

# Useful and New Modules

Europython 2006 @ CERN

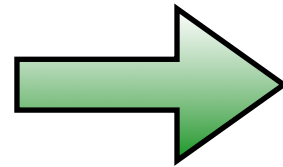
Andrew Dalke / Dalke Scientific Software  
dalke@dalkescientific.com



# About me

Using Python as my primary language since 1998

Develop software for computational chemistry and biology. Consultant since 1999.

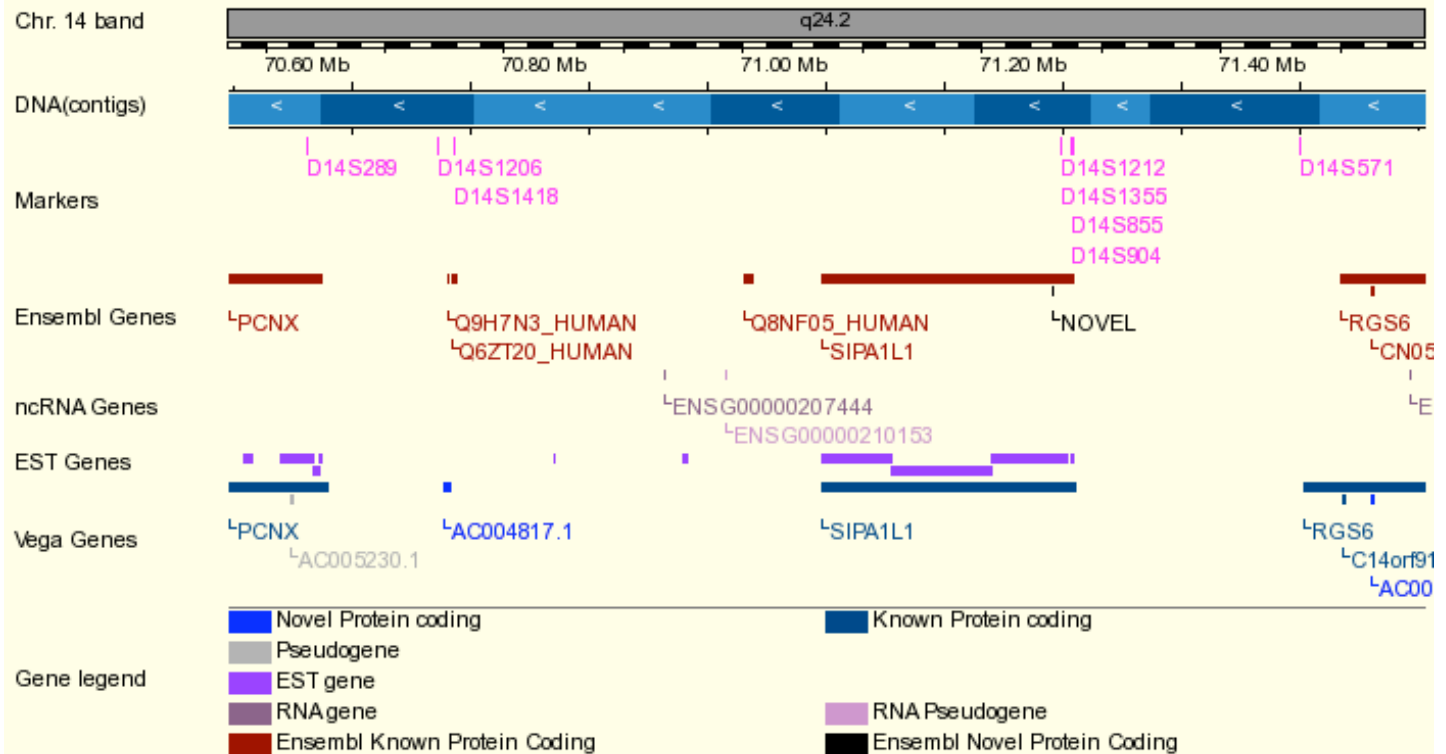
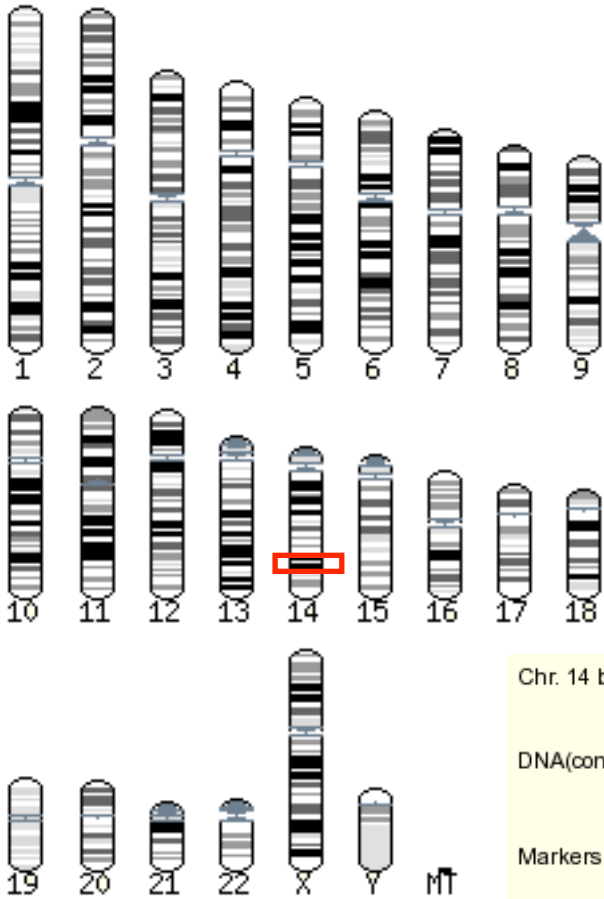


