Scientific Programming with Python

Randy M. Wadkins, Ph.D.
Asst. Prof. of Chemistry & Biochem.
What is Python?
Python for computers is:

* a scripting language
* a programming language
* interpreted, so it doesn’t crash your computer
This boot camp derived from:

* my own research experience
* syllabus of CHEM 534 (Physical Biochem.)

few, if any, have programming knowledge!
Textbook (supplemental):

- Highly recommended!
- Beautiful introduction to programming
Chapter 1

Computers and Programs

Objectives

- To understand the respective roles of hardware and software in a computing system.
- To learn what computer scientists study and the techniques that they use.
- To understand the basic design of a modern computer.
- To understand the form and function of computer programming languages.
- To begin using the Python programming language.
- To learn about chaotic models and their implications for computing.

1.1 The Universal Machine

Almost everyone has used a computer at one time or another. Perhaps you have played computer games or used a computer to write a paper or balance your checkbook. Computers are used to predict the weather, design airplanes, make movies, run businesses, perform financial transactions, and control factories.

Have you ever stopped to wonder what exactly a computer is? How can one device perform so many different tasks? These basic questions are the starting point for learning about computers and computer programming.

A modern computer can be defined as “a machine that stores and manipulates information under the control of a changeable program.” There are two
Reasons to use Python as a programming language

- very clear, readable syntax
- strong introspection capabilities
- intuitive object orientation
- natural expression of procedural code
- full modularity, supporting hierarchical packages
- exception-based error handling
- very high level dynamic data types
- extensive standard libraries and third party modules for virtually every task
- extensions and modules easily written in C, C++ (or Java for Jython)
- embeddable within applications as a scripting interface
But who cares about that?

* Python is powerful and reasonably fast
* plays well with others (C easily converted to Python code)
* runs everywhere (Mac, Win, Linux, Nokia 60 cell phone!)
* Open source and free (as in beer)!
* Available at www.python.org
Incredible scientific utilities!

- Numeric/NumPy has multidimensional array addressing built in
- ScientificPython works with vectors, creates VRML output, MPI functions...the works!
- MMTK built for calculations on molecules
- PyMol can be used for visualization
6th annual Python convention

March 13-16, 2008

http://us.pycon.org/2008/about/
Two ways to run a Python script:
(i) command line

```bash
[rwadkins:~] randy2% python
ActivePython 2.5.1.1 (ActiveState Software Inc.) based on
Python 2.5.1 (r251:54863, May 1 2007, 17:40:00)
[GCC 4.0.1 (Apple Computer, Inc. build 5250)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```
Two ways to run a Python script:

(ii) as a program

```python
i = 1
sum = 0
while i < 11:
    sum = sum + i
    i = i + 1
print sum
```
Python control flow uses white spaces

```python
i = 1
sum = 0

while i < 11:
    sum = sum + i
    i = i + 1

print sum
```

Indentation indicates loop
from Numeric import *
# from numpy import *
from LinearAlgebra import *
from math import sqrt

m = {
    'C' : 12.0,
    'O' : 16.0
}

k = {
    'k1' : 1.0,
    'c'  : 0.1
}

mol = 'CC'
k = 1.0
c = 0.1
num_atoms = len(mol)

b11 = b22 = (k + c)/12.0
b12 = b21 = -c/12.0

B = array ([b11, b12,b21,b22])
B.shape = (2,2)

print B
print eigenvalues(B)
print eigenvectors(B)
Python can talk directly to the OS

