Biopython's Bio.PDB

Thomas Hamelryck
thamelry@binf.ku.dk
BIO/DIKU, University of Copenhagen
December 2019

What is Biopython?

- http://www.biopython.org
- Collection of bioinformatics modules
- Comes with extensive structural bioinformatics support

Biopython for biological sequences

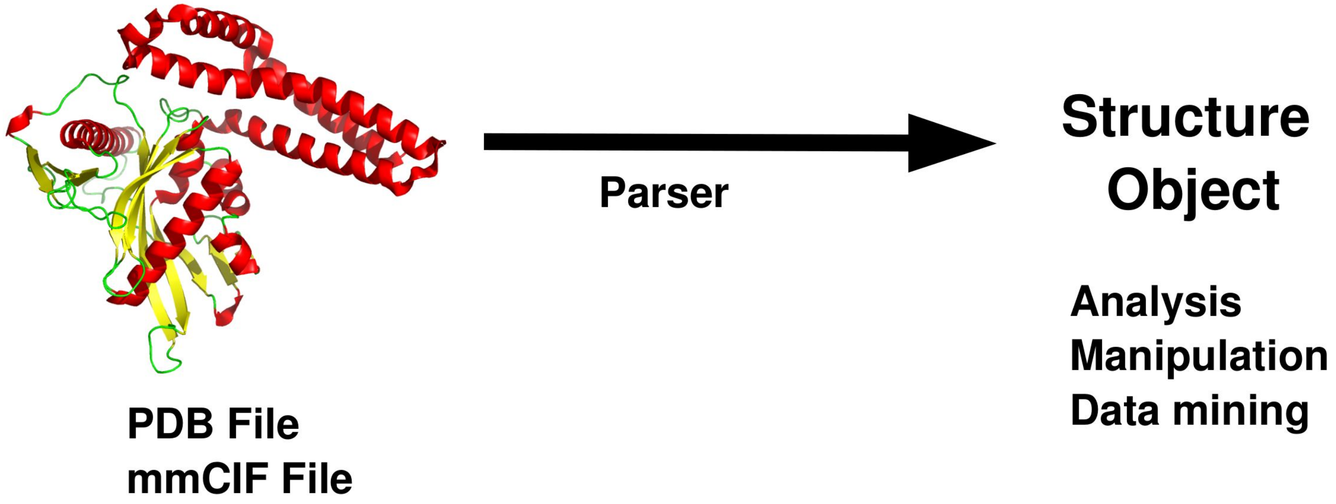
- Sequence class
- Parsing sequence files
  - FASTA, GenBank, SwissProt
- Parsing program output
  - ClustalW, BLAST
- Access to online services
  - EUtils, ExPASy
- Running programs
  - BLAST, ClustalW

Biopython's other functionality

- Various parsers
  - KEGG, Affymetrics, Medline,...
- BioSQL
  - Store sequences & their annotations
- **Structural bioinformatics**
  - **Bio.PDB**

Bio.PDB's purpose

- Allow easy access to molecular structure data



Bio.PDB's functionality

- Parsers
  - PDB, mmCIF
- Structure class
- Solvent exposure
  - ASA, rASA, Residue depth, HSE, CN
- Secondary structure
  - DSSP
- Misc
  - Polypeptide class, PDB database interface, PDB output...
  - Superposition class, Vector class...

