Python and the Daylight Toolkit

Andrew Dalke <dalke@dalkescientific.com>
Dalke Scientific Software, LLC

My history

University of Illinois
- VMD and NAMD

Molecular Applications Group
- Look/GeneMine and DiscoveryBase

Bioreason
- PyDaylight, group maximum common substructure

Dalke Scientific Software
- Biopython, EUtils, PyRSS2Gen, ...
- metrics (for CombiChem)
- PyDrone (for AstraZeneca)

What do I do?

Help researchers do more science in less time

Make software tools fit the scientist's model of the world

- consistent and coherent idioms
- things should work as expected
- and fail as expected

Python

- A very high-level programming language.
- First released 14 years ago.
- A modern language, with support for:
  Object-oriented programming, modules, exceptions, iterators, garbage collection
- One of the few languages based on user-interface studies
- Easy to use and also powerful
  Chemists and software developers both enjoy it

- Implementations for C, Java and (in development) .Net

Interfacing to C

It's easy to extend Python using C, by hand, using semi-automated tools (Boost.Python) or automatic processors (SWIG, SIP).

I started at Bioreason in 1998. Daylight shop.
Wanted to use Python to develop new tools.

Roger Critchlow had written DaySWIG.
Made SWIG interface files based on the toolkit headers.
SWIG generated interfaces for Python, Perl, Tcl, ...

Sounds great, right? ....

My first dayswig_python program

```
import sys
from dayswig_python import dt_smilin, dt_cansmi, NULL_OB

for line in sys.stdin:
    mol = dt_smilin(line)
    if mol == NULL_OB:
        print “Cannot parse”
        continue
    print dt_cansmiles(mol, 0)
```

A slightly buggy cansmiles; updated for 2005

What's the problem?

```
import sys
from dayswig_python import dt_smilin, dt_cansmi, NULL_OB

for line in sys.stdin:
    mol = dt_smilin(line)
    if mol == NULL_OB:
        print "Cannot parse"
        continue
    print dt_cansmiles(mol, 0)
    dt_dealloc(mol)
```

Daylight handle created

But never destroyed

About 20% of Daylight toolkit calls are to dt_dealloc

Daylight's data model

Daylight toolkits use an object model with atom, bond, molecule, reaction, database and other data types. Objects have properties. Some properties are lists.

Very clean but hidden behind a C/Fortran interface.

Other languages have more explicit support for the idioms used by the toolkit

Automatic Garbage Collection

```
class smart_ptr:
    def __init__(self, handle):
      self.handle = handle
    def __int__(self, handle):
      return self.handle
    def __del__(self):
      dayswig_python.dt_dealloc(self.handle)

def smilin(smiles):
    return smart_ptr(dayswig_python.dt_smilin(smiles))

for line in sys.stdin:
    mol = smilin(line)
    if mol == dayswig_python.NULL_OB:
        print "Cannot parse"
        continue
    print dt_cansmi(mol, 0)
```

"smart_ptr" is a wrapper

It can be used wherever a Daylight integer handle is used

Hook into Python's garbage collection

Modules

Each module is its own namespace.
Don't need to put a "dt_" in front of everything

```
from daylight import Smiles

for line in sys.stdin:
    mol = Smiles.smilin(line)
    if mol == dayswig_python.NULL_OB:
        print "Cannot parse"
        continue
    print dt_cansmi(mol, 0)
```

Classes

Wrap the toolkit molecule type into a Molecule claass

```
class Molecule:
    def __init__(self, handle):
        self.handle = handle
    def cansmiles(self, iso=0):
        return dayswig_python.dt_cansmiles(self.handle)

def smilin(smiles):
    mol = smart_ptr(dayswig_python.dt_smilin(smiles))
    return Molecule(mol)
```

```
for line in sys.stdin:
    mol = Smiles.smilin(line)
    if mol == dayswig_python.NULL_OB:
        print “Cannot parse”
        continue
    print mol.cansmiles()
```