```python
import string
import os
import sys

inFile = sys.argv[1]
inParts = string.split(inFile, ".")
outName = inParts[0] + "1.chg"
os.system("mv %s %s" % (inFile, outName))
```
Python as a go-between: an example in 3D-QSAR

* Quasar does the QSAR
* Output is text files not readable by other software
* Want pretty pictures for journals using molscript and Render-3D
* Python is the workhorse for converting Quasar files to input files for drawing programs
2: Ligand 6167: $\Delta G = -15.650$, $[\text{soln}] = -19.257$, $T\Delta S = -2.100$, $\Delta [\text{int}] = 0.000$ kcal/mol
Quasar output

- extended PDB files
- vertex properties

Python

- normal PDB files
- color information

molscript
**Quasar uses non-standard PDB files**

<table>
<thead>
<tr>
<th>ATOM</th>
<th>VLO</th>
<th>VWL</th>
<th>ID</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>TEMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>934</td>
<td>VL0</td>
<td>VWL</td>
<td>32</td>
<td>-3.65898</td>
<td>8.10650</td>
<td>-8.41721</td>
<td>0.00000</td>
</tr>
<tr>
<td>935</td>
<td>VL0</td>
<td>VWL</td>
<td>33</td>
<td>-4.92248</td>
<td>8.29969</td>
<td>-7.23485</td>
<td>0.00000</td>
</tr>
<tr>
<td>936</td>
<td>VL0</td>
<td>VWL</td>
<td>34</td>
<td>-4.92248</td>
<td>8.29969</td>
<td>-5.51191</td>
<td>0.00000</td>
</tr>
<tr>
<td>937</td>
<td>VL0</td>
<td>VWL</td>
<td>35</td>
<td>-4.69652</td>
<td>10.37606</td>
<td>-8.52262</td>
<td>0.00000</td>
</tr>
<tr>
<td>938</td>
<td>VL0</td>
<td>VWL</td>
<td>36</td>
<td>-4.69591</td>
<td>11.87668</td>
<td>-7.89883</td>
<td>-0.10000</td>
</tr>
<tr>
<td>939</td>
<td>VL0</td>
<td>VWL</td>
<td>37</td>
<td>-6.03914</td>
<td>10.75929</td>
<td>-7.49375</td>
<td>0.00000</td>
</tr>
<tr>
<td>940</td>
<td>VL0</td>
<td>VWL</td>
<td>38</td>
<td>-5.61242</td>
<td>12.02086</td>
<td>-6.53271</td>
<td>0.00000</td>
</tr>
<tr>
<td>941</td>
<td>VL0</td>
<td>VWL</td>
<td>39</td>
<td>-6.15519</td>
<td>9.44225</td>
<td>-6.53271</td>
<td>0.00000</td>
</tr>
<tr>
<td>942</td>
<td>VL0</td>
<td>VWL</td>
<td>40</td>
<td>-5.96978</td>
<td>11.00684</td>
<td>-5.31875</td>
<td>-0.10000</td>
</tr>
<tr>
<td>943</td>
<td>VL0</td>
<td>VWL</td>
<td>41</td>
<td>-5.18417</td>
<td>9.70534</td>
<td>-4.58906</td>
<td>0.00000</td>
</tr>
<tr>
<td>944</td>
<td>VL0</td>
<td>VWL</td>
<td>42</td>
<td>-3.23175</td>
<td>12.99136</td>
<td>-8.50449</td>
<td>0.00000</td>
</tr>
<tr>
<td>945</td>
<td>VL0</td>
<td>VWL</td>
<td>43</td>
<td>-1.81673</td>
<td>13.85631</td>
<td>-7.68694</td>
<td>0.00000</td>
</tr>
<tr>
<td>946</td>
<td>VL0</td>
<td>VWL</td>
<td>44</td>
<td>-3.48914</td>
<td>14.37073</td>
<td>-7.47562</td>
<td>0.00000</td>
</tr>
<tr>
<td>947</td>
<td>VL0</td>
<td>VWL</td>
<td>45</td>
<td>-4.80714</td>
<td>13.36537</td>
<td>-7.24959</td>
<td>0.00000</td>
</tr>
<tr>
<td>948</td>
<td>VL0</td>
<td>VWL</td>
<td>46</td>
<td>-2.22489</td>
<td>14.59241</td>
<td>-6.26166</td>
<td>-0.10000</td>
</tr>
<tr>
<td>949</td>
<td>VL0</td>
<td>VWL</td>
<td>47</td>
<td>-4.17755</td>
<td>14.33352</td>
<td>-6.01044</td>
<td>0.00000</td>
</tr>
<tr>
<td>950</td>
<td>VL0</td>
<td>VWL</td>
<td>48</td>
<td>-4.82516</td>
<td>12.94020</td>
<td>-5.30062</td>
<td>0.00000</td>
</tr>
</tbody>
</table>
chain_old = 0
VWL_flag = 0
while 1:
    inputline = input.readline()
    if not inputline : break
    splitline = string.split(inputline)
    if splitline[0] == 'ATOM':
        name = splitline[0]
        serial = splitline[1]
        atom = splitline[2]
        residue = splitline[3]
        chain = int(splitline[4])
        if chain_old:
            if chain_old <> chain and residue <> 'VWL':
                outFile.write("TER\n")
                chain_old = chain
            elif residue == 'VWL' and not VWL_flag:
                outFile.write("TER\n")
                VWL_flag = 1
            elif residue == 'VWL' and VWL_flag:
                chain = chain_old + 1
        else:
            chain_old = chain
    x = float(splitline[5])
    y = float(splitline[6])
    z = float(splitline[7])

    outFile.write("%-6s%5s  %-4s%-4s %4d    %8.3f%8.3f%8.3f\n" %
                 (name, serial, atom, residue, chain, x, y, z))
### Quasar data stored in its own format

<table>
<thead>
<tr>
<th>Particle</th>
<th>mfp</th>
<th>Distribution:</th>
<th>00000200000</th>
<th>000000000000</th>
<th>000000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle 1: mfp=0</td>
<td>198</td>
<td>00000200000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 2: mfp=0</td>
<td>129</td>
<td>00000071000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 3: mfp=0</td>
<td>166</td>
<td>00200320000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 4: mfp=0</td>
<td>192</td>
<td>00600200000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 5: mfp=0</td>
<td>151</td>
<td>00000301900</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 6: mfp=0</td>
<td>194</td>
<td>00000240000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 7: mfp=0</td>
<td>187</td>
<td>00000130000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 8: mfp=0</td>
<td>199</td>
<td>00000100000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 9: mfp=0</td>
<td>200</td>
<td>00000000000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 10: mfp=5</td>
<td>44</td>
<td>00000152040</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 11: mfp=0</td>
<td>199</td>
<td>00000100000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 12: mfp=0</td>
<td>195</td>
<td>00000500000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 13: mfp=6</td>
<td>85</td>