Santa Fe, New Mexico, USA

Göteborg, Sweden

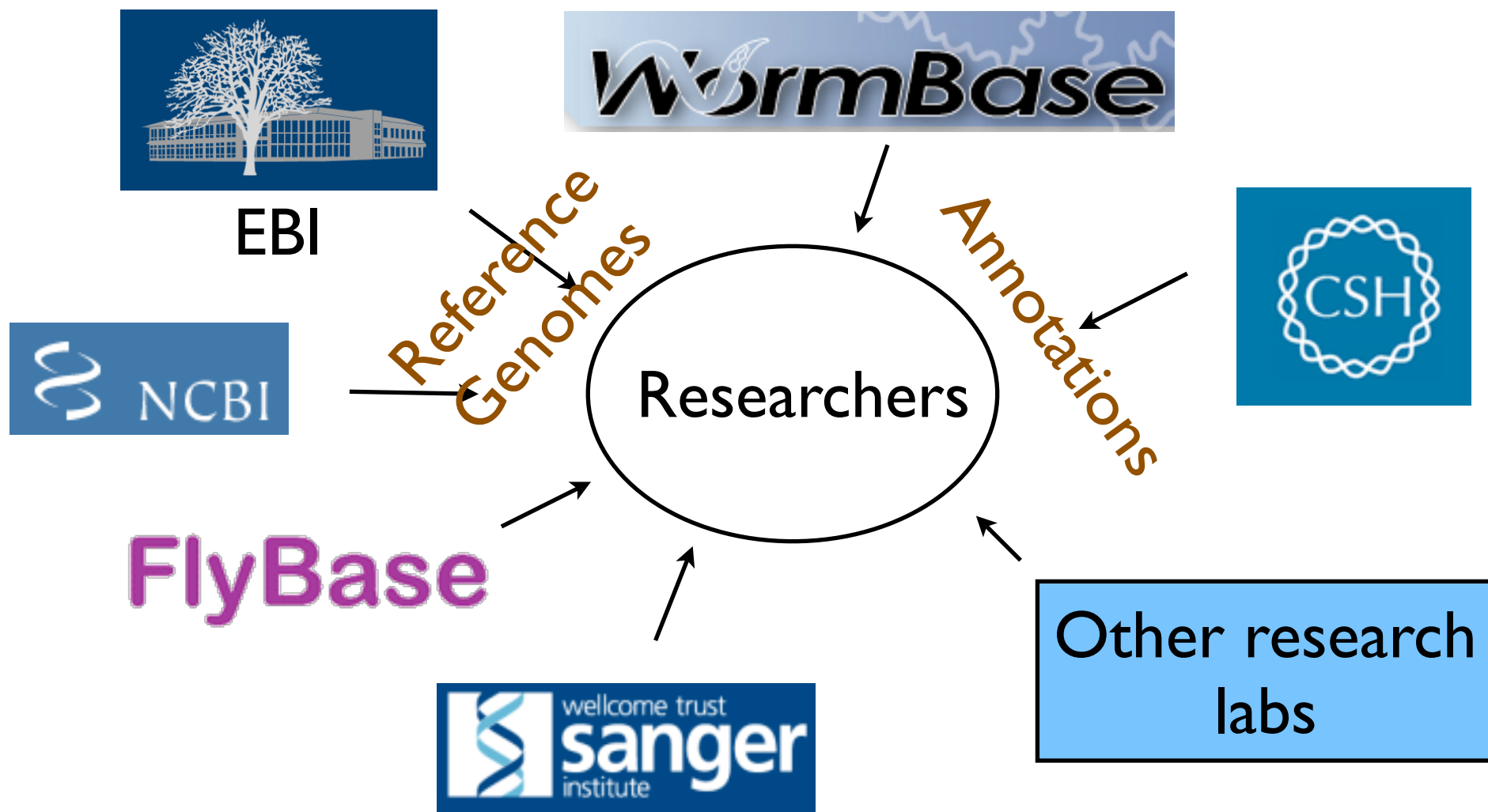
# Genomic data

*e!* **Ensembl Human**



# DAS

Distributed Annotation of Sequences - [biodas.org](http://biodas.org)



# SQLite

Most labs don't have computer experts - scale down!  
Setting up Oracle / MySQL / PostgreSQL / .. server is hard.

SQLite is a small C library that implements a self-contained, embeddable, zero-configuration SQL database engine. Features include:

- Transactions are atomic, consistent, isolated, and durable (ACID) even after system crashes and power failures.
- Zero-configuration – no setup or administration needed.
- Implements most of SQL92.
- A complete database is stored in a single disk file.
- Database files can be freely shared between machines with different byte orders.
- Supports databases up to 2 terabytes (2<sup>41</sup> bytes) in size.
- Sizes of strings and BLOBs limited only by available memory.
- ....

from [sqlite.org](http://sqlite.org) ...

Other research  
labs

# sqlite3 in Python 2.5

```
import sqlite3

conn = sqlite3.connect(":memory:")
c = conn.cursor()
c.execute("""CREATE TABLE Segments (
    id INTEGER PRIMARY KEY,
    name VARCHAR,
    length INTEGER)""")

c.execute("""CREATE TABLE Features (
    id INTEGER PRIMARY KEY,
    segment_id INTEGER,
    name VARCHAR,
    start INT,
    end INT)""")

c.execute("INSERT INTO Segments VALUES (1, 'Chr1', 247249719)")
counter = 100
for (name, start, end) in (("LAPTM5", 30977903, 31003254),
                          ("Q6ZRP8_HUMAN", 30963926, 30972178),
                          ("MATN1", 30956711, 30969474),):
    c.execute("INSERT INTO Features VALUES (?, ?, ?, ?, ?)", (
