


 PYTHON FOR STRUCTURAL
BIOINFORMATICS


 Sophie COON
and
Michel SANNER 

The Scripps Research Institute
La Jolla, California

 SCHEDULE

- I - Fundamentals
Code development strategies, Python
- II - PMV
Fundamentals, main commands
- III - From building blocks to applications
MolKit, DejaVu ViewerFramework, ...
Putting it all together
Writing a simple command
- Conclusion

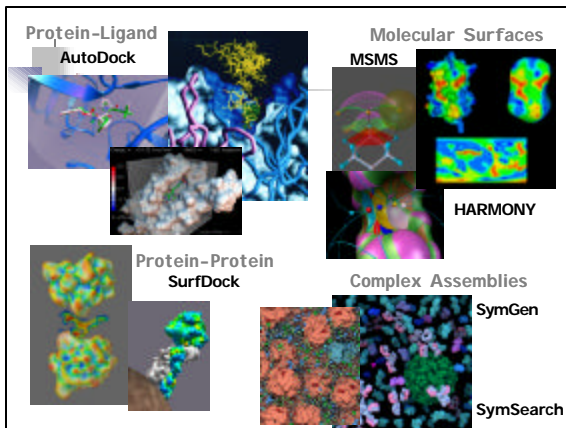
*(1 mistake bloc -> block)

 I - Fundamentals

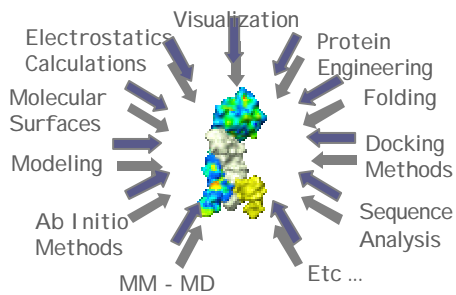
- Code development strategies
- Python primer

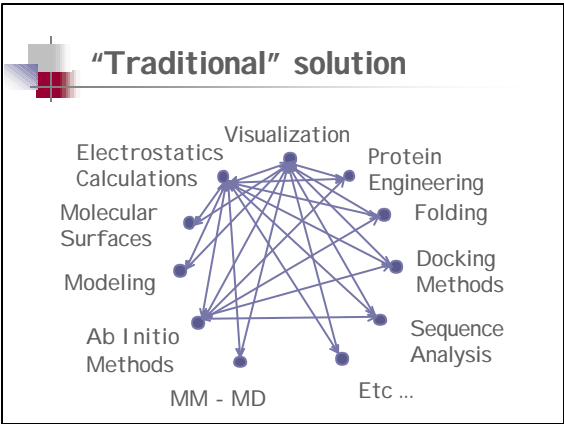
I - Fundamentals

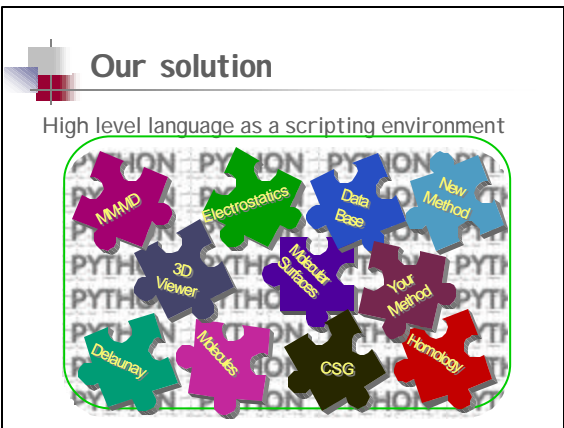
- Code development strategies
 - The challenge
 - Traditional solution
 - Our solution

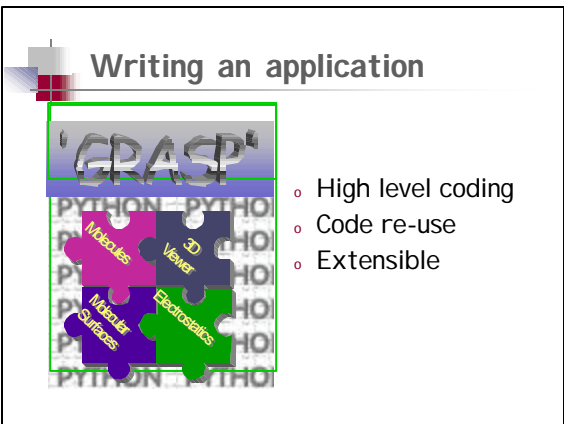


The challenge









Why Python ?

