Python and Chemical Informatics

The Daylight and OpenEye toolkits, part I

Presented by Andrew Dalke, Dalke Scientific Software
for David Wild's I590 course at Indiana University
Feb. 23, 2005

What is Python?

- Byte-compiled high-level language
- Dynamically typed and strongly typed
- Very portable
- Easily extendable. Many platform-specific extensions.
- See http://www.python.org/ for more documentation and tutorials

High-level languages in Chemistry

- FORTRAN and C are fast
- Not as optimized for humans
- In the 1980s many large chemistry apps included a domain-specific language for "scripting" commands
- http://www.dalkescientific.com/writings/PyCon2004.html

Some choices

- In the early 1990s Tcl arose as an embeddable, extensible command language. Easy to embed and extend. Simple language. Not scalable to large project or complex data.
- Perl? Synthesis of "the unix way." By 1997 or so most had abandoned sh/awk/"little language" approach. But difficult to extend and tricky to handle complex data like molecular graphs. "Executable line noise" and "TMTOWTDI"

Python

First released in 1990. Based on research into how to make programming languages easier to use. Easy to embed and extend. Mixed OO and imperative styles (handles complex data with ease). Both non-programmers and software developers enjoy it.

Fewer people used it. “whitespace is significant?!” No obvious niche. “Programming in the large.”

Python in Chemistry

I first heard of people using Python in chemistry (molecular modeling) in 1995 or so. By 1997 I was hooked. Started PyDaylight in 1998. Been advocating it since.

Not the only one. UCSF/CGL with Chimera. Konrad Hinsen with MMTK.

Currently the most popular high-level language for comp. chemistry / chemical informatics. Software available from OpenEye, DeLano Scientific (PyMol), CCDC (eg Relibase). Internally used at Tripos, AstraZeneca, Vertex, Abbott and many more sites.

Starting Python

Interactive mode at the Unix prompt.
Could also use one of several IDEs.

```
% python
Python 2.2.2 (#1, Feb 24 2003, 19:13:11)
[GCC 3.2.2 20030222 (Red Hat Linux 3.2.2-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello from Python!"
Hello from Python!
>>>
```

"print" is a statement

Text in matching 'single' or "double" quotes are strings.

A bit of math

```
>>> 4 + 5
9
>>> 2 * 4 + 5
13
>>> p = 9
>>> 13 * p
117
>>> q = 3 ** p
>>> q
19683
>>>
```

The interactive shell prints the result of an expression (unless it returns None)

Assignment (in this case to the variable named "p") is not an expression

Exponentiation

Create a variable named “location” with value “New Mexico”

Print adds a space between two fields

```
>>> location = "New Mexico"
>>> print "Hello from", location
Hello from New Mexico
>>> s = "Hello from " + location
>>> print s
Hello from New Mexico
>>> s
'Hello from New Mexico'
>>>
```

Can add two strings to make a new string. (The space is not automatically added.)

Dynamically typed

All variables are "reference to object." There are no variable types. You do not need to say (and cannot say) that a variable may only reference a string, integer, atom, etc.

```
>>> smiles = 8
>>> smiles = "c1ccccc1"
>>> print smiles
c1ccccc1
>>>
```

Strongly typed

No automatic conversion between data types
(except some numeric types like floats and integers)

```
>>> print "2" + 9
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

Zen of Python

"Explicit is better than implicit."
"In the face of ambiguity, refuse the temptation to guess."

String methods

```
>>> "New Mexico".startswith("New")
1
>>> "New Mexico".startswith("Mexico")
0
>>> "New Mexico".find("Mexico")
4
>>> "New Mexico"[:4]
'New '
>>> "New Mexico"[4:]
'Mexico'
>>> "New Mexico"[4]
'M'
>>> "New Mexico".lower()
'new mexico'
>>> "New Mexico".upper()
'NEW MEXICO'
>>>
```

The ‘if’ statement

```
>>> pH = 6.0
>>> if pH < 7.0:
...     print "Acidic"
... else:
...     print "Basic"
...
Acidic
>>>
```

Indentation is used for code blocks

There is one code block for the “if” and another for the “else”

More ‘if’s, ‘and’s and ‘or’s

