PYTHON FOR STRUCTURAL BIOINFORMATICS

Sophie COON & Michel SANNER

MGL laboratory
The Scripps Research Institute, La Jolla, CA

SCHEDULE

I- Introduction 15 minutes
  • Background and motivation
  • Brief Python overview

II- PMV 20 minutes
  • Fundamental concepts, main commands …

III- PMV BUILDING BLOCS 30 minutes
  • MolKit, DejaVu, ViewerFramework

BREAK 15 minutes
SCHEDULE

IV- Extending PMV  
• Scripting capabilities in PMV  
  † scripts  
  † macros  
  † commands  
• Visual Programming Environment in PMV  
V- Conclusion  
Questions

I- INTRODUCTION

❖ Background and motivation  
  • Laboratory  
  • Challenge  
  • Software development strategy  
❖ Introduction to Python
The Challenge

Visualization

Electrostatics Calculations

Molecular Surfaces

Modeling

Ab Initio Methods

Protein Engineering

Folding

Docking Methods

Sequence Analysis

MM - MD

Etc ...
"Traditional" solution

- Electrostatics Calculations
- Molecular Surfaces
- Modeling
- Ab Initio Methods
- MM - MD

Visualization

Protein Engineering

Docking Methods

Folding

Sequence Analysis

N^2 interfaces

low interoperability

No code Reuse

Our solution

High level language as a scripting environment

- Interactive
- Dynamic
- Platform independent
Writing an application

High level coding
Code re-use
Extensible

Why Python?

<table>
<thead>
<tr>
<th>features</th>
<th>Not met by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object oriented</td>
<td>Tcl, Perl, C, ..</td>
</tr>
<tr>
<td>Advanced data structures</td>
<td>Tcl, Perl</td>
</tr>
<tr>
<td>Powerful data-parallel arrays</td>
<td>Tcl</td>
</tr>
<tr>
<td>Readability and modularity</td>
<td>Perl</td>
</tr>
<tr>
<td>High level</td>
<td>C, C++, Fortran</td>
</tr>
<tr>
<td>Platform independence</td>
<td>C, C++, Java, ...</td>
</tr>
<tr>
<td>Interpreted</td>
<td>C, C++, Java</td>
</tr>
</tbody>
</table>
I- INTRODUCTION

❖ Background and motivation
❖ Python overview
   • Language characteristics
   • Basics types
   • Control flow
   • Functions
   • More...

Language characteristics

❖ Interpreted, high level, object oriented
❖ Flexible and extensible
❖ Introspection and self-documentation
❖ Platform independent
❖ Open source
❖ Rapidly gaining acceptance
❖ Access to C, C++ fortran code…
Data types and data structures

- basic data types:
  ```python
  >>> a=2
  int
  >>> b=7.3
  float
  >>> s="name"
  >>> s='name'
  string
  No memory allocation and no variable declaration.
  ```

- data structures:
  ```python
  >>> l = [1,2,5]
  list: mutable sequence
  >>> l[0]
  1
  >>> l[2] = 6
  >>> l[1:2]
  [6,5]
  >>> t = (1,2,5)
  Tuple immutable sequence
  >>> t[-1]
  5
  >>> t[3] = 6
  >>> t[:2]
  (1,2)
  >>> t = {'so':23}
  Dictionary associative arrays
  >>> t['so']
  23
  >>> t['new'] = '45'
  >>> t.items()
  [('so',23), ('new', '45')]
  ```

Control Flow

- While:
  ```python
  >>> b = 0
  >>> while b < 5:
  ...    print b; b=b+1
  0 1 2 3 4
  ```

- If:
  ```python
  >>> q = 'Please Enter a number'
  >>> x = int( raw_input (q) )
  >>> if x == 0:
  ...    print 'x equals 0'
  ...    elif x < 0:
  ...      print 'x is negative'
  ...    else:
  ...      print 'x is positive'
  ```
Function and arguments

Positional arguments (required)

```python
def func( a, b, n1=10, n2='hello' ):
```

Named arguments (optional)

Function name

Argument matching:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>n1</th>
<th>n2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>'string'</td>
<td>3.14</td>
<td>'hello'</td>
</tr>
<tr>
<td>7.2</td>
<td>'string'</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>'hello'</td>
<td>2</td>
<td>5</td>
<td>'bye'</td>
</tr>
</tbody>
</table>

ERROR: missing argument

ERROR: positional argument after named argument

More ...