Molecules have atoms and bonds

Use __getattr__ to implement “atoms” and “bonds” properties. Convert the given stream into a Python list of wrapped objects

```
class dayobject:
    def __init__(self, handle):
        self.handle = handle

class Atom(dayobject):
    pass
class Bond(dayobject):
    pass

class Molecule(dayobject):
    def __getattr__(self, name):
        if name == “atoms”:
            return streamlist(self.handle, TYP_ATOM, Atom)
         elif name == “bonds”:
            return streamlist(self.handle, TYP_BOND, Bond)
```

streamlist

(for completeness)

```
def toList(seq, converter):
    data = []
    while 1:
        ele = dt_next(seq)
        if not ele:  # NULL_OB
            return data
        data.append(converter(ele))

def streamlist(handle, property, converter):
    strm = dt_stream(handle, property)
    if not strm:  # NULL_OB
        raise DaylightError, 'Property not available'
    data = toList(strm, converter)
    dt_dealloc(strm)
    return data
```

atom properties

```
class Atom(dayobject):
    def __getattr__(self, name):
        if name == “charge”:
            return dayswig_python.dt_charge(self.handle)
        elif name == “symbol”:
            return dayswig_python.dt_symbol(self.handle)
        ...
        else:
            raise AttributeError(name)
    def __setattr__(self, name, value):
        if name == “charge”:
            dayswig_python.dt_setcharge(self.handle, value)
        elif name == “symbol”:
            raise TypeError(”read-only property: %s” % (name,))
        else:
            self.__dict__[name] = value
```

The real code uses a dispatch table.

Using it

```
from daylight import Smiles

line = raw_input("Enter a SMILES string: ")
mol = Smiles.smilin(line)
print mol.cansmiles()
for atom in mol.atoms:
    print atom.symbol, atom.charge, len(atom.bonds)
```

```
Enter a SMILES string: C[N+]12CCC2CCCC1
[N+]12(C(CC1)CCCC2)C
C 0 1
N 1 4
C 0 2
C 0 2
C 0 3
C 0 2
C 0 2
C 0 2
C 0 2
```

Error checking

"Errors should never pass silently.
Unless explicitly silenced."
- the Zen of Python

About half of a well-written C program is error handling.

Every possible error that could happen should be checked. If it can't be handled pass it back to the caller.

Making a well-written program is very tedious!

Exceptions are language support for the common case of "give up and let the caller deal with it."

cansmi with exceptions

```
def lastError():
    # get the last error message from the error
    # use dt_smilinerrors() if version < 4.91
    # else use dt_errors(DX_ERR_NONE)

def smilin(smiles):
    mol = dayswig_python.dt_smilin(smiles)
    if mol == dayswig_python.NULL_OB:
        raise DaylightError(lastError())
    return Molecule(smart_ptr(mol))
```

```
for line in sys.stdin:
    try:
      mol = Smiles.smilin(line)
    except DaylightError, err:
        print "Cannot parse:", err
        continue
    print mol.cansmiles()
```

Polymorphic return types

dt_smilin can take a molecule or a reaction SMILES

```
def smilin(smiles):
    mol = dt_smilin(smiles)
    t = dt_type(mol)

    if t == TYP_MOLECULE:
        return Molecule(smart_ptr(mol))
    if t == TYP_REACTION:
        return Reaction(smart_ptr(mol))

    if not mol:
        daylight.__in_smilin = 1
        msg = daylight.lastError()
        daylight.__in_smilin = 0
        raise daylight.BadFormat(msg)

    # Not sure what this is, so get rid of it.
    dt_dealloc(mol)
    raise daylight.BadFormat(
         'Cannot interpret the SMILES string data type')
```

SMARTS objects

Using Python's interactive shell

```
>>> from daylight import Smiles, Smarts
>>> mol = Smiles.smilin("Oc1nc(CC#N)nc2ccccc12")
>>> pat = Smarts.compile("[C,O]aa")
>>> pathset = pat.match(mol)
>>> print len(pathset), "paths"
4 paths
>>> for path in pathset:
...   for atom in path.atoms:
...     print atom.symbol,
...   print
...
O C N
O C C
C C N
C C N
>>>
```

SMARTS filter

```
# Usage: smarts_filter.py <smarts> [optional list of file names]

import sys
from daylight import Smiles, Smarts

def smarts_filter(infile, pattern):
    for line in infile:
        smiles = line.split()[0]
        mol = Smiles.smilin(smiles)
        if pattern.match(mol, 1) is not None:
            print smiles

if __name__ == "__main__":
    smarts = sys.argv[1]
    filenames = sys.argv[2:]
    pattern = Smarts.compile(smarts)
    if not filenames:
        smarts_filter(sys.infile, pattern)
    else:
        for filename in filenames:
            smarts_filter(open(filename), pattern)
```

