

 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

---

---

---


---

---

---

---

---

 I - Fundamentals

- Code development strategies
- Python primer

---

---

---

---

---

---

---

---

# I - Fundamentals

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

---

---

---

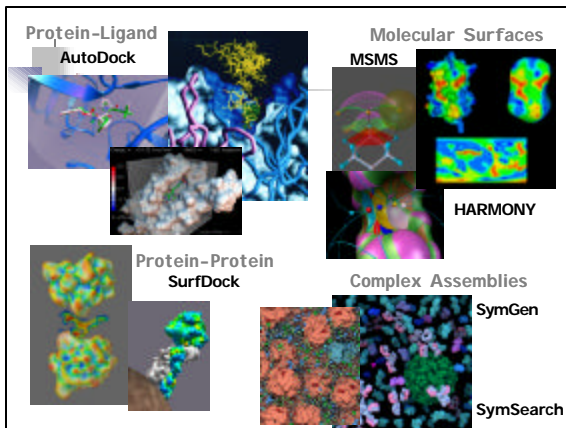
---

---

---

---

---



---

---

---

---

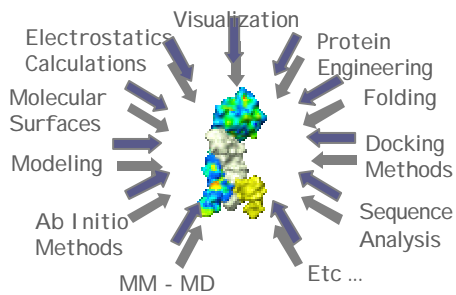
---

---

---

---

## The challenge



---

---

---

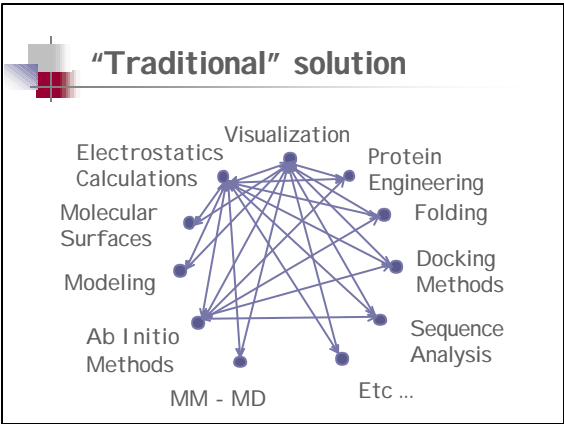
---

---

---

---

---




---

---

---

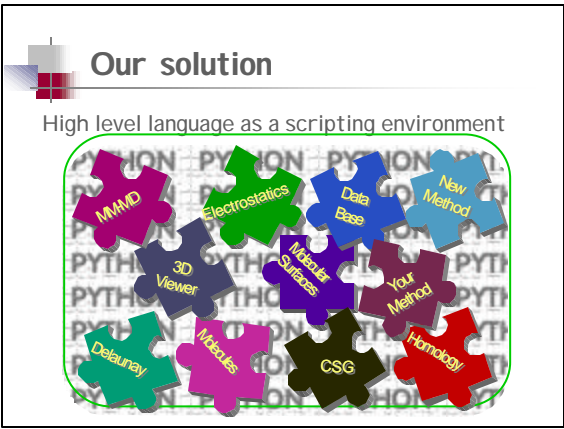
---

---

---

---

---




---

---

---

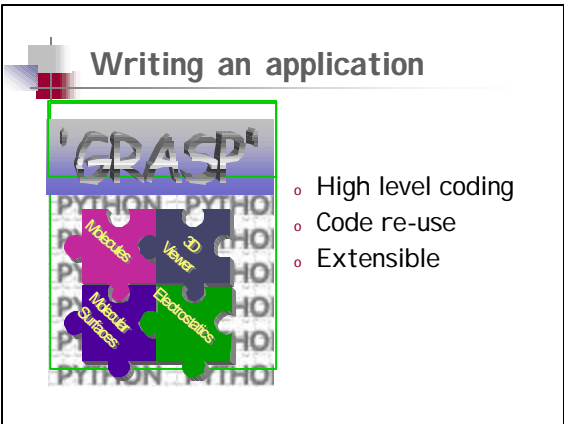
---

---

---

---

---




---

---

---


---

---

---

---

---



## 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, ...

---

---

---


---

---

---

---

---



## 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 : UI UC - Theoretical Biophysics Group.  
<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 demoNPAC1 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

```

>>> a = 2           # integer
>>> b = 7.5         # float   - No Memory allocation
>>> c = 'hello'     # string  - No variable declaration
>>> d = " World!"  # string too
>>> # this is a comment - simple and intuitive
>>> print c+d
>>> print "sum :", a+b
  
```

---

---

---

---

---

---

---

---

## List - mutable sequences

lst contains:

```

>>> lst = [2, 1, 'hello'] → [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']

```

---

---

---

---

---

---

---

---

## Tuple - immutable sequences

tup contains:

```

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

```

---

---

---

---

---

---

---

---

## Sequences indexing and slicing

```

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

```

0	1	2	3	4	5
1	'b'	6	'a'	8	'e'
-6	-5	-4	-3	-2	-1

- Indexing : lst[i]

```

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

```

0	1	2	3	4	5
1	'b'	6	'a'	8	'e'
-6	-5	-4	-3	-2	-1

- Slicing : lst[from:to]

```

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

```

0	1	2	3	4	5
1	'b'	6	'a'	8	'e'
-6	-5	-4	-3	-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?