Our language needed :	Not met by :
◦ Object-Oriented	Tcl, Perl, C, ...
◦ Advanced data structures	Tcl, Perl
◦ Powerful data-parallel arrays	Tcl
◦ Readability and modularity	Perl
◦ High-level	C, C++, Fortran
◦ Platform independence	C, C++, Java, ...
◦ Interpreted	C, C++, Java, ...

* (add interpreted, rearrange the slide)

Python based molecular software

- Chimera : UCSF - Computer Graphics Lab.
<http://www.cgl.ucsf.edu/chimera>
- MMTK : CNRS - Institut de Biologie Structurale
<http://starship.python.net/crew/hinsen/mmtk.html>
- Pymol : Delano Scientific.
<http://www.pymol.org>
- PyDayLight : Daylight Chemical Information Systems, Inc.
<http://starship.python.net/crew/dalke/PyDaylight/>
- MDTools : UIUC - Theoretical Biophysics Group.
<http://www.ks.uiuc.edu/~jim/mdtools/>
- ...

* Add "molecular" in the title

I - Fundamentals

- Python primer
 - Language characteristics
 - Basics
 - Standard library
 - Extending Python
 - Numeric extension

Language characteristics

- Interpreted, high level, object-oriented
- Flexible and extensible
- Introspection, self-documenting
- Platform independent
- Open-source
- Rapidly gaining acceptance

Cool!

Basics

```

>>> a = 2          # integer
>>> b = 7.5        # float
>>> c = 'hello'    # string
>>> d = " World!" # string too
>>> # this is a comment
>>> print c+d
>>> print "sum :", a+b
  
```

- No Memory allocation
- No variable declaration

simple and intuitive

List - mutable sequences

```

>>> lst = [2, 1, 'hello']
lst contains:
>>> lst          ▶ [2, 1, 'hello']
>>> lst.append(5) ▶ [2, 1, 'hello', 5]
>>> lst.remove(1) ▶ [2, 'hello', 5]
>>> lst.insert(1, 'Paul') ▶ [2, 'Paul', 'hello', 5]
>>> lst.sort()      ▶ [2, 5, 'Paul', 'hello']
  
```

* lst contains instead of list contains

Tuple - immutable sequences

tup contains:

```

>>> tup = (1,2,'spam')
# automatic packing and unpacking
>>> tup = 1,2, 'spam'
>>> a,b,c = tup
>>> print a, b, c
# correct singleton syntax
>>> tup = (1,)
>>> tup[0] = 6 # ERROR : cannot assign value in a immutable sequence !

```

* Add 'tup contains' for consistency

Sequences indexing and slicing

```

>>> lst = [1,'b',6,'a',8,'e']

```

- Indexing : `lst[i]`

```

>>> lst[4]
>>> lst[-2]

```

- Slicing : `lst[from:to]`

```

>>> lst[2:4]
>>> lst[2:-2]
>>> lst[-4:-2]

```

* slicing instead of scling

Dictionary

```

dict = { key1 : value1, key2 : value2, ...}

```

item1

```

>>> dict = { 'Marc':25, 30 : 'Jim' }
>>> dict[5.7] = 'joe'
# Accessing the information
>>> dict.items()
>>> dict.values()
>>> dict.keys()
>>> dict.has_key('Marc')

```

* Corrected mistakes in dict values() and dict keys()

Control flow

- While :


```
>>> b = 0
>>> while b < 5:
...     print b
1 2 3 4
```
- If :


```
>>> 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'
```
- For, range(), break, continue :


```
>>> seq = range(-3, 4, 1)
>>> print seq
[-3,-2,-1,0,1,2,3]
>>> for s in seq:
...     if s < 0:
...         continue
...     else:
...         print s
0,1,2,3
```

Functions and arguments

Positional arguments (required)

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

Function name Named arguments (optional)

Argument matching:

	a	b	n1	n2
>>> func(2, 'string', 3.14)	2	'string'	3.14	'hello'
>>> func(7.2, 'string', n2=15)	7.2	'string'	10	15
>>> func('hello', 2, 5, 'bye')	'hello'	2	5	'bye'
>>> func(n1 = 5)	ERROR: missing argument			
>>> func(n1=12, 3.14)	ERROR: positional argument after named argument			