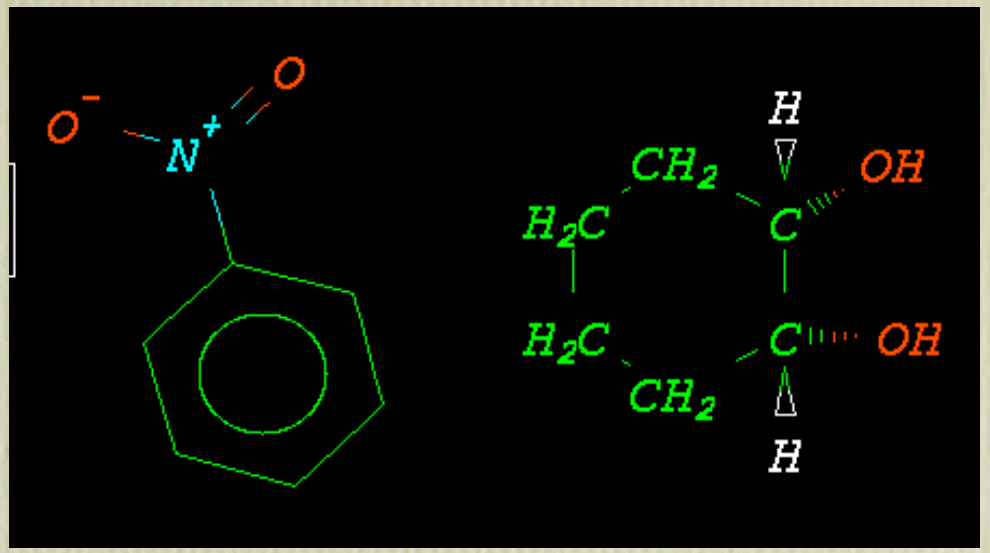
Fingerprints

```
>>> from daylight import Fingerprint as FP
>>> from daylight import Smiles
>>> mol = Smiles.smilin("c1ccccc1")
>>> fp = FP.generatefp(mol)
>>> fp.nbits
2048
>>> fp.bitcount
4
>>> fp2 = fp.copy()
>>> fp[0], fp2[0]
(0, 0)
>>> fp2[0] = 1
>>> fp[0], fp2[0]
(0, 1)
>>> FP.euclid(fp, fp2) * 2048
1.0
>>>
```

```
>>> fp3 = FP.allocfp(2048)
>>> FP.similarity(fp3, fp3, "1+2")
3.0
>>> fp3.stringvalue = \
'....2.............................U..........
.....U.............2.................+.........
............0.............6..................
.......................0.........6............
.........6.............................6............
..+....................0.......................
E....2..........E2.............+................
............2...1'.decode("daylight-fp")
>>> fp3.nbits
19
>>>
```

Depictions

Back-end support for PIL, PDF, QT, and Tk




```
# From PIL, the Python Imaging library
import Image, ImageDraw, ImageFont

from daylight import Smiles, Depict
from daylight.Depict import PILCallback, Colors

im = Image.new("RGB", (450, 250) )

mol = Smiles.smilin("c1ccccc1[N+](=O)[O-].O[C@H]1CCCC[C@H]1O")
dep = Depict.Depiction(mol)
dep.calcxy()
dep.color_atoms()

cb = PILCallback.PILCallback(im, font = ImageFont.load_path(
  "-adobe-courier-bold-o-normal--24-240-75-75-m-150-iso8859-1.pil"))
dep.depict(cb)

im.save(open("cis-resorcinol.png", "wb"), "png")
```

Program objects

```
>>> import os
>>> from daylight import Program
>>> clogptalk = os.path.expandvars("${DY_ROOT}/bin/clogptalk")
>>> prog = Program.MsgList(clogptalk)
>>> print prog("c1ccccc1")
['c1ccccc1 2.142 I LogPstar: 2.13']
>>> prog.NOTICE()
['Copyright (c) 1990-1997 Daylight CIS, Inc.',
 'Copyright (c) 1985-1997 Pomona College.']
>>> prog("Q")
['Q 0.000 80']
>>> prog.ENGLISHERRORS()
[]
>>> prog("Q")
['Q 0.000 (Invalid smiles input (fatal))']
>>>
```

Includes a “RestartableProgram” for those programs that crash on some input.