- 3- If l2 = l1[1:-2] guess the elements contained in l2.

---

---

---

---

---

---

---

---



## Solution

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

---

---

---

---

---

---

---

---



## Dictionary

```
dict = { key1 : value1, key2 : value2, ...}
item1
>>> dict = { 'Marc':25, 30 : 'Jim' }
>>> dict[5.7] = 'joe'           ▶ { 'Marc':25, 30:'Jim', 5.7:'joe' }
# Accessing the information
>>> dict.items()               ▶ (('Marc',25), (30,'Jim'), (5.7,'joe'))
>>> dict.values()              ▶ [ 25,'Jim', 'joe']
>>> dict.keys()                ▶ ['Marc', 30, 5.7]
>>> dict.has_key('Marc')      ▶ True
```

---

---

---

---

---

---

---

---

## Control flow

- While :
 

```
>>> b = 0
>>> while b < 5:
...     print b; b=b+1
0 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			

---

---

---

---

---

---

---

---

## 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 ...

```

---

---

---

---

---

---

---

---

## 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
```

---

---

---

---

---

---

---

---

## 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. I mport the class Rectangle from the package myModule.
- 4- Create an instance rect1 of that class.

---

---

---

---

---

---

---

---

## Solution

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

---

---

---

---

---

---

---

---

## Packages - organizing modules

```

>>> 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

```

Makes subdir1 a package

```

from subdir1 import Foo
from subdir1.foo import Func
from subdir1.foo import Object

```

subdir2 is **NOT** a package, bar.py **CANNOT** be imported

---

---

---

---

---

---

---

---

## 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

---

---

---

---

---

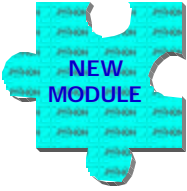
---

---

---

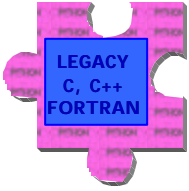
## Extending PYTHON

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
        
```

SWIG

```

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'
```

A 3x4 array of integers is shown with dimensions m (width) and n (height). A red arrow points from the C struct fields to the array.