```
>>> pH = 6.0
>>> if pH < 2.0 or pH > 10.0:
...   print "Don't pour it down the sink"
...
>>> pH = 7.15
>>> if 7.1 < pH and pH < 7.3:
...   print "Near physiological pH"
... elif ph <= 7.1:
...   print "Too acidic"
... elif pH > 7.3:
...   print "Too basic"
... else:
...   print "this only happens when pH == 7.3"
...
Near physiological pH
>>> if 7.1 < pH < 7.3:
...   print "Another way to write the test"
...
Another way to write the test
>>>
```

Lists

```
>>> names = ["*", "H", "He", "Li", "Be"]
>>> names[2]
'He'
>>> names[:3]
['*', 'H', 'He']
>>> names[-2:]
['Li', 'Be']
>>> names[-1]
'Be'
>>> names.append("B")
>>> names.extend( ["C", "N", "O", "F"] )
>>> names
['*', 'H', 'He', 'Li', 'Be', 'B', 'C', 'N', 'O', 'F']
>>> names.reverse()
>>> names
['F', 'O', 'N', 'C', 'B', 'Be', 'Li', 'He', 'H', '*']
>>> names.reverse()
>>>
```

Indicies start with ‘0’

“:” indicates a range/sublist

Negative numbers count backwards from the end

Dictionaries

```
>>> full_names = {
...   "*": "unknown",
...   "H": "hydrogen",
...   "He": "helium",
...   "Li": "lithium",
...   "Be": "beryllium",
...   "B": "boron",
...   "C": "carbon"}
>>> full_names["C"]
'carbon'
>>> full_names.keys()
['Be', 'C', 'B', 'H', '*', 'He', 'Li']
>>> full_names.values()
['beryllium', 'carbon', 'boron', 'hydrogen', 'unknown', 'helium', 'lithium']
>>> full_names.has_key("He")
1
>>> full_names["O"]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: O
>>> full_names["O"] = "oxygen"
>>> full_names["O"]
'oxygen'
>>>
```

Functions

```
>>> def get_full_name(atomno):
...   if not (0 <= atomno < len(names)):
...     return "unknown"
...   name = names[atomno]
...   if full_names.has_key(name):
...     return full_names[name]
...   return "unknown"
...
>>> get_full_name(0)
'unknown'
>>> get_full_name(1)
'hydrogen'
>>> get_full_name(8)
'oxygen'
>>> get_full_name(9)
'unknown'
>>> full_names["F"] = "florine"
>>> get_full_name(9)
'florine'
>>>
```

Start a function definition with "def"

This function is named "get_full_name".

It takes one parameter, internally named "atomno"

The body of the function definition is indented

"names" and "full_names" are found as global variables

Modules

Modules are repositories of variables, functions, classes and more.

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.cos(math.pi/4)
0.70710678118654757
>>> math.sqrt(2)/2
0.70710678118654757
>>>
```

"Namespaces are one honking great idea -- let's do more of those!"

‘for’ loops

Iterate over all elements of a list or list-like object

```
>>> names = ['*', 'H', 'He', 'Li', 'Be', 'B', 'C', 'N', 'O', 'F']
>>> for name in names:
...   if name.startswith("H"):
...     print name, "starts with an H"
...   if len(name) == 1:
...     print name, "has only one letter"
...   if name == "O":
...     break
...
* has only one letter
H starts with an H
H has only one letter
He starts with an H
B has only one letter
C has only one letter
N has only one letter
O has only one letter
>>>
```

Can stop early with the “break” statement

list-like objects?

