I - Fundamentals
- Code development strategies
- Python primer

II - PMV
- Fundamentals, main commands

III - From building blocks to applications
- MolKit, DejaVu, ViewerFramework...
- Putting it all together
- Writing a simple command

Conclusion
I - Fundamentals

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

The challenge

Visualization
Docking
Methods
Folding
Protein Engineering
Docking Methods
Sequence Analysis
Etc ...

Electrostatic Calculations
Molecular Surfaces
Modeling
Ab Initio Methods
MM - MD

Protein-Protein Complex Assemblies
Protein-Protein Ligand Ligand Complex Assemblies
AutoDock
SurfDock
SymGen
MSMS
HARMONY
SymSearch
“Traditional” solution

Electrostatics Calculations
Molecular Surfaces
Modeling
Ab Initio Methods
MM - MD

Visualization
Protein Engineering
Folding
Docking Methods
Sequence Analysis
Etc...

Our solution

High level language as a scripting environment

Writing an application

- High level coding
- Code re-use
- Extensible
Why Python?

<table>
<thead>
<tr>
<th>Our language needed</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>

Python based molecular software

  [http://www.cdl.ucsf.edu/chimera](http://www.cdl.ucsf.edu/chimera)
- MMTK: CNRS - Institut de Biologie Structurale
  [http://starship.python.net/crew/hinsen/mmtk.html](http://starship.python.net/crew/hinsen/mmtk.html)
- PyMOL: Delano Scientific.
  [http://www.pymol.org](http://www.pymol.org)
- PyDayLight: Daylight Chemical Information Systems, Inc.
  [http://starship.python.net/crew/dalke/PyDaylight/](http://starship.python.net/crew/dalke/PyDaylight/)
  [http://www.ks.uiuc.edu/~jim/mdtools/](http://www.ks.uiuc.edu/~jim/mdtools/)
- …

I - Fundamentals

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

1. Open the demoNPACI folder on your desktop
2. Double click on icon to start the installation of Python.
3. Follow the installation instructions:
   - Use default settings for steps 1 through 5
   - Step 6 Install Tcl/Tk: YES
   - Use default settings for the remaining steps.
4. In C:\Program Files\Python:
   - Double click starts a Python interpreter

Language characteristics

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

Basics

```python
>>> 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
```
List - mutable sequences

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

```
lst contains:
[2, 1, 'hello']
[2, 1, 'hello', 5]
[2, 5, 'hello', 5]
[2, 5, 'Paul', 'hello', 5]
[2, 5, 'Paul', 'hello']
```

Tuple - immutable sequences

```python
>>> 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 an immutable sequence !
```

Sequences indexing and slicing

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

- Indexing : `lst[i]`
  ```
  >>> lst[4]
  >>> lst[-2]
  ```

- Slicing : `lst[from:to]`
  ```
  >>> lst[2:-1]
  >>> lst[2::-1]
  ```
Exercise

In the Python interpreter:

1. Create a list l1 containing the following elements: 1, 2.6, 'b', 7, 9, 'z'
2. Assign a new value to the 3rd element of l1?

Solution

```python
>>> l1 = [1, 2.6, 'b', 7, 9, 'z']
>>> l1[2] = 'new'
>>> print l1
[1, 2.6, 'new', 7, 9, 'z']
>>> l2 = [1:-2]
>>> print l2
[2.6, 'new', 7]
```

Dictionary

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

>>> dict = { 'Marc':25, 30 : 'Jim' }
>>> dict[5.7] = 'joe'

# Accessing the information
>>> dict.items()  # This returns the items in the dictionary as a list of tuples.
[('Marc', 25), (30, 'Jim'), (5.7, 'joe')]
>>> dict.values()  # This returns the values in the dictionary as a list.
[25, 'Jim', 'joe']
>>> dict.keys()  # This returns the keys in the dictionary as a list.
['Marc', 30, 5.7]
>>> dict.has_key('Marc')  # This checks if the key exists in the dictionary.
True
```
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'
  ```