* func(n1=5) instead of func(=5)

Functions - Arbitrary arguments

```
def func(*args, **kw)
```

Tuple of positional arguments Dictionary of named arguments

Argument matching:

	args	kw
>>> func(2, 'string', n1 = 5, n2 = 'a')	(2, 'string')	n1: 5, n2: 'a'

Combining the two argument passing methods

```
>>> def func(a, b, f=10, *args, **kw):
```

Classes - basics

```

class Rectangle:
    def __init__(self, width, length): # Constructor
        self.w = width                # instance attribute
        self.l = length                # instance attribute

# call the constructor to create an instance
rect1 = Rectangle( 5, 6 )
# create another instance
rect2 = Rectangle( 4, 2 )
# access instance's attributes
print rect1.w, rect1.h  → 5 6
print rect2.w, rect2.h  → 4 2

```

Classes - methods

```

class Rectangle:
    def __init__(self, width, length): # Constructor
        self.w = width                # instance attribute
        self.l = length                # instance attribute
    def area(self):                    # method area
        return self.w * self.l

# call the constructor to create an instance
rect1 = Rectangle( 5, 6 )
rect2 = Rectangle( 4, 2 )
# calling a method
print rect1.area()                  → 30
print rect1.area() + rect2.area()   → 38

```

Classes - overwriting operators

```

class Rectangle:
    def __init__(self, width, length): # Constructor
        self.w = width                # instance attribute
        self.l = length                # instance attribute
    def __add__(self, right):          # defines rect1+rect2
    def __mult__(self, right):         # defines rect1*rect2
    def __repr__(self):                # defines print(rect1)
    def __call__(self, *args, **kw):   # define rect1('hello')
    etc ...

```

* def __repr__(self) instead of def __repr__(self, right)

Modules

The file MyModule.py contains:

```
def func1(b):
    print 'You called func1 with b = ',b
```

```
>>> import MyModule
>>> dir(MyModule)
[ '__builtins__', '__doc__', '__file__', '__name__', func1 ]
>>> MyModule.func1(10)
You called func1 with b = 10

>>> from MyModule import func1
>>> func1(10)
You called func1 with b = 10

>>> from MyModule import *
>>> func1(10)
You called func1 with b = 10
```

Packages - organizing modules

```
>>> import sys
>>> print sys.path # print the Python path
[ 'dir1', 'dir2', 'dir3', ... ]
```

```
subdir1:
  __init__.py  ← Makes subdir1 a package
  foo.py:
    def Func(a,b):
    ...
    class Object:
    ...
subdir2:
  data.txt
  bar.py      ← subdir2 is NOT a package, bar.py CANNOT be imported
```

```
from subdir1 import foo
from subdir1.foo import Func
from subdir1.foo import Object
```

Standard libraries

sys, os, string, math, cgi, commands, shelve etc...

Example:


```
>>> import sys
>>> path = sys.path
>>> print path[:3]
[ '', 'C:\\Program Files\\Python', 'C:\\Program Files\\Python\\Lib\\plat-win']
```

Extending Python

Add functionality to Python
Gaining access to legacy code

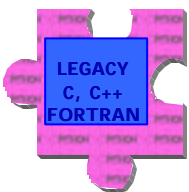
Extending PYTHON

Implement



Platform independent
Portability

Wrap



Platform dependent
speed

Wrapping C code

```

#include "MyLib.h"
...
PyObject *fact_w(PyObject *obj)
{
    int n = PyGetInt(obj);
    int res = fact(n);
    PyObject *Pyres = PyMakeInt(res)
    return Pyres;
}
...
MyLibModule.so
        
```

```

MyLib.h
extern int fact(int i);
        
```

```

MyLib.a
/* Compute factorial of n */
int fact(int n) {
    if (n<=1) return 1;
    else return n*fact(n-1);
}
        
```

```

>>> import MyLib
>>> a = 5
>>> MyLib.fact(a)
        
```

Numeric extension

Efficient storage and manipulation of large arrays of data.