File objects are list-like - It looks like a list of lines

“repr” converts a string to its Python presentation

```
>>> infile = open("/usr/local/daylight/v481/data/drugs.smi")
>>>
>>> for line in infile:
...   print repr(line)
...
'N12CCC36C1CC(C(C2)=CCOC4CC5=O)C4C3N5c7ccccc76 Strychnine\n'
'c1ccccc1C(=O)OC2CC(N3C)CCC3C2C(=O)OC cocaine\n'
'COc1cc2c(ccnc2cc1)C(O)C4CC(CC3)C(C=C)CN34 quinine\n'
'OC(=O)C1CN(C)C2CC3=CCNc(ccc4)c3c4C2=C1 lyseric acid\n'
'CCN(CC)C(=O)C1CN(C)C2CC3=CNc(ccc4)c3c4C2=C1 LSD\n'
'C123C5C(O)C=CC2C(N(C)CC1)Cc(ccc4O)c3c4O5 morphine\n'
'C123C5C(OC(=O)C)C=CC2C(N(C)CC1)Cc(ccc4OC(=O)C)c3c4O5 heroin\n'
'c1ncccc1C1CCCN1C nicotine\n'
'CN1C(=O)N(C)C(=O)C(N(C)C=N2)=C12 caffeine\n'
'C1C(C)=C(C=CC(C)=CC=CC(C)=CCO)C(C)(C)C1 vitamin a\n'
>>>
```

The lines end with the newline character

SMILES file

The SMILES file format is simple

- All data fits on a single line
- There is no header line
- The first word is the SMILES string
- The second word (if it exists) is the name or other molecule identifier
- Words are separated by whitespace
- No common definition for how to interpret text after the 2nd word

“split” a string

string.split() breaks a string up into a list of the words that were white-space separated

```
>>> s = "N12CCC36C1CC(C(C2)=CCOC4CC5=O)C4C3N5c7ccccc76 Strychnine\n"
>>> s.split()
['N12CCC36C1CC(C(C2)=CCOC4CC5=O)C4C3N5c7ccccc76', 'Strychnine']
>>> s.split()[0]
'N12CCC36C1CC(C(C2)=CCOC4CC5=O)C4C3N5c7ccccc76'
>>> smiles, name = s.split()
>>> smiles
'N12CCC36C1CC(C(C2)=CCOC4CC5=O)C4C3N5c7ccccc76'
>>> name
'Strychnine'
>>>
```

“tuple assignment”

Extracting the SMILES

Get a list of all the SMILES strings that are in the file.

```
>>> all_smiles = []
>>> for line in open("/usr/local/daylight/v481/data/drugs.smi"):
...     words = line.split()
...     all_smiles.append(words[0])
...
>>> len(all_smiles)
10
>>> all_smiles
['N12CCC36C1CC(C(C2)=CCOC4CC5=O)C4C3N5c7ccccc76',
'c1ccccc1C(=O)OC2CC(N3C)CCC3C2C(=O)OC', 'COc1cc2c(ccnc2cc1)C(O)C4CC(CC3)C(C=C)CN34',
'OC(=O)C1CN(C)C2CC3=CCNc(ccc4)c3c4C2=C1',
'CCN(CC)C(=O)C1CN(C)C2CC3=CNc(ccc4)c3c4C2=C1',
'C123C5C(O)C=CC2C(N(C)CC1)Cc(ccc4O)c3c4O5',
'C123C5C(OC(=O)C)C=CC2C(N(C)CC1)Cc(ccc4OC(=O)C)c3c4O5', 'c1ncccc1C1CCCN1C',
'CN1C(=O)N(C)C(=O)C(N(C)C=N2)=C12', 'C1C(C)=C(C=CC(C)=CC=CC(C)=CCO)C(C)(C)C1']
>>>
```

PyDaylight

- You’ve done a bit of toolkit programming
- Tedious - need to declare data types, correctly use dt_dealloc, use Daylight streams to walk through a list, and check all return types for errors
- PyDaylight does most of that for you
- It’s a “thick” wrapper to the toolkit. See http://daylight.com/meetings/mug99/Dalke/ http://daylight.com/meetings/mug00/Dalke/

Getting set up

PyDaylight is an extension to Python.
Python needs to know where extensions can be found.
It uses the PYTHONPATH environment variable.
This is a colon (“:”) separated path list.

I compiled PyDaylight and put it in my home directory on xavier, under `/home/adalke/local/lib/python2.2/site-packages` so you will need to do one of the following

for bash use

```
export PYTHONPATH=/home/adalke/local/lib/python2.2/site-packages
```

for csh/tcsh use

```
setenv PYTHONPATH /home/adalke/local/lib/python2.2/site-packages
```

Parsing a SMILES string

"daylight" is a module
"Smiles" is a submodule of the "daylight" module
This loads the module "daylight.Smiles" and makes it accessible through the variable named "Smiles"

```
>>> from daylight import Smiles
>>> mol = Smiles.smilin("c1ccccc1O")
>>> mol
Molecule(1)
>>>
```

Convert the given SMILES into a Molecule object

The Molecule has a reference to the Daylight toolkit handle. In this case the handle's value is `1`

The Molecule class

A Molecule instance has atoms, bonds, and cycles.