---

---

---

---

---

---

---

---

### Reshaping

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

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	<code>Numeric.reshape(B,(3,2))</code>	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$
shape (2,3)		(3,2)

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	<code>Numeric.reshape(B,(-1,))</code>	$[1, 2, 3, 4, 5, 6]$
shape (2,3)		(6,1) or (6,)

---

---

---

---

---

---

---

---

### Indexing & Slicing

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

A 3x3 array is shown with row (r) and column (c) indices. A green oval highlights the first row (B[0,:]). A red circle highlights the element 4 (B[1,0]). A blue oval highlights the second column (B[:,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

---

---

---

---



---

---

---

---

## 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.

- o 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

Action	Mouse button	Modifier
Rotation	Right	
Scaling	Right	Alt
XY Translation	Right	Control
Z Translation	Right	Shift
Picking	Left	

---

---

---


---

---

---

---

---



### 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 -> 1crn.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.

---

---

---

---

---

---

---

---



## Working with PMV

- 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

---

---

---

---

---

---

---

---



## Working with PMV

- 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"

---

---

---

---

---

---

---

---



## Working with PMV

- "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

<ul style="list-style-type: none"> <li>◦ Color geometries by:           <ul style="list-style-type: none"> <li>⌞ atom type</li> <li>⌞ residue type               <ul style="list-style-type: none"> <li>shapely, Rasmol, N to C</li> </ul> </li> <li>⌞ chains</li> <li>⌞ molecules</li> <li>⌞ properties</li> <li>⌞ secondary structure type</li> <li>⌞ ...</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>◦ Display by:           <ul style="list-style-type: none"> <li>⌞ lines</li> <li>⌞ cpk</li> <li>⌞ sticks and balls</li> <li>⌞ ...</li> </ul> </li> <li>◦ Label:           <ul style="list-style-type: none"> <li>⌞ by properties</li> </ul> </li> </ul>
--	---

---

---

---


---

---

---

---

---



## Advanced representations

<ul style="list-style-type: none"> <li>◦ MSMS molecular surface           <ul style="list-style-type: none"> <li>⌞ compute &amp; display :               <ul style="list-style-type: none"> <li>◦ MSMSMol</li> <li>◦ MSMSSel</li> </ul> </li> </ul> </li> <li>◦ CA trace:           <ul style="list-style-type: none"> <li>⌞ compute</li> <li>⌞ extrude</li> <li>⌞ display</li> </ul> </li> <li>◦ Spline:           <ul style="list-style-type: none"> <li>⌞ compute</li> <li>⌞ display</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>◦ Secondary Structure:           <ul style="list-style-type: none"> <li>⌞ get SS information:               <ul style="list-style-type: none"> <li>◦ from file</li> <li>◦ from stride</li> </ul> </li> <li>⌞ extrude               <ul style="list-style-type: none"> <li>◦ default (rectangle, circle)</li> <li>◦ circle</li> <li>◦ rectangle</li> <li>◦ ellipse</li> <li>◦ ...</li> </ul> </li> <li>⌞ display</li> </ul> </li> </ul>
--	---

---

---

---

---

---

---

---

---



## 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

---

---

---

---

---

---

---

---



## III - From Building Blocks to applications

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

---

---

---

---

---

---

---

---



## III - From Building Blocks to applications

- o MolKit
  - ⌞ Hierarchical data-structure
  - ⌞ TreeNodes and TreeNodeSets
  - ⌞ Derived classes
  - ⌞ Parsers
  - ⌞ Examples