References & information

- Bio.PDB FAQ
  - https://biopython.org/wiki/The_Biopython_Structural_Bioinformatics_FAQ
- Reference articles
  - **PDB parser and structure class implemented in Python.** Hamelryck, T., Manderick, B. (2003) Bioinformatics 19: 2308–2310
  - **Biopython: freely available Python tools for computational molecular biology and bioinformatics.** (2009) P. Cock, T. Antao, JT. Chang, BA. Chapman, CJ. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski… Bioinformatics 25:1422–1423

Importing Bio.PDB

```
# Import all of biopython
from Bio import *

# Import Bio.PDB alone
from Bio.PDB import *

# Import PDBParser alone
from Bio.PDB.PDBParser import PDBParser
```

Parsing a PDB file

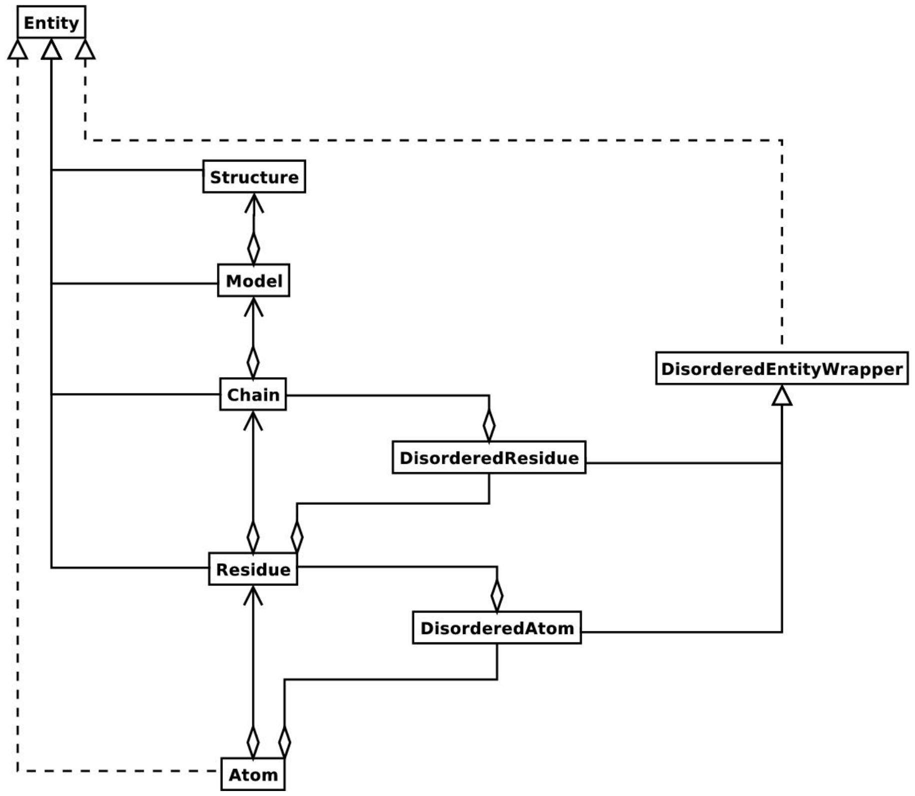
```
from Bio.PDB import *

# Create parser
parser=PDBParser()

# Get structure from file
structure=parser.get_structure(“Trypsin”, “2PTC.pdb”)
print(structure)
```

Structure object architecture

- SMCRA
  - Structure
  - Model
  - Chain
  - Residue
  - Atom



Using a structure object

```
# First model
model=structure[0]
# Chain E (trypsin)
chain=model[“E”]
# Residue 16
residue=chain[16]
# The Calpha atom
atom=residue[“CA”]
# Shortcut
atom=structure[0][“E”][16][“CA”]
```

Using a Structure object

```
# Loop over model
for model in structure:
    # Loop over chain
    for chain in model:
        # Loop over residues
        for residue in chain:
            # Loop over atoms
            for atom in residue:
                print(atom)
```

Identifiers

```
# The get_id() method works on all levels

a=atom.get_id()               # “CA”
r=residue.get_id()            # (“H_GLC, 10, “A”)
c=chain.get_id()              # “A”
m=model_get_id()              # 0
```

Iterator shortcuts

```
# Get first model
model=structure[0]
# Loop over all atoms in structure
for atom in model.get_atoms():
    print(atom)
# Loop over all residues in model
for res in model.get_residues():
    print(res)
# Get list of atoms in structure
atom_list=list(model.get_atoms())
```