        counter, 1, name, start, end))
    counter += 1

# features overlapping 3097000 to 31000000
c.execute("""SELECT name FROM Features Where
           Features.start < 31000000 AND Features.end >= 30970000""")
for (name,) in c:
    print name
```

DB-API 2.0 compliant.  
Also supports SQLite extensions.

search yields

LAPTM5

Q6ZRP8\_HUMAN

# I am not a SQL programmer

apparently range searches are slow in SQL

```
# features overlapping 3097000 to 31000000  
c.execute("""SELECT name FROM Features Where  
Features.start < 31000000 AND Features.end >= 30970000""")
```



Hossein Rezazadeh

Searching 648,497 locations takes about 10 seconds on my laptop. Indices don't help.

Real SQL programmers use clever indexing schemes. Or PostgreSQL. I'll try brute strength.

# Pure Python

```
class Location(object):
    # __slots__ affects memory use and search time
    # w/o: 118MB (183B/loc), search in 1.46 s
    # w/ : 29MB (45B/loc), search in 0.92 s
    __slots__ = ["id", "start", "end"]
    def __init__(self, id, start, end):
        self.id, self.start, self.end = id, start, end

def overlaps(locations, start, end):
    for location in locations:
        if (start < location.end and
            end >= location.start :
            yield location
```



# Faster? Less Memory?



Python/C Extension

- by hand
- SWIG, Boost, SIP

Pyrex

Pysco

RPython (from PyPy)

web service

ctypes

# ctypes

a “Foreign Function Interface” for Python, new in 2.5

```
>>> from ctypes import *
>>> libc = CDLL("/usr/lib/libc.dylib", RTLD_GLOBAL)
>>> libc.strlen("Hello!")
6
>>> libc.strcmp("This", "That")
8
>>>
```