- **For, range(), break, continue**:  
  ```python
  >>> 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)

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

Function name

<table>
<thead>
<tr>
<th>Argument matching</th>
<th>Named arguments (optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td><code>b</code></td>
</tr>
<tr>
<td><code>n1</code></td>
<td><code>n2</code></td>
</tr>
</tbody>
</table>

Argument matching:

- `func( 2, 'string', 3.14 )`
- `func( 7.2, 'string', n2=15 )`
- `func( 'hello', 2, 5, 'bye' )`
- `func( n1=5 )`
- `func( n1=12, 3.14 )`

ERROR: missing argument
ERROR: positional argument after named argument

Functions - Arbitrary arguments

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

Tuple of positional arguments

Dictionary of named arguments

```text
<table>
<thead>
<tr>
<th>Argument matching</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a, b, n1=10, n2='hello'</code></td>
</tr>
</tbody>
</table>

Combining the two argument passing methods

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

```text
<table>
<thead>
<tr>
<th>Argument matching</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a, b, f=10, *args, **kw</code></td>
</tr>
</tbody>
</table>
Classes - basics

```python
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)
    rect2 = Rectangle(4, 2)

    # access instance's attributes
    print(rect1.w, rect1.h)  # 5 6
    print(rect2.w, rect2.h)  # 4 2
```

Classes - methods

```python
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

```python
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
        pass

    def __mul__(self, right):  # defines rect1 * rect2
        pass

    def __repr__(self):  # defines print(rect1)
        pass

    def __call__(self, *args, **kw):  # define rect1('hello')
        pass

e etc...
```
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

---

Exercise

1. Open in emacs the file myModule.py located in the demo folder.
2. Extend the class Rectangle with a method to compute the perimeter.
3. Start a Python interpreter. Import the class Rectangle from the package myModule.
4. Create an instance rect1 of that class.

---

Solution

```python
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
    def perimeter(self): # method perimeter
        return 2*(self.w + self.l)

>>> from MyModule import Rectangle
>>> rect1 = Rectangle(7,3) # creating an object rect1
>>> per = rect1.perimeter() # calling the new method.
>>> print per
```

---
Packages - organizing modules

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

- `subdir1`: __init__.py
- `foo.py`: def Func(a,b):
- `class Object:`

- `subdir2`: data.txt
- `bar.py` (CANNOT be imported)

Makes `subdir1` a package

- 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:

```python
>>> 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

Wrap

NEW MODULE

LEGACY C, C++, FORTRAN

Platform independent

Portability

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;
}

...extern int fact(int i);

MyLib.h

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

MyLib.a

MyLibModule.so

SWIG

>>> import MyLib

>>> a = 5

>>> MyLib.fact(a)

Numeric extension

Efficient storage and manipulation of large arrays of data.
**Concepts**

*In C:*

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

*In Python:*

```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'
```

**Reshaping**

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

[[1 2 3]
 [4 5 6]]

Numeric.reshape(B,(3,2))          # (3, 2)
shape (2, 3)                      # (3, 2)

[[1 2 3 4 5 6]]

Numeric.reshape(B,(-1,))          # (6,1) or (6,)
shape (2,3)                        # (6,1) or (6,)
```

**Indexing & Slicing**

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

[[1 2 3]
 [4 5 6]
 [7 8 9]]