- Classes
- Exceptions
- Packages
- etc...
- Tutorial:
  - http://www.python.org/doc
II- PMV:

- Fundamentals
- Basic commands
- More advanced manipulations

PMV: Python-based Molecule Viewer
III- PMV BUILDING BLOCS

- **MolKit**
  - Hierarchical data structures
  - Derived classes
  - Parsers
  - Examples

- **DejaVu**

- **ViewerFramework**

Hierarchical data-structure

```
TreeNode
  .parent
  .top
  .children
  .elementType
  .adopt(child)
...
```

```
TreeNode
  .top
  .elementType

TreeNodeSet(ListSet)
  [TreeNode, TreeNode, ...]
```

```
MolKit
  PyBabel
  Numeric
```
**TreeNode and TreeNodeSet specialization**

TreeNode Specialization

```
TreeNode
  ├ Molecule ── Atom ── SecondaryStructure ── ...
  │         ├── Residue ── Chain ── Protein ── ...
  │         │            └ Helix ── Strand ── Turn ── Coil
```

TreeNodeSet Specialization:

```
TreeNodeSet
  ├ MoleculeSet ── AtomSet ── SecondaryStructureSet ── ...
  │         └ ResidueSet ── ChainSet ── ProteinSet ── ...
  │         └ HelixSet ── StrandSet ── TurnSet ── CoilSet
```

**Parsers**

```
from MolKit.pdbParser import PdbParser
parser = PdbParser(’/1crn.pdb’)
mols = parser.parse()
```
Example

# Importing the Read function from the MolKit package.
>>> from MolKit import Read
# Read calls the proper parser depending on the file extension
>>> mols = Read("./1crn.pdb")
>>> mol = mols[0]
# print the name of all the residues of all the chains of mol
>>> print mol.chains.residues.name
# call the full_name method of the atoms 20 to 85 and print the
# result
>>> print mol.chains.residues.atoms[28:85].full_name()

Example

# import the Atom class located in the molecule module in the
# MolKit package
>>> from MolKit.molecule import Atom
# find in the 'mol' tree all the nodes of type Atom.
>>> allAtoms = mol.findType(Atom)
# get the subset of these atoms for which the value of the
# temperatureFactor attribute is strictly superior to 20.
>>> set1 = allAtoms.get(lambda x: x.temperatureFactor > 20)
Example

# get the parent of the atoms and get rid off the duplicates
>>> allResidues = allAtoms.parent.uniq()

or

>>> allResidues = allAtoms.residues.uniq()

# import the Numeric extension
>>> import Numeric

# Compute the geomCenter of each residue and store it in
# a new attribute called geomCenter.
>>> for r in allResidues:
. . . coords = r.atoms.coords
. . . r.geomCenter = Numeric.sum(coords)/len(coords)

III- PMV BUILDING BLOCS

☞ MolKit
☞ DejaVu:
  • Overview
  • Features
  • DejaVu and MolKit
☞ ViewerFramework
Overview

from DejaVu import Viewer
vi = Viewer()
from DejaVu.Spheres import Spheres
centers = [[0,0,0],[3,0,0],[0,3,0]]
s = Spheres('sph', centers = centers)
s.Set(quality=10)
vi.AddObject(s)

Demo Code

>>> from DejaVu import Viewer
>>> vi = Viewer()

>>> from DejaVu.Spheres import Spheres
>>> centers = [[0,0,0],[3,0,0],[0,3,0]]
>>> s = Spheres('sph', centers = centers)
>>> s.Set(quality=10)

>>> vi.AddObject(s)
Features

- OpenGL Lighting and Material model
- Object hierarchy with transformation and rendering properties inheritance
- Arbitrary clipping planes
- Material editor
- DepthCueing (fog), global anti-aliasing
- glScissors/magic lens
- Multi-level picking
- Extensible set of geometries

DejaVu and MolKit
Demo Code

```python
>>> from MolKit import Read
>>> molecules = Read('/tsri/pdb/struct/1crn.pdb')
>>> mol = molecules[0]  # Read returns a ProteinSet

>>> coords = allAtoms.coords
>>> radii = allAtoms.radius

>>> sph = Spheres('sph', centers = coords, radii = radii, quality=10)
>>> vi.AddObject(sph)
```

III - PMV BUILDING BLOCS

- **MolKit**
- **DejaVu**
- **ViewerFramework**
  - Overview
  - Design features
  - Putting it all together
Overview

Design features

- Dynamic loading of commands
- Python shell for scripting
- Dual interaction mode (GUI/Shell)
- Support for command:
  - development, logging, GUI, dependencies
- Lightweight commands: Macros
- Dynamic commands (introspection)
- Extensible set of commands
Putting it all together

PMV

Msms
Commands

ViewerFramework

DejaVu

MolKit
Idle

Numeric
PyOpenGL
Tkinter

Python Interpreter

BREAK
IV- EXTENDING PMV

- Writing a resource file
- Writing a python script
- Writing a Macro
- Writing a PMV Commands
- Visual Programming in Pmv

Writing a resource file

- Execute a set of PMV commands
  - read a molecule (protease.pdb)
  - load and compute msms surface
  - color by atomType
- Open myrc.py in an editor
- Copy and paste the command log string
- In a new PMV session:
  - File -> source -> myrc.py
Write a distance script

Goal:

• color the molecular surface of the protease by the smallest distance to all atoms in the ligand.