Cannot be `RTLD_LOCAL` (default)  
because Python has already loaded libc

# argument types

default interface only understands  
integer, string and pointer arguments

```
>>> libc.rinttol(0)
```

```
0
```

```
>>> libc.rinttol(1)
```

```
0
```

```
>>> libc.rinttol(1.5)
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
ctypes.ArgumentError: argument 1: <type 'exceptions.TypeError'>:
```

```
Don't know how to convert parameter 1
```

```
>>> libc.rinttol.argtypes = [c_float]
```

```
>>> libc.rinttol(0)
```

```
0
```

```
>>> libc.rinttol(1)
```

```
1
```

```
>>> libc.rinttol(1.5)
```

```
2
```

```
>>> libc.rinttol(2.5)
```

```
2
```

## Fundamental ctypes data types

c\_byte, c\_char, c\_char\_p, c\_double, c\_float,  
c\_int, c\_int{8,16,32,64}, c\_long, c\_longlong,  
c\_short, c\_ubyte, c\_uint, c\_uint{8,16,32,64},  
c\_ulong, c\_ulonglong, c\_ushort, c\_void\_p,  
c\_wchar, c\_wchar\_p, HRESULT, py\_object

# response type

The default response type is `c_int`

```
>>> libm = CDLL("/usr/lib/libm.dylib", RTLD_GLOBAL)
>>> libm.atan2.argtypes = [c_float, c_float]
>>> libm.atan2(1, 1)
-1877689272

>>> libm.atan2.restype = c_float
>>> libm.atan2(1, 1)
0.78539812564849854
>>> import math
>>> math.pi/4
0.78539816339744828
>>>
```

# Arrays

an array type is created using `(datatype) * length`  
arrays are instances of a given array type

```
>>> arr_t = c_int * 10
>>> arr = arr_t()
>>> arr
<__main__.c_long_Array_10 object at 0x538210>
>>> arr[0]
0
>>> arr[0] = 1
>>> arr[0]
1
>>> arr[10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: invalid index
>>>
```

If you need a resizing (list-like) array,  
see my “CTypesList” recipe in  
the Python Cookbook

# Complex Structures

```
from ctypes import *
```

```
class Location(Structure):  
    _fields_ = [("id", c_int),  
               ("start", c_int),  
               ("end", c_int)]
```

```
>>> location = Location(1, 100, 102)
```

```
>>> location.start, location.end  
(100, 102)
```

```
>>>
```

**Including arrays!**

```
>>> arr = (Location * 3)
```

```
>>> arr = (Location * 3)((1,100,102), (2,50, 130), (3,-40,12))
```

```
>>> arr[0].start, arr[1].id, arr[2].end
```

```
(100, 2, 12)
```

```
>>>
```

**Just need a bit of C code...**

# Pure C

```
typedef struct {  
    int id;  
    int start;  
    int end;  
} Location;
```

```
/* assumes hitlist has sufficient space */  
void overlaps(int num_locations, Location *locations,  
              int start, int end,  
              int *num_hits, int *hitlist) {  
    int i, n=0;  
  
    for (i=0; i<num_locations; i++, locations++) {  
        if (start < locations->end &&  
            end >= locations->start) {  
            n++;  
            *hitlist++ = i;  
        }  
    }  
    *num_hits = n;  
}
```

# setup.py

```
from distutils.core import setup, Extension

setup(name="location_search", version="0.0",
      ext_modules = [Extension("location_search", ["search.c"])]
    )
```

**setup.py builds arbitrary shared libraries.  
Not limited to Python extensions**

```
ln -s build/lib.darwin-7.9.0-Power_Macintosh-2.4/location_search.so .
```



# Does it work?

```
from ctypes import *
location_search = CDLL("./location_search.so")

class Location(Structure):
    _fields_ = [("id", c_int),
                ("start", c_int),
                ("end", c_int)]

locations = (Location * 3)((1,100,102), (2,50, 130), (3,-40,12))

hitlist = (c_int*3)()
nhits = c_int()

location_search.overlaps(len(locations), locations,
                          10, 60, byref(nhits), hitlist)

print nhits
for i in range(nhits.value):
    print locations[hitlist[i]].id
```

Use 'byref' to pass the &object

<p>c_long(2) 2 3</p>
------------------------------

# With type annotations

```
location_search = CDLL("./location_search.so")
```

```
location_search.overlaps.argtypes = [  
    c_int, POINTER(Location), c_int, c_int,  
    POINTER(c_int), POINTER(c_int) ]
```

```
location_search.overlaps.restype = None
```

... **ctypes automatically passes in the reference**

```
location_search.overlaps(len(locations), locations,  
                          10, 60, nhits, hitlist)
```

# Is it better?

memory required: 7,782,976 bytes

(best is 7,781,964; Python with `__slots__` = 29MB)