These numbers are the handles to the underlying Daylight objects

```
>>> mol
Molecule(1)
>>> mol.atoms
[Atom(3), Atom(4), Atom(5), Atom(6), Atom(7), Atom(8), Atom(9)]
>>> mol.bonds
[Bond(11), Bond(12), Bond(13), Bond(14), Bond(15), Bond(16), Bond(17)]
>>> mol.cycles
[Cycle(19)]
>>>
```

A Molecule may contain 0 or more of what a chemist would call a molecule: **`O.O`** contains two water molecules

Atoms and Bonds

```
>>> for atom in mol.atoms:
...     print atom.symbol, len(atom.bonds), atom.imp_hcount, atom.aromatic
...
C 2 1 1
C 2 1 1
C 2 1 1
C 2 1 1
C 2 1 1
C 3 0 1
O 1 1 0
>>>
```

Some atom properties:

atomic symbol
list of bonds (except implicit hydrogens)
implicit hydrogen count
aromatic flag (1 if aromatic, 0 if not)

```
>>> for bond in mol.bonds:
...    print bond.atoms[0].symbol, bond.symbol, bond.atoms[1].symbol, bond.bondorder
...
C ~ C 2
C ~ C 1
C ~ C 2
C ~ C 1
C ~ C 2
C ~ C 1
C - O 1
>>>
```

Some bond properties:

list of the two atoms
bond type (1, 2, 3, or 4) and symbol
aromatic flag (1 if aromatic, 0 if not)
bond type (Kekule)

And Cycles

Cycles have a list of the atoms and of the bonds in the cycle

```
>>> heroin = Smiles.smilin(
...     "C123C5C(OC(=O)C)C=CC2C(N(C)CC1)Cc(ccc4OC(=O)C)c3c4O5")
>>> for cycle in heroin.cycles:
...     name = aromatic_name[cycle.aromatic]
...     print len(cycle.atoms), "member", name, "ring"
...
5 member nonaromatic ring
6 member nonaromatic ring
6 member nonaromatic ring
6 member nonaromatic ring
6 member aromatic ring
>>>
```

Generating a SMILES

Molecules have a “cansmiles” method which returns a canonical SMILES string

```
>>> heroin.cansmiles()
'CN1CCC23C4Oc5c3c(CC1C2C=CC4OC(=O)C)ccc5OC(=O)C'
```

“xsmiles” returns the Kekule form

```
>>> heroin.xsmiles()
'CN1CCC23C4OC=5C3=C(CC1C2C=CC4OC(=O)C)C=CC5OC(=O)C'
```

Both take an option ‘iso’ flag to include isomeric labeling (atomic weight and chirality)

```
>>> uf6 = Smiles.smilin("[235U](F)(F)(F)(F)(F)F")
>>> uf6.cansmiles()
'F[U](F)(F)(F)(F)F'
>>> uf6.cansmiles(1)
'F[235U](F)(F)(F)(F)F'
```

cansmiles

Convert all SMILES from a file into canonical form

```
>>> from daylight import Smiles
>>> for line in open("/usr/local/daylight/v481/data/drugs.smi"):
...   smiles = line.split()[0]
...   mol = Smiles.smilin(smiles)
...   print mol.cansmiles()
...
O=C1CC2OCC=C3CN4CCC56C4CC3C2C6N1c7ccccc75
COC(=O)C1C2CCC(CC1OC(=O)c3ccccc3)N2C
COc1ccc2nccc(C(O)C3CC4CCN3CC4C=C)c2c1
CN1CC(C=C2C1CC3=CNc4cccc2c34)C(=O)O
CCN(CC)C(=O)C1CN(C)C2Cc3c[nH]c4cccc(C2=C1)c34
CN1CCC23C4Oc5c3c(CC1C2C=CC4O)ccc5O
CN1CCC23C4Oc5c3c(CC1C2C=CC4OC(=O)C)ccc5OC(=O)C
CN1CCCC1c2cccnc2
Cn1cnc2n(C)c(=O)n(C)c(=O)c12
CC(=CCO)C=CC=C(C)C=CC1=C(C)CCC1(C)C
>>>
```