---

---

---

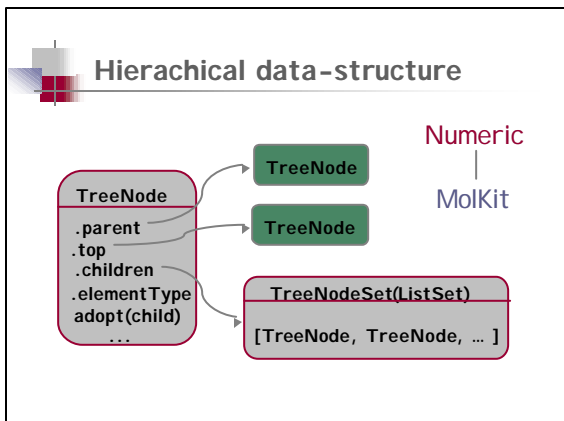
---

---

---

---

---




---

---

---

---

---

---

---

---

### Hierarchical structure (cont'd)

- building trees by adoption
 

```
parent = TreeNode()
child = TreeNode()
parent.adopt(child)
```
- TreeNodeSet slicing and indexing
 

```
node.children[5]
node.children[10:25]
```
- multi-level hierarchy
 

```
node.children[0].children[1]
```
- dynamic adding of new members
 

```
node = TreeNode()
node.newMember -- myvalue
```
- shortcut to access children's members
 

```
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()
  - ≪ assignUniqIndex()
  - ≪ isAbove() / isBelow()
  - ≪ ...

---

---

---

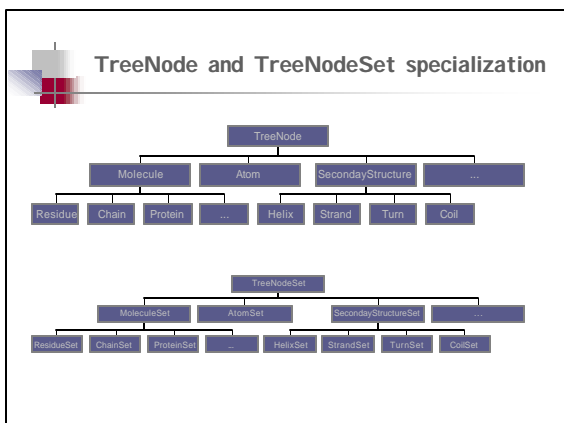
---

---

---

---

---




---

---

---

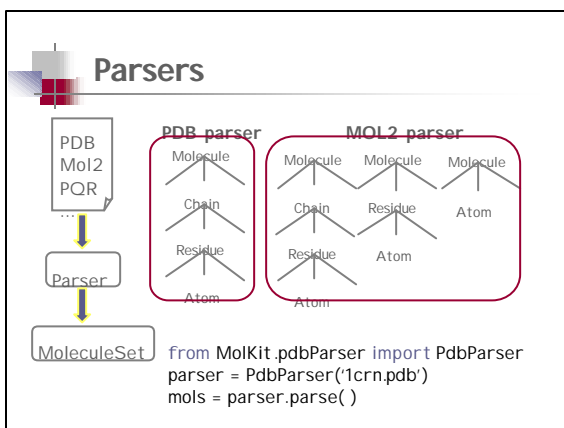
---

---

---

---

---




---

---

---

---

---

---

---

---

### 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

---

---

---

---

---

---

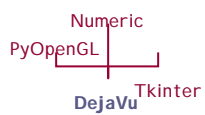
---

---

## Overview



DEMO



```
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

- 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

---

---

---

---

---

---

---

---

## 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 ....

---

---

---

---

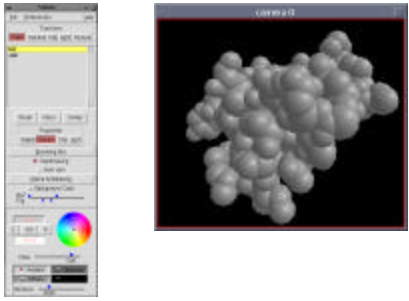
---

---

---

---

## DejaVu and MolKit




---

---

---

---

---

---

---

---

## Demo Code

```

>>> 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

- o ViewerFramework
  - Overview
  - Design features
  - Implementation

---

---

---

---

---

---

---

---

## Overview

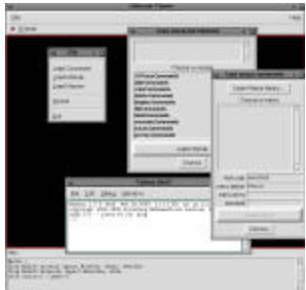


Diagram illustrating the software stack:

```

Numeric
  |
PyOpenGL --- Tkinter
  |
DejaVu
  |
  Idle
  |
ViewerFramework

```

---

---

---

---

---

---

---

---

## 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

---

---

---

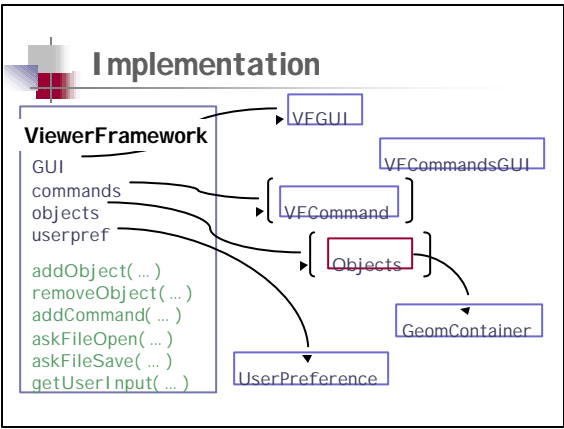
---

---

---

---

---




---

---

---

---

---

---

---

---

## III - From Building Blocks to applications

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

---

---

---

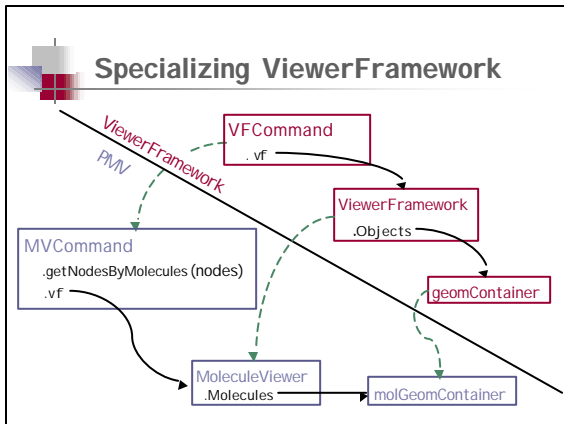
---

---

---

---

---




---

---

---

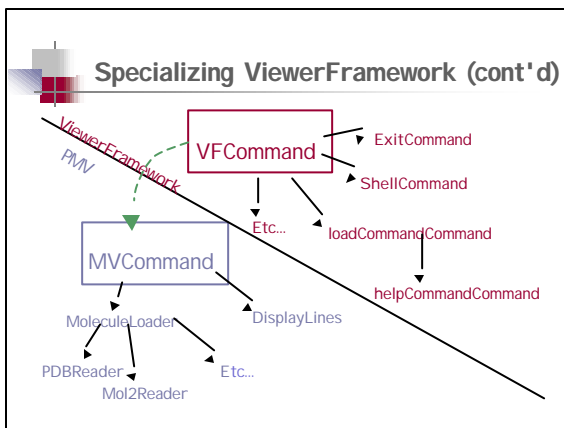
---

---

---

---

---




---

---

---

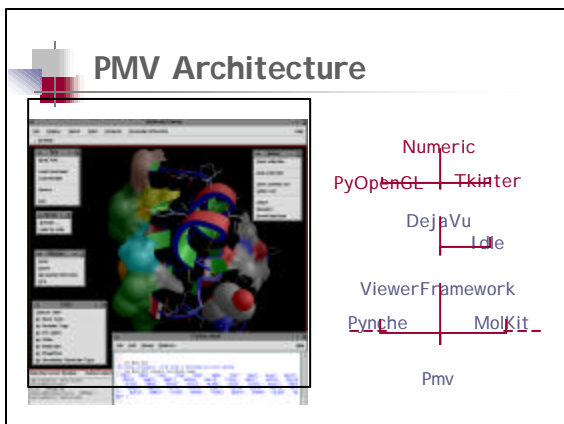
---

---

---

---

---




---

---

---

---

---

---

---

---



## MolKit in PMV

```
>>> 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

```
>>> # 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

---

---

---

---


---

---

---

---

## 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

```

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):
        if not kw.has_key('redraw'): kw['redraw']=1
        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

```

mv.addCommand( MyCommand(),
               Command name 'myCommand',
               in PMV.
               MyCommandGUI )
  
```

Instance of a Command

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

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




---

---

---

---

---

---

---

---



## Python

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

---

---

---

---

---

---

---

---



## Availability

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

---

---

---

---

---

---

---

---



## Future directions

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

---

---

---

---

---

---

---

---