And more ...

Molecular editing
Thor/Merlin interfaces
MCL to PyDaylight converter
SMIRKS
HTTP toolkit

Cartridge

What about the Oracle cartridge?

Use ODBC or the cx_Oracle interface for Python

```
import cx_Oracle

connection = cx_Oracle.connect("user", "password", "TNS")

query = “NCCc1ccc(O)c(O)c1”
cursor = connection.cursor()
cursor.arraysize = 50
cursor.execute("""
        select smi, tanimoto(smi, :query) from demo
        where tanimoto(smi, :query) > 0.7
        order by tanimoto(smi, :query) desc”””,
        query = query)
for smi, tani:
    print smi, tani
```

Many Python libraries

Numeric / numarray - linear algebra, FFTs
matplotlib - plotting package
SciPy - includes the above plus ODE solvers, numerical integration, special functions, genetic algorithms, ...
reportlab - PDF generation

Other packages for OpenGL, XML, web programming, other database clients, embedded databases, CORBA, SOAP, image manipulation, GUI programming (Qt, wx, FLTK, Tk), network programming, win32 COM programming, web scraping

Biopython

A collection of bioinformatics software

- parsers (many bioinformatics formats)
- interfaces to local binaries (NCBIStandalong, ClustalW, Emboss)
- interfaces to web services and databases (NCBIWWW, EUtils, sequence retrieval)
- sequence, alignment, pathways, structure APIs
- implements algorithms for clustering, HMMs, SVMs, kD trees, tries
- graphical displays
- supports the BioSQL schema