Concepts

In C:

```
PyArrayObject
void *data
int *shape
char typecode
```

In Python:

```
import Numeric
ar = Numeric.array( [(1,5,5,9),
                    (-2,3,4,26),
                    (1, 2, 3, 7) ])
print ar.shape           (3,4)
print ar.typecode()     'l'
print ar.iscontiguous() '1'
```

* ar.shape = (3,4) and not (4,3)

Reshaping

```
>>> B = Numeric.array([ [1,2,3],[4,5,6] ])
[ 1 2 3 ] Numeric.reshape(B,(2,3)) [ 1 2 ]
[ 4 5 6 ]                             [ 3 4 ]
                                         [ 5 6 ]
shape (2, 3)                             (3, 2)

[ 1 2 3 ] Numeric.reshape(B,(-1,)) [ 1 2 3 4 5 6 ]
[ 4 5 6 ]                             (1,6) same as (6,)
shape (2,3)                             (1,6) same as (6,)
```

* I invert the shape of all arrays

Indexing & Slicing

```
>>> B = Numeric.array( [ [1,2,3],[4,5,6],[7,8,9] ] )
```

* B[1,0] instead of B[0,1]

Element-wise operations

Element-wise operation at C speed !

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ bop } \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 \text{ bop } 4 \\ 2 \text{ bop } 5 \\ 3 \text{ bop } 6 \end{bmatrix} \quad \text{uop } \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} \text{uop } 1 \\ \text{uop } 2 \\ \text{uop } 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ bop } 2 \text{ same as } \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ bop } \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \text{ bop } 2 \\ 2 \text{ bop } 2 \\ 3 \text{ bop } 2 \end{bmatrix}$$

~~$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ bop } \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$~~

binary operators (bop) : +, x, /, -, %, power ...
 unary operators (uop) : sin, cos, sqrt ...

Universal functions.

The most common Numeric universal functions:

- take, transpose, repeat, choose, ravel,
- nonzero, where, compress, diagonal, trace,
- searchsorted, sort, argsort, argmax, fromstring,
- dot, matrixmultiply, clip, indices, swapaxes,
- concatenate, innerproduct, array_repr, array_str,
- resize, diagonal, repeat, convolve, where, identity,
- sum, cumsum, product, cumproduct, alltrue,
- sometrue.....

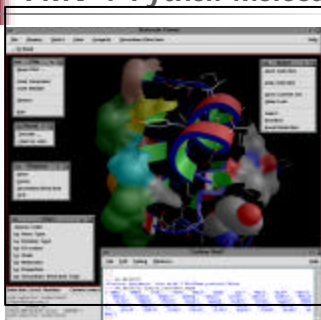
II - PMV

- o Fundamentals
- o Commands
- o Specialized extension: ADT

II - PMV

- o Fundamentals
 - Application design features

PMV : Python Molecule Viewer



DEMO



Application design features

- Dynamic loading of commands
- Python shell for scripting
- Dual interaction mode (GUI /Shell)
- Lightweight commands: Macros
- Command logging
- Dynamic commands (introspection)
- User-preferences / customization




Application design features (cont'd)

- Load multiple molecules
- Hierarchical representation of molecules
- Create/Select homogeneous sets
- Current selection concept
- Commands apply to current selection
- Interactive commands




II - PMV

- **Commands**
 - ⌘ Create, delete, write molecules
 - ⌘ Selection
 - ⌘ Basic representations and coloring
 - ⌘ Advanced representations
 - ⌘ Editing molecules
 - ⌘ ...




Create, delete, write molecules

- o Create:
 - z PDBReader
 - z Mol2Reader
 - z PDBQReader
 - z PDBQSReader
 - z PQRReader
 - z GeneralReader
- o Delete
 - z deleteMol
- o Write
 - z PDBWriter




Selection

- o From string
- o By picking
 - z molecule, chain, residue, atom
- o By distance
- o Displayed lines or cpk
- o Invert selection
- o On chain
- o ...