What do we need:

• compute this distance and store the value in a new atom attribute. Use this new property to color the molecular surface of the protease

```python
import Numeric
# Get a handle on the protease atoms
proteaseAtoms = self.Mols[0].allAtoms
# Get a handle on the ligand atoms
ligandAtoms = self.Mols[1].allAtoms
def distanceToClosestPoint(point, setOfPoints):
    """computes the shortest distance between 'point' and 'setOfPoints'"""
    diff = Numeric.array(point) - Numeric.array(setOfPoints)
    diff = diff*diff
    len = Numeric.sqrt(Numeric.sum(diff,1))
    return min(len)
```

**closestDistance.py**

```python
# Get a handle on the coordinates of all the atoms of the ligand
ligandAtomsCoords = ligandAtoms.coords
# Call the distanceToClosestPoint for each atom of the protease
# and store the result in a new atom attribute called “closest”
for a in proteaseAtoms:
    a.closest = distanceToClosestPoint(a.coords, ligandAtomsCoords)
```

---

**In PMV Color msms by distance.**

- File -> ReadMolecule protease.pdb
- File -> ReadMolecule indinavir.pdb
- File -> source -> script -> distance.py
- Select -> Select From String -> protease
- Compute -> MSMS for Mol
- Color -> By Property -> Atom, ‘closest’, RGB
Writing the distanceMac.py macro

```python
import Numeric
# Implement a class
def ClosestAtom():
    """ask for 2 molecules, then compute for each atom
    the distance to the closest atom in the other
    molecule. After execution, each atom has a new
    attribute 'closest' holding the distance"
    from Pmv.guiTools import MoleculeChooser
    # Create a gui to let the user choose the two molecules
    p = MoleculeChooser(self, 'extended', 'Choose 2 molecules')
    mols = p.go(modal=1)
    if len(mols) != 2:
        print "ERROR: two molecules need to be selected"
        return
    mol1Atoms = mols[0].allAtoms
    mol2Atoms = mols[1].allAtoms

    def distanceToClosestPoint(point, setOfPoints):
        """computes the shortest distance between point and setOfPoints"
        diff = Numeric.array(point) - Numeric.array(setOfPoints)
        diff = diff * diff
        len = Numeric.sqrt(Numeric.sum(diff, 1))
        return min(len)

    mol2AtomsCoords = mol2Atoms.coords
    for a in mol1Atoms:
        a.closest = distanceToClosestPoint(a.coords, mol2AtomsCoords)

    mol1AtomsCoords = mol1Atoms.coords
    for a in mol2Atoms:
        a.closest = distanceToClosestPoint(a.coords, mol1AtomsCoords)
```

February 3-6 2003  BioCon 2003 - San Diego, CA - 43

Writing the distanceMac.py macro

```python
def distanceToClosestPoint(point, setOfPoints):
    """computes the shortest distance between point and setOfPoints"
    diff = Numeric.array(point) - Numeric.array(setOfPoints)
    diff = diff * diff
    len = Numeric.sqrt(Numeric.sum(diff, 1))
    return min(len)

mol2AtomsCoords = mol2Atoms.coords
for a in mol1Atoms:
    a.closest = distanceToClosestPoint(a.coords, mol2AtomsCoords)

mol1AtomsCoords = mol1Atoms.coords
for a in mol2Atoms:
    a.closest = distanceToClosestPoint(a.coords, mol1AtomsCoords)
```

February 3-6 2003  BioCon 2003 - San Diego, CA - 44
Loading the macro in PMV

File -> loadMacro -> Open Macro Library
distanceMac.py
Macros -> distanceMac -> closestDistance
protease, ligand

Command overview

```python
class MyCommand(MVCommand):
    def __call__:
        # Hook to call the command from pyShell
    def doit:
        # implements what the command does
    def guiCallback:
        # Hook to get user inputs
    def setUndo:
        # Hook to have an undoable command
    def checkDependencies:
        # Where command’s dependencies are checked
    def onAddObjectToViewer:
        # Where new molecules geometries are created
    Etc ...
```
IV- EXTENDING PMV

Visual Programming

- overview
- viperCommands
- radii of cpk by properties
- color msms of ligand by distance to the protease.