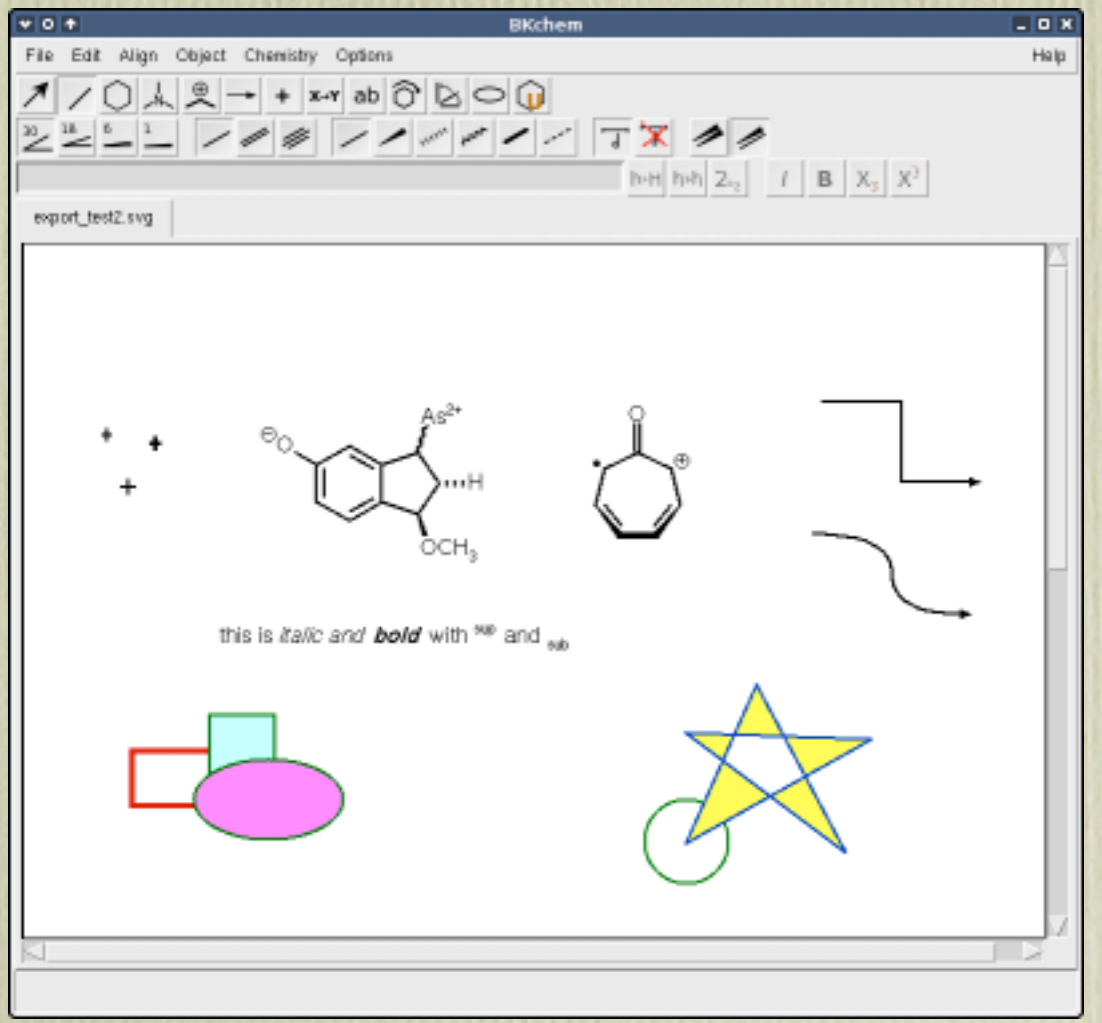
Other often used packages

MMTK - the molecular modeling toolkit
libsvm - a library for Support Vector Machines

PyMol, VMD, PMV and ViPEr, Chimera - structure visualization

Haven't tried these yet

BKchem - a free chemical drawing program.



PyQuante

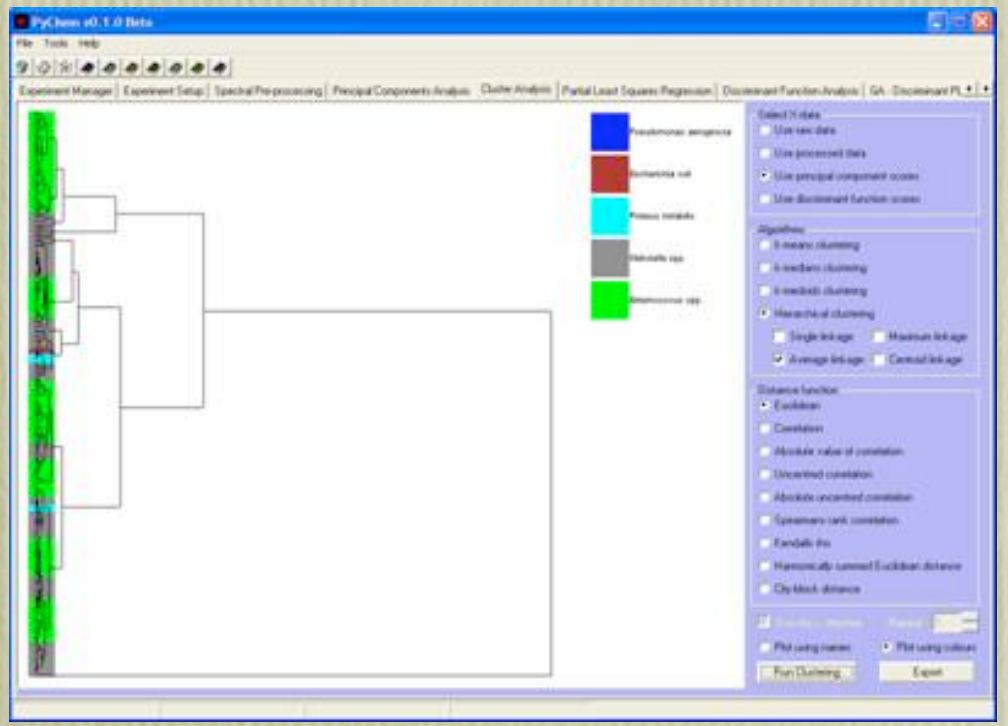
“ suite of programs for writing quantum chemistry software.”

```
def rhf(atomlist):
    "General wrapper for restricted closed-shell hartree fock"
    bfs = getbasis(atomlist,basis)
    S,h,Ints = getints(bfs,atomlist)
    orbs = get_guess(h,S)
    nel = get_nel(atomlist,charge)
    nclosed,nopen = divmod(nel,2)
    enuke = get_enuke(atomlist)
    eold = 0.
    for i in range(MaxIter):
        D = mkdens(orbs,0,nocc)
        G = get2JmK(Ints,D)
        F = h+G
        orbe,orbs = GHeigenvectors(F,S)
        energy = get_energy(h,F,D,enuke)
        print energy
        if abs(energy-eold) < ConvCriteria: break
        eold = energy
    return energy
```