Basic representations and coloring

- o Color geometries by:
 - z atom type
 - z residue type
 - shapely, Rasmol, N to C
 - z chains
 - z molecules
 - z properties
 - z secondary structure type
 - z ...
- o Display by:
 - z lines
 - z cpk
 - z sticks and balls
 - z ...
- o Label:
 - z by properties



Advanced representations

- o MSMS molecular surface
 - ⌞ compute & display :
 - o MSMSMol
 - o MSMSSel
- o CA trace:
 - ⌞ compute
 - ⌞ extrude
 - ⌞ display
- o Spline:
 - ⌞ compute
 - ⌞ display
- o Secondary Structure:
 - ⌞ get SS information:
 - o from file
 - o from stride
 - ⌞ extrude
 - o default (rectangle, circle)
 - o circle
 - o rectangle
 - o ellipse
 - o ...
 - ⌞ display




Editing molecules

- o AI DE Module:

(pyBabel reimplementation of some of the Babel v1.6 functionalities)

 - ⌞ atom type assignment
 - ⌞ gasteiger charges calculation
 - ⌞ atom type conversion
 - ⌞ rings detection
 - ⌞ bond order assignment
 - ⌞ aromaticity detection
 - ⌞ hydrogen atoms addition

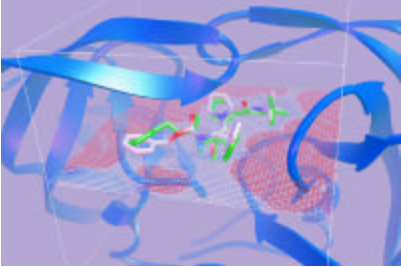


II - PMV

- o Specialized extension: ADT
 - ⌞ AutoDock fundamentals
 - ⌞ AutoDock ToolKit (ADT)

AutoDock fundamentals

Automated docking of a flexible ligand to macromolecules using affinity grids



AutoDock ToolKit (ADT)

- o AutoTors : ligand preparation
- o AutoGpf : grid definition
- o AutoDpf : docking parameters definition
- o AutoStart : job launching and monitoring
- o AutoAnalyze : docking results analysis

DEMO

III - From Building Blocks to applications

- o MolKit
- o DejaVu
- o ViewerFramework
- o Putting it all together
- o Writing a simple command

* blocks instead of 'blocks'

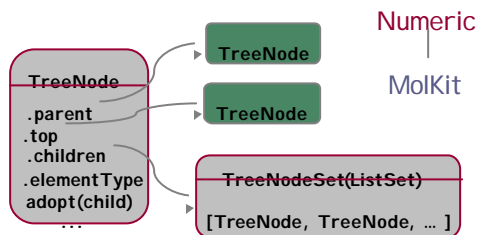
III - From Building Blocks to applications

o MolKit

- Hierarchical data-structure
- TreeNodes and TreeNodeSets
- Derived classes
- Parsers
- Examples

* blocks instead of blocks

Hierarchical data-structure



Hierarchical structure (cont'd)

- building trees by adoption
- TreeNodeSet slicing and indexing
- multi-level hierarchy
- dynamic adding of new members
- shortcut to access children's members

```
parent = TreeNode()
child = TreeNode()
parent.adopt(child)

node.children[5]
node.children[10:25]

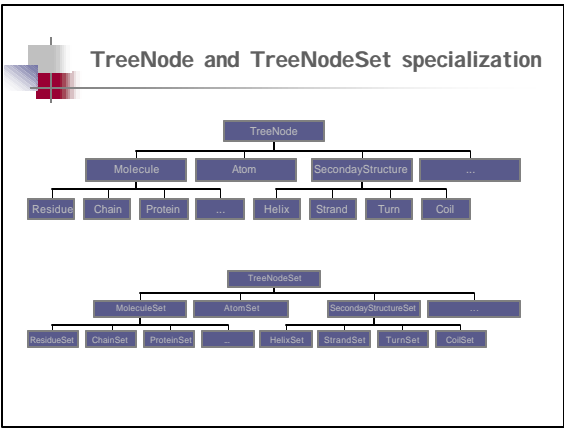
node.children[0].children[-1]

node = TreeNode()
node.newMember = myvalue

print node.children.name
['NoName', 'NoName', 'NoName',...]
```

TreeNode, TreeNodeSet

- o **TreeNodeSet:**
 - ⌘ Boolean operation
 - ⌘ uniq()
 - ⌘ split()
 - ⌘ sort()
 - ⌘ NodesFromName()
 - ⌘ findChildrenOfType()
 - ⌘ findParentOfType()
 - ⌘ ...
- o **TreeNode:**
 - ⌘ adopt() / remove()
 - ⌘ full_name()
 - ⌘ NodeFromName()
 - ⌘ split() / merge()
 - ⌘ getParentOfType()
 - ⌘ findType()
 - ⌘ compare()
 - ⌘ assignUniquel ndex()
 - ⌘ isAbove() / isBelow()
 - ⌘ ...