Error Handling

Not all strings are SMILES strings

```
>>> Smiles.smilin("Not a SMILES string")
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "/home/adalke/local/lib/python2.2/site-packages/daylight/Smiles.py", line 59, in smilin
    raise daylight.BadFormat, msg
daylight.BadFormat:    ERROR: Invalid input: t (dy_parse_smiles)
```

PyDaylight converts Daylight's error codes into Python exceptions which can be caught and dealt with

```
>>> import daylight
>>> try:
...   Smiles.smilin("QQQ")
... except daylight.DaylightError:
...   print "It didn't like that"
...
It didn't like that
>>>
```

See the Python documentation for details on using exceptions

Simple filtering

Suppose you want the SMILES strings in a SMILES file which the molecule does not contain a nitrogen

```
>>> from daylight import Smiles
>>> for line in open("/usr/local/daylight/v481/data/drugs.smi"):
...   smiles = line.split()[0]
...   mol = Smiles.smilin(smiles)
...   for atom in mol.atoms:
...     if atom.number == 7:
...       break  # found a nitrogen so break out of the loop
...   else:  # this is an 'else' to the 'for' statement
...     # only get here if there was no 'break' in the loop
...     print smiles
...
C1C(C)=C(C=CC(C)=CC=CC(C)=CCO)C(C)(C)C1
>>>
```

SMARTS searches

Most substructure searches are more complicated than testing for only a single atom.

"How many carboxyl groups are in heroin?"

Import both Smiles and Smarts modules

```
>>> from daylight import Smiles, Smarts
>>> carboxyl = Smarts.compile("C(=O)O")
>>> carboxyl
SmartsObject(637)
>>> heroin = Smiles.smilin(
...    "C123C5C(OC(=O)C)C=CC2C(N(C)CC1)Cc(ccc4OC(=O)C)c3c4O5")
>>> matches = carboxyl.match(heroin)
>>> len(matches)
2
>>>
```

"Compile" the SMARTS pattern into an object

The terms "compile" and "match" were picked to match Python's "re" module

Match failures

A SMARTS match() returns Python's None object if there are no matches

```
>>> mol = Smiles.smilin("C")
>>> matches = carboxyl.match(mol)
>>> print matches
None
>>>
```

Match successes

On success the returned MatchObject is a list of paths. Each path has a list of atoms and bonds.

```
>>> mol = Smiles.smilin("c1cccnc1O")
>>> pat = Smarts.compile("[n,c]cO")
>>> for path in pat.match(mol):
...   for atom in path.atoms:
...     print atom.symbol,
...   print
...
C C O
N C O
>>>
```

Nitrogen counts

This prints the number of nitrogens in each molecule.

```
>>> from daylight import Smiles, Smarts
>>> nitrogen = Smarts.compile("[#7]")
>>> for line in open("/usr/local/daylight/v481/data/drugs.smi"):
...   smiles = line.split()[0]
...   matches = nitrogen.match(Smiles.smilin(smiles))
...   if matches is None:
...     print 0, smiles
...   else:
...     print len(matches), smiles
...
2 N12CCC36C1CC(C(C2)=CCOC4CC5=O)C4C3N5c7ccccc76
1 c1ccccc1C(=O)OC2CC(N3C)CCC3C2C(=O)OC
2 COc1cc2c(ccnc2cc1)C(O)C4CC(CC3)C(C=C)CN34
2 OC(=O)C1CN(C)C2CC3=CCNc(ccc4)c3c4C2=C1
3 CCN(CC)C(=O)C1CN(C)C2CC3=CNc(ccc4)c3c4C2=C1
1 C123C5C(O)C=CC2C(N(C)CC1)Cc(ccc4O)c3c4O5
1 C123C5C(OC(=O)C)C=CC2C(N(C)CC1)Cc(ccc4OC(=O)C)c3c4O5
2 c1ncccc1C1CCCN1C
4 CN1C(=O)N(C)C(=O)C(N(C)C=N2)=C12
0 C1C(C)=C(C=CC(C)=CC=CC(C)=CCO)C(C)(C)C1
>>>
```