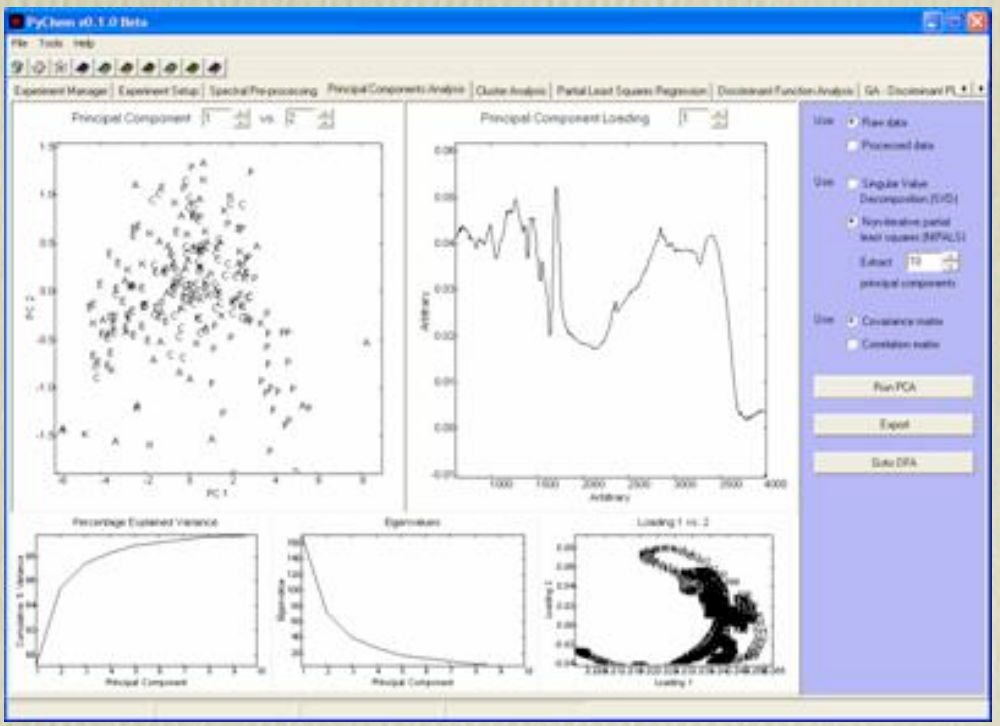
PyChem .org.uk - Roger Jarvis

"The purpose of this software tool is to provide a simple to install and easy to use graphical interface to chemometric algorithms."

Cluster analysis



PCA



Boa Constructor / wxPython / SciPy / wxPyPlot / PyCluster

mmlib

The Python Macromolecular Library (mmLib) is a software toolkit and library of routines for the analysis and manipulation of macromolecular structural models, implemented in the Python programming language.

It is accessed via a layered, object-oriented application programming interface, and provides a range of useful software components for parsing mmCIF, and PDB files, a library of atomic elements and monomers, an object-oriented data structure describing biological macromolecules, and an OpenGL molecular viewer. The mmLib data model is designed to provide easy access to the various levels of detail needed to implement high-level application programs for macromolecular crystallography, NMR, modeling, and visualization. This includes specialized classes for proteins, DNA, amino acids, and nucleic acids. Also included is a extensive monomer library, element library, and specialized classes for performing unit cell calculations combined with a full space group library.

esra

Esra is a pure Java library for the interactive analysis of molecular mechanics data. Written with the Java Reflection API (and particularly Jython and Mathematica in mind), it allows you to mangle your data in your favorite scripting language.

This is the first stable release. It features the basic building blocks for the analysis of molecular mechanics data. Its IO routines support formats that include GROMOS96, PDB, vmd/AMBER, and SYBYL/mol2. It has a Java linear algebra library, basic (geometric) analysis tools such as rmsds, fitting procedures, distance/angle/dihedral measurements, and hydrogen bond analysis.

Questions?

For more on Python in chemistry and biology see

http://www.dalkescientific.com/writings/