- raw strings
- for loop shortcuts (=foreach)
- named method parameters
- string formatting shortcuts

Python Goodies: Syntactic for Lists

- **Java**

```
List list = new LinkedList();
list.add( new Integer( 1 ) );
list.add( new Integer( 2 ) );
list.add( new Integer( 3 ) );
```

- **Python**

```
list = [1, 2] list.append( 3 )
```

Python Goodies: Syntactic for Maps

- **Java**

```
Map map = new HashMap(); map.put( "one", new
    Integer( 1 ) );
map.put( "two", new Integer( 2 ) );
map.put( "three", new Integer( 3 ) );
System.out.println( map.get( "one" ) );
```

- **Python**

```
map = { "one" : 1, "two" : 2, "three" : 3 }
print map[ "one" ]
```

Python Goodies: For Loop Shortcut

- **Java**

```
double sum = 0.0;
for(Iterator it=nums.iterator();it.hasNext() )
  {
     sum += ((Double)it.next()).doubleValue();
   }
```

- **Python**

```
sum = 0.0
for x in nums:
    sum = sum + x
```

Python Goodies: Named Method Parameters

- **Java**

```
JFrame frame = new JFrame( "Server" );
frame.setSize( new Dimension( 200, 200 ) );
frame.setVisible( true );
```

- **Python**

```
frame = JFrame( "Server", visible=1,
    size=(200,200) )
```

Python Goodies: String Formatting Shortcuts

- **Java**

```
double x = 10000.0 / 3.0;
NumberFormat nf = NumberFormat.getNumberInstance();
nf.setMinimumFractionDigits( 2 );
nf.setMaximumFractionDigits( 2 );
String s = nf.format( x );
for( int i = s.length(); i < 10; i++ )
        System.out.print( ' ' );
System.out.print( s );
```

- **Python**

```
x = 10000.0 / 3.0
print "%10.2f" % x
```

Python Goodies: Raw Strings

- Raw String Especially Useful for Regular Expressions
- In Python you can start and end strings with single or double quotes and use the other kind of quote without escaping inside the string (same as in XML)
- In Python you can use triple-quotes (""") strings for multi-line text snippets without escaping new lines.

Python Goodies: Raw Strings

Java	Python
"\\\\$\\\\d+,\\\\d+\\\\."	r'\\$\\d+,\\d+\\.'
"\\\\s((::)(\\\\w+))\\\\b"	r'\\s((::)(\\w+))\\b‘
"c:\\\\sandbox\\\\doc\\\\talk"	r'c:\\sandbox\\doc\\talk'
“Bob says, \\"Python Rocks\\""	‘Bob says, "Python Rocks"

Python Class Example

Java

```
import java.applet.*;
public class Server extends Applet
 {
     public void paint( Graphics g )
          {
               g.drawString( "Server Applet", 10, 10 );
           }
 }
```

Python

```
from java.applet import Applet
class Server( Applet ):
    def paint( self, g ):
                g.drawString( "Server Applet" )
```

Embedding Python in Your App

```
import org.python.util.PythonInterpreter;
import org.python.core.*;

public class SimpleEmbedded
 {
    public static void main( String args[] ) throws
   PyException {
      // create a python interpreter
     PythonInterpreter interp = new PythonInterpreter();
```

Compiling Python Using jythonc

- **jythonc** lets you compile Python scripts to stand-alone Java byte code (that is, .class files).

- Why? You can use Python scripts compiled to plain-vanilla Java classes as applets, Java beans and so on and distribute your frozen Python scripts in jars.

Example

- Create a skeleton class to allow usage of a Python class in a Java GUI (as a java.awt.Component)

```
jythonc Graph.py
```

Python At Work

- **System Utilities** - system admin tools, portable shell scripts
- **Internet Scripting** - CGI scripts, parse HTML, process XML, email tools
- **User Interfaces (UIs)** - rapid prototyping
- **Component Glue** - scripting for apps, COM scripting
- **Distributed Programming** - COM, CORBA, XML-RPC
- **Numeric Programming, Database Programming, Image Processing, Artificial Intelligence, and More**

Python vs. Java Script vs. Basic

Python supports programming in-the-large

- Modules
- Classes
- Exceptions

What is Jelly?

- XML scripting engine turning XML into executable code
- built-in expression language
- easily extensible; lets you plug-in your own tags
- easily embeddable; lets you add Jelly to your own app; no dependencies on servlets or JSP
- open-source; Apache license; official Jakarta project

Jelly Script

```
<j:jelly xmlns:j="jelly:core">
    <j:choose>
        <j:when test="${user.locale}=='de_AT'">
            Server ${user.name}
        </j:when>
        <j:otherwise>
            Hello ${user.name}
        </j:otherwise>
    </j:choose>
</j:jelly>
```

Jelly Script

```
<babelfish:translate from="en"
  to="fr">
      Welcome ${user.name} to Jelly
</babelfish:translate>
```

One Language Can't Do It All

- Scripting on the Rise.
- The Death of General Purpose Languages and Monolithic Applications.
- Most prefer the single-purpose languages to general-purpose languages such as Java, C/C++.

Extending Python

- Python is great for rapid application development
  - Little overhead in creating classes, functions, etc.
  - Can be slow at times, in surprising places
- Python is fairly easy to profile
  - time.clock() module
  - Python Profiler
- It is fairly easy to write slow features in C
  - Write the program in Python
  - Profile
  - Rewrite slow features in C
- Of course, it's never really that easy...

Profiling Python

```
def main():
   print "Hello, World"

import profile
profile.run('main()') #can also sort by time, ncalls, etc.

Hello, World
   3 function calls in 0.050 CPU seconds
Ordered by: standard name
ncalls tottime percall cumtime percall function
     1   0.000   0.000   0.000   0.000  main()
     1   0.000   0.000   0.000   0.000  ?
     1   0.500   0.500   0.500   0.500  profile
```

End of Python

- Next Lecture
  - Chapter 6 Data Types
  - Chapter 7 Expressions and Assignment Structures
  - Chapter 8 Statement-Level Control Structures