Parsers

PDB
Mol2
PQR
...
↓
Parser
↓
MoleculeSet

PDB parser
 Molecule
 ↙ ↘
 Chain Residue
 ↙ ↘
 Residue Atom

MOL2 parser
 Molecule Molecule Molecule
 ↙ ↘ ↘
 Chain Residue Atom
 ↙ ↘
 Residue Atom

```

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

Examples

```

>>> from MolKit import Read
>>> molecules = Read('./1crn.pdb')
>>> mol = molecules[0] # Read returns a ProteinSet
>>> print mol.chains.residues.name
>>> print mol.chains.residues.atoms[20:85].full_name()

>>> from MolKit.molecule import Atom
>>> allAtoms = mol.findType(Atom)
>>> set1 = allAtoms.get(lambda x: x.temperatureFactor > 20)

>>> allResidues = allAtoms.parent.uniq()
>>> import Numeric
>>> for r in allResidues:
...     coords = r.atoms.coords
...     r.geomCenter = Numeric.sum(coords) / len(coords)



```

III - From Building Blocks to applications

- o DeJaVu
 - z Overview
 - z Features
 - z Geometries
 - z DeJaVu and MolKit

* blocks instead of 'blocks'

Overview

Numeric
PyOpenGL

DeJaVu

Tkinter


```

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




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

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

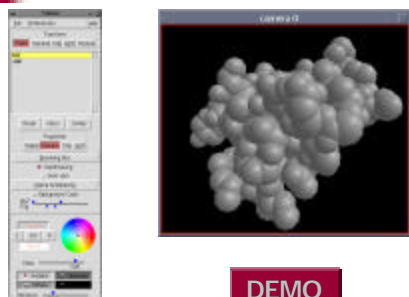
* lens instead of fence



Geometries

Geom	IndexedGeoms
o PolyLine	o IndexedPolyLines
o Points	o IndexedPolygons
o Spheres	o Triangle_Strip
o Labels	o Quad_Strip
o Arc3D...	o Cylinders

DejaVu and MolKit



DEMO

Demo Code

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

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

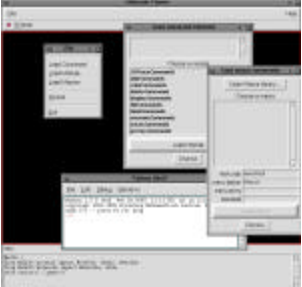
>>> sph = Spheres('sph', centers = coords, radii = radii,
>>>               quality=10)
>>> vi.AddObject(sph)
```

III - From Building Blocks to applications

- o ViewerFramework
 - Overview
 - Design features
 - Implementation

* blocks instead of blocks

Overview



```

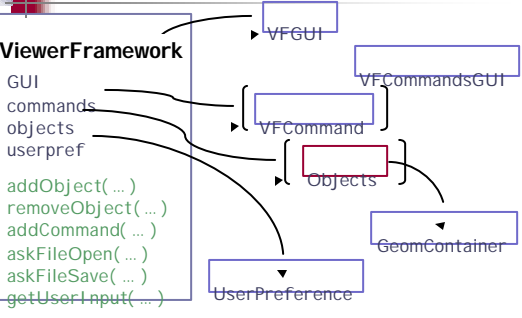
    graph TD
      ViewerFramework --> PyOpenGL
      ViewerFramework --> Tkinter
      PyOpenGL --- Numeric
      Tkinter --- Numeric
      ViewerFramework --> DejaVu
      ViewerFramework --> Idle
  
```

* ViewerFramework instead of Framework

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

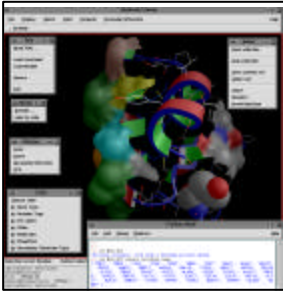
Implementation



```

    classDiagram
      class ViewerFramework {
        GUI
        commands
        objects
        userpref
        addObject(...)
        removeObject(...)
        addCommand(...)
        askFileOpen(...)
        askFileSave(...)
        getUserInput(...)
      }
      class VFGUI
      class VFCCommand
      class VFCCommandGUI
      class Objects
      class UserPreference
      class GeomContainer
      ViewerFramework --> VFGUI
      ViewerFramework --> VFCCommand
      ViewerFramework --> VFCCommandGUI
      ViewerFramework --> Objects
      ViewerFramework --> UserPreference
      ViewerFramework --> GeomContainer
  
```

PMV Architecture