<td>00000121030</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 14: mfp=0</td>
<td>188</td>
<td>00000840000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 15: mfp=0</td>
<td>114</td>
<td>00000424400</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
<tr>
<td>Particle 16: mfp=0</td>
<td>181</td>
<td>00000172000</td>
<td>000000000000</td>
<td>000000000000</td>
<td></td>
</tr>
</tbody>
</table>
# make a color dictionary

colordict = {   '0': '[128, 128, 128]',  # hydrophobic neutral
    '1': 'red',  # salt bridge, positive
    '2': 'blue',  # salt bridge, negative
    '3': 'green',  # H-bond donor
    '4': 'yellow',  # H-bond acceptor
    '5': 'orange',  # Hydrophobic, positive
    '6': 'redorange',  # Hydrophobic, negative
    '7': 'magenta',  # H-bond flip-flop
    '8': 'cyan'}  # surface solvent

* the next number is the residue number where the particles begin
* this should be fixed later to be an input integer

residue_offset = 17
residue_final = 306

*outFile.write("select %s-%s\nwireframe off\n" % (residue_offset, residue_final))
while 1:
    inputline = input.readline()
    if not inputline: break
    splitline = string.split(inputline)
    mfp = splitline[2]
    value = mfp[4]
    outFile.write("select %s\ncolour %s\n" % (residue_offset, colordict[value]))
    # outFile.write("wireframe off\n")
    if colordict[value] <> '[128, 128, 128]':
        outFile.write("spacefill 0.2\n")
    residue_offset = residue_offset + 1

outFile.close()
Summary of Python

- quick, easy language to program in
- capable of very high or very low level tasks
- math/science libraries for even the most involved calculations
- excellent for working with text files
- downside: slow! But, python can be compiled to develop stand-alone executables.
More Resources!
The goal of this course is to teach students of the physical sciences and mathematics how to use various computational tools to accomplish tasks common to a variety of disciplines. The target student is a physical science (Physics, Chemistry, Biology, Geology,..) or Mathematics major who has completed at least 2 semesters of calculus and is comfortable using computers, although no formal programming experience is assumed.

The primary tool we will use will be the Python programming language which is fairly easy to learn yet has all the characteristics of a modern programming language such as object orientation, wide and open source code base, full featured graphical user interface (GUI) tools, and easy cross platform compatibility (Linux, Windows, and Mac). With the exception of some work in Mathematica, all tools will be open source (free) and cross platform to facilitate students ability to work outside of the computer lab.

The syllabus for such a course is still being developed, and the current version can be downloaded here. I expect there to be some flexibility based on the interests and background of the students enrolled.

The basic topics of the course will be:

* Introduction to computer programming with Python
* Graphical representation of data
* Linear and non-linear regression methods and interpolation
* Numerical differentiation
* Numerical integration
* Numerical precision and error issues
* Roots of functions and solutions of systems of equations
* A tour of capabilities of Mathematica

If you have questions or would like more information about the status of the course, please email me at jgladden@olemiss.edu.
An introduction to Python for scientists

Six lectures and exercises.

- **Part 1: Python as a calculator** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Solutions to the exercises for part 1** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Part 2: Dealing with text files** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Solutions to the exercises for part 2** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Part 3: Advanced calculations** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Solutions to the exercises for part 3** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Part 4: Higher mathematics** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Solutions to the exercises for part 4** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Part 5: Make your own data types** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Solutions to the exercises for part 5** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Part 6: Graphical user interfaces** — by Konrad Hinsen — last modified 2007-01-23 15:20
- **Solutions to the exercises for part 6** — by Konrad Hinsen — last modified 2007-01-23 15:20
Python Scripting for Computational Science

Second Edition
How to Think Like a Computer Scientist
Learning with Python 2nd Edition

by Jeffrey Elkner, Allen B. Downey and Chris Meyers

Table of Contents

- Foreword
- Preface
- Contributor List
- Chapter 1: The way of the program
- Chapter 2: Variables, expressions, and statements
- Chapter 3: Functions
- Chapter 4: Conditionals
- Chapter 5: Fruitful functions
- Chapter 6: Iteration
- Chapter 7: Case Study
- Appendix A: GASP
- Appendix B: Debugging
- Appendix C: Creating a new datatype
- Appendix D: Doctests and test driven development
- Appendix E: Complete Python listings
- Appendix F: Recommendations for further reading
- GNU Free Document License
- Index

download a gzipped tar file
of the Libre source and HTML [here](http://www.ibiblio.org/obp/thinkCSpy/)
Acknowledgements

- Phil Potter, Ph.D., St. Jude
  - M. Wierdl
  - C. L. Morton
  - J. K. Weeks
  - L. Oliver
  - J. Hyatt
- Dr. Mary K. Danks, St. Jude
- NIH grant CA108755
- NSF EPSCoR grant EPS-0556308