B[1,0]                      # element
B[0,:1]                     # row
B[1,0]                      # column
```
Element-wise operations

Element-wise operation at C speed!

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{bmatrix}
\begin{bmatrix}
4 & 5 \\
6 & 7 \\
8 & 9
\end{bmatrix} =
\begin{bmatrix}
4 & 5 \\
6 & 7 \\
8 & 9
\end{bmatrix}
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{bmatrix} =
\begin{bmatrix}
4 & 5 \\
6 & 7 \\
8 & 9
\end{bmatrix}
\]

binary operators (bop) : +, -, *, /, %, 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

- Fundamentals
- Commands
- Specialized extension: ADT
**Installing PMV**

1- Open the demoNPACI folder on your desktop.
2- Double click on icon to install PMV.
3- Open the demo directory and double click on to start PMV.

---

**An extensible and customizable application.**

- **Load commands on the fly:**
  - loadCommand & loadModule commands.
  - Loading the readPDB commands and the displayCommands module.
    - File -> loadCommand -> fileCommands -> readPDB
    - File -> loadModule -> displayCommands -> Load Module
  - **Exercise:**
    1- Load the "deleteMol" command from the deleteCommands module.
    2- Load the "colorCommands" module.

---

**Basic interaction with the viewer**

<table>
<thead>
<tr>
<th>Action</th>
<th>Mouse Button</th>
<th>Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotation</td>
<td>Right</td>
<td></td>
</tr>
<tr>
<td>Scaling</td>
<td>Right</td>
<td>Alt</td>
</tr>
<tr>
<td>X/Y Translation</td>
<td>Right</td>
<td>Control</td>
</tr>
<tr>
<td>Z Translation</td>
<td>Right</td>
<td>Shift</td>
</tr>
<tr>
<td>Picking</td>
<td>Left</td>
<td></td>
</tr>
</tbody>
</table>
An extensible and customizable application.

- Define Commands to be applied to molecule when loaded: onAddObjectCommand
  - Loading a molecule displayed as lines:
    - File -> OnAddObjectCommand -> buildBondsByDistance -> displayLines -> dismiss
  
  ! The order in which the commands are selected is the order in which the commands are applied to the molecule.
    - File -> Read PDB -> 1cm.pdb -> Open
  
  - Exercise:
    - Load a new molecule to be displayed as cpk and colored by residue type.

An extensible and customizable application

- User preferences:
  - SetUserPreference command.
    - Set for the current session
      - Made as default for the following sessions

- Customization of the GUI
  - Customize command.

An extensible and customizable application.

- User specific customization files .pmvrc:
  1- Open the .pmvrc in emacs
    - Right click on the .pmvrc file and send to 'emacs'
  2- Start a customized Pmv session:
    - Double click on the icon to start a new Pmv session
  3- Load molecules:
    - Load the protease displayed as lines.
    - Load the indinavir displayed as cpk and colored by atom type.
Interactive Commands:
- Bind a Command to a mouse picking event.
- For applying the same command to multiple subsets of atoms.

Altering the representation of subsets of atoms:
ICOM -> displaySticksAndBalls -> ICOM level -> Residue

Undo Command:
- Undo commands stack
- User preference to set the number of undo.
- The "Delete Molecule" command voids the undo stack.

File -> Undo "last command executed"

"Regular Commands":
- Define the current selection:
  - Apply multiple commands to the same current selection.
- Current selection
  - homogeneous
  - empty -> everything is selected
Working with PMV

- Create a selection of Atoms:
  ICOM -> select -> Atom -> pick and drag select in the viewer.
- Change to a selection of Residues:
  Icom level Residue.
- Save the current selection as a set:
  Select -> Save current selection as a set -> set1 -> OK -> Clear selection.

Working with PMV.

- Retrieve the saved selection:
  Select -> Select a set -> set1 -> OK
- Color the selection by atom type and display as sticks and balls.
  Color -> by Atom Type -> lines -> OK
  Un/Display -> sticks and balls -> OK

Surfaces, Ribbons, etc...

- Compute the molecular surface of the protease and alter its representation:
  Clear Selection
  File -> Load Module -> msmsCommands -> Dismiss
  Select -> Select From String -> Molecule List ... -> protease -> Select -> Dismiss
  Compute -> Msms For Molecule -> density = 2.0 -> OK
  Color -> by Residue Type -> Rasmol -> msms -> OK
  Clear selection.

  And more ...
Working with PMV at a higher level

- Exercise:
  Represent the protease with a standard Ribbon diagram colored by chains.
  And more ....

Cool stuffs ...

- Color the surface of the ligand by the closest distance to protease.
- Magic lens.

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

- Create:
  - PDBReader
  - Mol2Reader
  - PDBQReader
  - PDBQSReader
  - PQRReader
  - GeneralReader

- Delete
  - deleteMol

- Write
  - PDBWriter
Selection

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

Basic representations and coloring

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

Advanced representations

- MSMS molecular surface
  - compute & display:
    - MSMS Mol
    - MSMS Sel
- CA trace:
  - compute
  - extrude
  - display
- Spline:
  - compute
  - display

Secondary Structure:

- get SS information:
  - from file
  - from stride
- extrude
  - default (rectangle, circle)
  - rectangle
  - ellipse
  - ...
- display
Editing molecules

- AIDE 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

III - From Building Blocks to applications

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

III - From Building Blocks to applications

- MolKit
  - Hierarchical data-structure
  - TreeNode and TreeNodeSets
  - Derived classes
  - Parsers
  - Examples
Hierarchical data-structure

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

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

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)
```