```

    graph TD
      Numeric --- PyOpenGL
      Numeric --- Tkinter
      PyOpenGL --- DejaVu
      Tkinter --- DejaVu
      DejaVu --- Idle
      Idle --- ViewerFramework
      ViewerFramework --- Pynche
      ViewerFramework --- MolKit
      Pynche --- Pmv
      MolKit --- Pmv
  
```

* ViewerFramework instead of Framework

MolKit in PMV

```

>>> print mv.Mols
<MoleculeSet instance> holding 2 Protein

>>> mv.Mols[0]
<Protein Instance> 1cm with 1 MolKit.protein.Chain

>>> from MolKit.protein import Residue
>>> residues = mv.Mols.findType(Residue)
>>> residues
<ResidueSet instance> holding 154 Residue

>>> residues.myIndex = range(len(residues))
>>> residues[1:10].myIndex
[1,2,3,4,5,6,7,8,9]
  
```

DejaVu in PMV

```

>>> # access to DejaVu features from the pyShell
>>> vi = mv.GUI.VIEWER
>>> camera = vi.cameras[0]
>>> camera.Set(color=(1.,1.,1.))
>>> vi.Redraw()

>>> # show the Viewer's original GUI
>>> vi.GUI.root.deiconify()

>>> # hide the Viewer's original GUI
>>> vi.GUI.root.withdraw()
  
```

III - From Building Blocks to applications

o Writing a simple command

- z MVCommand overview
- z Subclassing MVCommand
- z Loading the command

* blocks instead of blocks

MVCommand overview



```
class MyCommand(MVCommand):
    def guiCallback(self, *args, **kw):
        apply( self.doitWrapper , args, kw)

    def doitWrapper(self, *args, **kw):
        self.beforeDoit()
        self.vf.tryto(apply(self.doit,args,kw))
        self.afterDoit()

    def doit(self, *args, **kw):
        pass

    def __call__(self, *args, **kw):*
        apply( self.doitWrapper , args, kw)
```



Subclassing MVCommand

```
class MyReader (MVCommand):
    def guiCallback(self):
        fTypes = [('PDB file','*.pdb'), ('PDBQ file','*.pdbq'),
                 ('MOL2 file','*.mol2')]
        filename = self.vf.askFileOpen(types=fTypes,
                                     title='Choose molecule file')
        mol = self.doitWrapper(filename, log = 1)
        return mol
    def __call__(self, filename = None, log=1, redraw=0):
        if filename is None: self.guiCallback()
        else: self.doitWrapper(filename, log=log, redraw = redraw)
    def doit(self, filename):
        from MolKit import Read
        molecules = Read(filename)
        self.vf.AddMolecule(molecules)
```

Loading the command

```

mv.addCommand( MyCommand(),
               'myCommand',
               MyCommandGUI )

```

Instance of a Command

Command name in PMV.

Instance of a CommandGUI : describes the GUI associated with a command (radiobutton, checkbutton, menu entry ...)

Example

```

>>> from Pmv.myCmd import MyReader
>>> from ViewerFramework.VFCommand import CommandGUI
>>> # Create a menu entry GUI for the command MyReader
>>> MyReaderGUI = CommandGUI ()
>>> # Add the GUI to the menuRoot in the File menu
>>> MyReaderGUI .addMenuCommand('menuRoot', 'File',
                                'Read File ...',index=0)
>>> # add the command with its associated GUI, and name to PMV
>>> mv.addCommand(MyReader (), 'myread', MyReaderGUI )

```


DEMO

Conclusion

- o Validity of the approach
- o Python
- o Availability
- o Future directions

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



* short instead of Short

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
- Short comings
 - reference counting, distribution mechanism, no strong typing

Availability

- Modularity enables fine grain licensing schemes (a la carte)
- Core modules are freely available
- Online Download site:
<http://www.scripps.edu/~sanner/Python>



Future directions

- Add support for editing molecular structures (i.e. mutations, deletion, addition)
- Interface with MMTK, Tinker,
- Enhance documentation and tutorials
- Setup a CVS server for programmers wanting to help !
- Too many to list



Acknowledgments

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