[#7] matches [N,n]

match return None when there are no matches. 'is' is mostly used for checking for None. It is not the same as "=="

Structure Editing

Replace an oxygen with a sulpher

Doesn’t work like you might expect.

```
>>> mol = Smiles.smilin("c1ccccc1O")
>>> mol.atoms[-1].number
8
>>> mol.atoms[-1].number = 16
>>> mol.atoms[-1].number
8
>>>
```

The ‘mod’ flag

Molecules can only be modified if the ‘mod’ flag is set.

```
>>> mol.mod
0
>>> mol.mod = 1
>>> mol.atoms[-1].number = 16
>>> mol.atoms[-1].number
16
>>> mol.cansmiles()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "/home/adalke/local/lib/python2.2/site-packages/daylight/Molecule.py", line
116, in cansmiles
    raise DaylightError, "cannot make canonical smiles when mod is on"
daylight.DaylightError: cannot make canonical smiles when
mod is on
>>> mol.mod = 0
>>> mol.cansmiles()
'Sc1ccccc1'
>>>
```

But can only generate a SMILES string when the mod flag is off

Building acetic acid

atom by atom and bond by bond

```
>>> from daylight import Molecule, Bond
>>> mol = Molecule.Molecule()
>>> mol.mod = 1
>>> C1 = mol.addatom(6)
>>> C2 = mol.addatom(6)
>>> O1 = mol.addatom(8)
>>> O2 = mol.addatom(8)
>>> C1.imp_hcount = 3
>>> Bond.add(C1, C2)
Bond(6)
>>> Bond.add(C2, O1, 2)
Bond(7)
>>> Bond.add(C2, O2, 1)
Bond(8)
>>> O2.imp_hcount = 1
>>> mol.mod = 0
>>> mol.cansmiles()
'CC(=O)O'
>>>
```

Make 2 carbons and 2 oxygens

Make it a [CH3]

By default this adds a single bond

This adds a double bond

Specify the single bond

Make it an [OH]

Delete an atom or bond

Use the “dealloc” function in the “daylight” module

```
>>> mol = Smiles.smilin("c1ccccc1O")
>>> mol.mod = 1
>>> daylight.dealloc(mol.atoms[-1])
1
>>> mol.mod = 0
>>> mol.cansmiles()
'[c]1ccccc1'
>>>
```

The 1 means the deallocation succeeded

It’s [c] because I forgot to adjust the implicit hydrogen count on the carbon that was attached to the oxygen

```
>>> mol.mod = 1
>>> mol.atoms[-1].imp_hcount += 1
>>> mol.mod = 0
>>> mol.cansmiles()
'c1ccccc1'
>>>
```

Deletes can be tricky

I want to delete the 5th atom. Why doesn't this work?

```
>>> mol = Smiles.smilin("c1ccccc1O")
>>> mol.mod = 1
>>> daylight.dealloc(mol.atoms[4])
1
>>> mol.mod = 0
>>> mol.cansmiles()
Exception daylight.DaylightError: <daylight.DaylightError instance at
0x81e2cb4> in <bound method smart_ptr.__del__ of smart_ptr(1)> ignored
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "/home/adalke/local/lib/python2.2/site-packages/daylight/Molecule.py",
line 115, in cansmiles
    assert self.mod == 1, "dt_cansmiles returned None but mod
is off?"
AssertionError: dt_cansmiles returned None but mod is off?
```

The final chemistry isn't correct. In this case the implicit hydrogen counts of the neighboring atoms must be adjusted based on the old bond orders

Structure cleanup

New structures often need to be cleaned up.
("registered")

- remove salts
- adjust charge states
- normalize chemistry
- determine how to handle tautomers

Full registration is very complex.