Overview

Enabling scientists (non-programmers) to build computational networks
Overview

**Computational Flow:** no data duplication

**Dynamic:** on-the-fly node editing

**High-level:** node programmed using Python

**Portable:** SunOS, IRIX, OSF, Linux, windows

**Scriptable:** Python interpreter

**Component based:** NetworkEditor, DejaVu, MolKit,...

A User Interface NOT an Environment

---

ViPER architecture

[Diagram of ViPER architecture]

1. ViPER
2. NetworkEditor
3. Python Interpreter
4. Web
5. Standard
6. SymServer
7. Imaging
8. PIL
9. MolKit
10. Mslib
11. Your Library
12. Your Code
13. Visualization
14. DejaVu
15. Tkinter
16. Numeric
17. PyOpenGL
Example 1: Goal

GOAL:
Modulate the radii of the CPK by an atom property

Example 1: Instructions

1- Start pmv
2- Read molecule, display by CPK and color by Atom Type
   File -> ReadMolecule -> 1crn.pdb
   Un/display -> by CPK -> OK
   Color -> by AtomType ->"cpk" ->OK
3- start VPE:
   buttonbar -> VPE
   drag the needed nodes on the canvas
   create and run the network
### Example 1: Nodes location

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Category Name</th>
<th>Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pmv</td>
<td>PMV</td>
<td>Pmv viewer</td>
</tr>
<tr>
<td>3D Visualization</td>
<td>Filter</td>
<td>Choose Geom</td>
</tr>
<tr>
<td>Standard</td>
<td>Python</td>
<td>Call Method</td>
</tr>
<tr>
<td>Standard</td>
<td>Mapper</td>
<td>array Ufunc2</td>
</tr>
<tr>
<td>Pmv</td>
<td>Molecules</td>
<td>1crn</td>
</tr>
<tr>
<td>MolKit</td>
<td>Filter</td>
<td>select Nodes</td>
</tr>
<tr>
<td>MolKit</td>
<td>Input</td>
<td>Extract Atom Property</td>
</tr>
<tr>
<td>Standard</td>
<td>Input</td>
<td>Dial</td>
</tr>
<tr>
<td>3D Visualization</td>
<td>output</td>
<td>Redraw</td>
</tr>
</tbody>
</table>

### Example 1: Create the network

![Diagram of network creation process](image)
Example 2: Goal

Goal:
Color the molecular surface of the protease by the closest distance to the ligand

Example 2: Instructions

1- Start pmv
2- Read protease and indinavir
   File -> ReadMolecule -> protease.pdb
   File -> ReadMolecule -> indinavir.pdb
   selectFromString -> "protease"
   compute -> MSMSMol
3- start VPE:
   buttonbar -> VPE
   drag the needed nodes on the canvas
   create and run the network
4- Color the surface with the new atom property
   color -> By Property -> msms -> Atom, closest
### Example 1: Nodes location

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Category Name</th>
<th>Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pmv</td>
<td>Molecule</td>
<td>protease</td>
</tr>
<tr>
<td>Pmv</td>
<td>Molecule</td>
<td>indinavir</td>
</tr>
<tr>
<td>Standard</td>
<td>Python</td>
<td>getattr</td>
</tr>
<tr>
<td>Standard</td>
<td>Python</td>
<td>getattr</td>
</tr>
<tr>
<td>Standard</td>
<td>Python</td>
<td>iterate</td>
</tr>
<tr>
<td>Standard</td>
<td>Python</td>
<td>iterate</td>
</tr>
<tr>
<td>Standard</td>
<td>Python</td>
<td>setattr</td>
</tr>
<tr>
<td>Standard</td>
<td>builtin</td>
<td>min</td>
</tr>
<tr>
<td>Symserver</td>
<td>Mapper</td>
<td>distanceToPoint</td>
</tr>
</tbody>
</table>

---

### Example 1: Create the network

![Diagram](image-url)
V- CONCLUSION

- Validity of the approach
- Python
- Availability and support

Validity of the approach

- Set of components
  - extensible
  - inter-operable
  - re-usable
  - short development cycle
- User base expanding beyond our lab.
- Components re-use outside the field of structural biology
Python

- Appropriate language for this approach
  - modularity, extensibility, dynamic loading, object-oriented, virtually on any platform, many extensions from third party
- Rapidly growing community of programmers using Python for biological applications
- Shortcomings
  - reference counting, distribution mechanism, no strong typing

Availability and support

- Online Download site:
  http://www.scripps.edu/~sanner/python
- pmv mailing list
  email: majordomo@scripps.edu
  subject: subscribe pmv
- Reporting bugs and asking for features
  http://mgldev.scripps.edu/bugs/index.html
Acknowledgments

Christian Carrillo, Kevin Chan
Ruth Huey, Daniel Stoffler
Vincenzo Tchinke, Greg Paris
MGL at TSRI
Pat Walters, Matt Stahl
Don Bashford
Guido van Rossum & Python community

Useful links

Python website:
  • http://www.python.org

MGL web site:
  • http://www.scripps.edu/pub/olson-web/index.html

MGLTOOLS web site:
  • http://www.scripps.edu/~sanner/index.html