Atom methods

```
# Atom name
a.get_name()
# Temperature factor
a.get_bfactor()
# Coordinates as numpy array
a.get_coord()
# Coordinates as Vector object
a.get_vector()
# Alternative location specifier
a.get_altloc()
```

Distances and angles

```
# a1,a2,a3,a4 are Atom objects

# Distances: minus is overloaded for atom objects
distance=a1-a2

# Angles
from Bio.PDB import calc_angle
v1=a1.get_vector()
v2=a2.get_vector()
v3=a3.get_vector()
angle=calc_angle(v1, v2, v3)
```

Distances and angles

```
# Dihedral angles
from Bio.PDB import calc_dihedral
v1=a1.get_vector()
v2=a2.get_vector()
v3=a3.get_vector()
v4=a4.get_vector()
angle=calc_dihedral(v1, v2, v3, v4)
```

Residue methods

```
# Residue name
r.get_resname()
# Does the residue contain disordered atoms?
r.is_disordered()
# Residue identifier (a 3tuple) containing:
# (hetflag, sequence identifier, insertion code)
# The hetflag is:
# 1. “H_XXX” for ligand XXX
# 2. “W” for water
# 3. “ “ (blank) for amino acids.
hf, si, ic=residue.get_id()
```

Finding polypeptides

```
# Create a PPBuilder object
ppb=PPBuilder()

# Now find all Polypeptides in Model object m
pp_list=ppb.build_peptides(m)
for pp in pp_list:
	print pp

# Iterate over all residues in a Polypeptide object
for residue in pp1:
	print residue
```

Finding polypeptides

```
# Print sequence of first polypeptide
pp1=pp_list[0]
print pp1.get_sequence()

# Get phi, psi list
pp_list=pp1.get_phi_psi_list()
```

PDB output

```
# Create PDBIO object
io=PDBIO()

# Set the structure
io.set_structure(s)

# Save to file
io.save('out.pdb')
```

PDB output with selection

```
# Create a subclass of the Select base class
# You can overload these methods as necessary:
#     accept_atom
#     accept_residue
#     accept_chain
#     accept_model
class GlySelect(Select):
        def accept_residue(self, res):
                if res.get_resname()=='GLY':
                        return 1          # included in output
                else:
                        return 0          # not included in output
```

PDB output with selection

```
# Create PDBIO object
io=PDBIO()
# Set the structure
io.set_structure(s)
# Save
select=GlySelect()
io.save('out.pdb', select)
```

Vector class

```
From Bio.PDB import Vector
p=Vector(1,2,3)
q=Vector(4,5,6)
# Sum
z=p+q
# Dot product
dp=p*q
# Cross product
z=p**q
# Product with scalar
z=p**2
```

Vector class

```
# Division is done like this
z=p**(1/2)
# Length or norm
np=p.norm()


# Get vector of atom position
v=atom.get_vector()
```

Superposition

```
sup = Superimposer()
# Specify the atom lists
# 'fixed' and 'moving' are lists of Atom objects
# The moving atoms will be put on the fixed atoms
sup.set_atoms(fixed, moving)
# Print rotation/translation/rmsd
print sup.rotran
print sup.rms
# Apply rotation/translation to the moving atoms
sup.apply(moving)
```

Exercises

Use trypsin/trypsin inhibitor complex, pdb code **2PTC**

1. Find all residues in trypsin that have more than two close contacts to the inhibitor.
    - A “close contact” is any atom pair with distance lower than 3.5 Å.
2. Print (phi, psi) angles for both proteins, using the Polypeptide class and using your own code (use **calc_dihedral**).
    - Check if the (phi, psi) angles for the second amino acid match.
3. Output Trypsine to a separate PDB file.
4. Output all atoms within a sphere of 10 Å of the center of trypsin to a separate PDB file. Use the CA atoms to calculate the center.