Removing salts

Here’s one algorithm that mostly works.
DON’T USE IT - it fails too often

Get the SMILES string and split on the “.”. This gives each of the chemical compounds. The non-salt has the longest SMILES.

```
>>> filename = "/usr/local/daylight/v481/contrib/src/applics/clusterview/data.smi"
>>> for line in open(filename):
...   sizes = []
...   smiles = line.split()[0]
...   for smi in smiles.split("."):
...     sizes.append( (len(smi), smi) )
...   sizes.sort()
...   print sizes[-1][1]
...
```

Make a list that can be sorted by length.
Sort and pull out the longest SMILES

Better removal

Instead of using the longest SMILES string
convert each into a Molecule and use the atom counts.

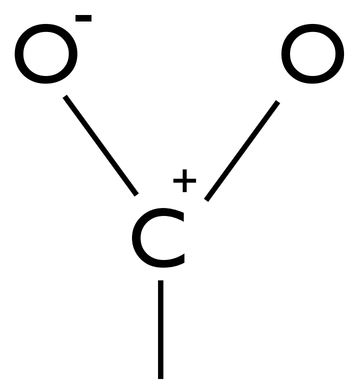
THIS CAN STILL FAIL

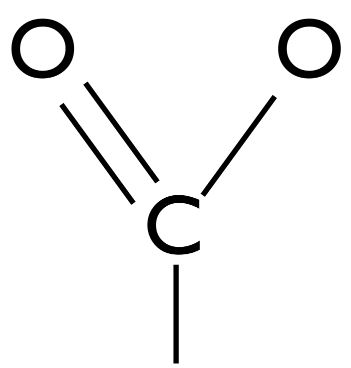
Sometimes the salt is larger than the compound.
Sometimes the salt *is* the compound.

Need to be careful.

Fixing chemistry

Different people have different notions of how to make chemistry fit the covalent graph model





A carboxylic group might be sketched (correctly) in either of these two forms. Need to pick one and convert the other form into this one.

Could use SMARTS and molecular editing

If a molecule matches the carboxylic SMARTS pattern
then change the charge on match atom 1 (the carbon) by -1
and change the charge on match atom 2 (the O-) by +1
and make the bond between atoms 1 and 2 a double bond

Feasible, and used in some companies

http://www.daylight.com/meetings/mug99/Kenny/kenny_mug99.htm

SMIRKS

SMIRKS is a concise language for representing reactions.

```
>>> from daylight import Smirks, Smiles
>>> t = Smirks.compile("[C+:1]([O-:2])[O:3]>>[C:1](=[O:2])[O:3]")
>>> mol = Smiles.smilin("C[C+]([O-])O")
>>> results = t.transform([mol])
>>> results
[Reaction(183)]
>>> results[0].product.cansmiles()
'CC(=O)O'
>>>
```

See http://www.daylight.com/dayhtml_tutorials/languages/smirks/

Exercise 1- compute_mw

Make a function called “compute_mw” which computes the molecular weight for a given SMILES string.

The function will start like this: `def compute_mw(smiles):`

PyDaylight includes molecular weight data in its Elements module

```
>>> from daylight import Elements
>>> Element.byNumber(8).mass
15.999000000000001
```

Remember to include the implicit hydrogen count.

Exercise 2 - explore the NCI data set

Using the NCI SMILES data set as converted by CACTVS, How many ...

1. ... SMILES are in the data set?
2. ... could not be processed by Daylight?
3. ... contain more than 30 atoms?
4. ... contain sulphers?
5. ... contain atoms other than N, C, O, S, and H?
6. ... contain more than one molecule in the SMILES?
7. ... have a chain of at least 15 atoms?
8. ... have more than 3 aromatic ring systems?

Exercise 3 - salts

How many different salts are in the NCI data set? List them as canonical SMILES.

How did you define which was a salt?

As I recall, at least one record has a salt which is larger than the compound. Find it, or show that I misremember.

Exercise 4 - atom deletion

Write a program that takes a SMILES string and atom position and deletes the atom at that position. It must fix the implicit hydrogen counts on what was the neighboring atoms.

The result may break a molecule into several molecules. (Eg, breaking "CCCC" in the second position creates "C" and "CC"). Print each molecule on its own line.

To get the SMILES and index from the command-line you may use

```
import sys

if len(sys.argv) != 3:
  raise SystemExit(”must give SMILES string and index”)
smiles = sys.argv[1]
index = int(sys.argv[2])
```