TreeNodeSet:
- Boolean operation
- uniq()
- split()
- sort()
- NodesFromName()
- findChildrenOfType()
- findParentOfType()

TreeNode:
- adopt() / remove()
- full_name()
- NodeFromName()
- split() / merge()
- getParentsOfType()
- findType()
- compare()
- assignUniqIndex()
- isAbove() / isBelow()

TreeNode, TreeNodeSet
TreeNode and TreeNodeSet specialization

Parsers

Examples

```python
>>> from MolKit import Read
>>> molecules = Read('./1crn.pdb')  # Read returns a ProteinSet
>>> 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

- DejaVu
  - Overview
  - Features
  - Geometries
  - DejaVu and MolKit

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

```python
>>> 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

Geometries

Geom
- PolyLine
- Points
- Spheres
- Labels
- Arc3D...

IndexedGeoms
- IndexedPolyLines
- IndexedPolygoms
- Triangle_Strip
- Quad_Strip
- Cylinders...

DejaVu and MolKit
..., from MolKit import Read
>>> molecules = Read('./1crn.pdb')
>>> mol = molecules[0]  # Read returns a ProteinSet
>>> coords = mol.chains.residues.atoms.coords
>>> radii = mol.defaultRadii()
>>> sph = Spheres('sph', centers = coords, radii = radii,
                   quality=10)
>>> vi.AddObject(sph)

III - From Building Blocks to applications

- ViewerFramework
  - Overview
  - Design features
  - Implementation
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

```
GeoContainer
VFGUI
VFCommandsGUIT
VFCommand
Objects
UserPreference
addObject(…)
removeObject(…)
addCommand(…)
askFileOpen(…)
askFileSave(…)
getUserInput(…)
```

III - From Building Blocks to applications

- Putting it all together:
  - Specializing ViewerFramework
  - MolKit in PMV
  - DejaVu in PMV
MolKit in PMV

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

>>> mv.Mols[0]
<Protein instance> 1crn 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

```python
# 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

- Writing a simple command
  - MVCommand overview
  - Subclassing MVCommand
  - Loading the command
**MVCommand overview**

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

class MyCommandGUI:
    def __init__(self):
        pass

    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**

```python
from Pmv.mvCommand import 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')
        val['redraw']=1
        mol = apply(self.doitWrapper, (filename, ), val)
        return mol
    def __call__(self, filename, **kw):
        kw = kw.copy()
        if not kw.has_key('redraw'): kw['redraw']=1
        kw = kw.copy()
        apply(self.doitWrapper, (filename, ), kw)
    def doit(self, filename):
        from MolKit import Read
        molecules = Read(filename)
        for mol in molecules: self.vf.addMolecule(mol)
```

**Loading the command**

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

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

---

31
Example

```python
>>> 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)
```

Conclusion

- Validity of the approach
- Python
- Availability
- 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
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

- 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