Test search took 0.026 seconds (0.92 in pure Python)

## Compiler-based extensions are still faster

```
>>> libm = CDLL("/usr/lib/libm.dylib", RTLD_GLOBAL)
>>> libm.cos.argtypes = [c_float]
>>> libm.cos.restype = c_float
>>> libm_cos = libm.cos
>>> import timeit
>>> timeit.Timer("cos(0.2)", "from __main__ import libm_cos as
cos").timeit()
5.1811199188232422
>>> timeit.Timer("cos(0.2)", "from math import cos").timeit()
1.9048159122467041
>>>
```

# Measuring memory use

from ctypes import \*                      No standard Python function.

```
class malloc_statistics_t(Structure):
    _fields_ = [ ("blocks_in_use", c_uint),
                 ("size_in_use", c_uint),
                 ("max_size_in_use", c_uint),
                 ("size_allocated", c_uint)]

libc = CDLL("/usr/lib/libc.dylib", RTLD_GLOBAL)
libc.malloc_zone_statistics.argtypes = [
    c_void_p, POINTER(malloc_statistics_t)]
libc.malloc_zone_statistics.restype = None

def memsize():
    stats = malloc_statistics_t()
    libc.malloc_zone_statistics(0, stats)
    return stats.size_in_use
```

Thanks to a c.l.py post  
by MrJeanI @ gmail.com  
with a C example

# Mommy?

## Where does data come from?

# Everywhere.

Entrez is a powerful [federated search](#) engine, or [web portal](#) that allows users to search a multiple of discrete [health sciences](#) databases, maintained by the [National Center for Biotechnology Information](#) with a single query string and user interface.

-wikipedia

# EUtils - Entrez web service

Query for “Dalke AND vmd”

[http://www.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?  
term=dalke+AND+vmd](http://www.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?term=dalke+AND+vmd)

Response is “Plain Old XML”

```
<?xml version="1.0"?>
<!DOCTYPE eSearchResult PUBLIC "-//NLM//DTD eSearchResult, 11 May 2002//EN"
"http://www.ncbi.nlm.nih.gov/entrez/query/DTD/eSearch_020511.dtd">
<eSearchResult>
  <Count>2</Count>
  <RetMax>2</RetMax>
  <RetStart>0</RetStart>
  <IdList>
    <Id>9390282</Id>
    <Id>8744570</Id>
  </IdList>
  <TranslationSet>
  </TranslationSet>
```

```
<TranslationStack>
  <TermSet>
    <Term>dalke[All Fields]</Term>
    <Field>All Fields</Field>
    <Count>38</Count>
    <Explode>Y</Explode>
  </TermSet>
  <TermSet>
    <Term>vmd[All Fields]</Term>
    <Field>All Fields</Field>
    <Count>100</Count>
    <Explode>Y</Explode>
  </TermSet>
  <OP>AND</OP>
</TranslationStack>
</eSearchResult>
```

# DAS2

Data stored in attributes, not in the elements' text

```
<?xml version="1.0" encoding="UTF-8"?>
<das:SEGMENTS xmlns:das="http://biodas.org/documents/das2">
  <das:FORMAT name="das2xml" />
  <das:FORMAT name="fasta" />
  <das:FORMAT name="genbank" />

  <das:SEGMENT uri="segment/chr1" title="Chromosome 1" length="246127941"
    reference="http://dalkescientific.com/human35v1/chr1"
    doc_href="http://www.ensembl.org/Homo_sapiens/mapview?chr=1" />
  <das:SEGMENT uri="segment/chr2" title="Chromosome 2" length="243615958"
    reference="http://dalkescientific.com/human35v1/chr2"
    doc_href="http://www.ensembl.org/Homo_sapiens/mapview?chr=2" />

  ....
</das:SEGMENTS>
```

# Reading XML

by hand - generally a bad idea

DOM - confusing

SAX - too low level

DTD → XML parser - fragile

## Various “pythonic” XML parsers

gnosis.xml.objectify, **ElementTree**, xmltramp

Implemented in: Python      C      part of lxml  
(uses libxml2, libxslt)

ElementTree is included with Python 2.5



# Basic Processing

```
<?xml version="1.0" encoding="UTF-8"?>
<das:SEGMENTS xmlns:das="http://biodas.org/documents/das2">
  <das:FORMAT name="das2xml" />
  <das:SEGMENT uri="segment/chr1" title="Chromosome 1" length="246127941"
    reference="http://dalkescientific.com/human35v1/chr1" />
</das:SEGMENTS>
```

```
>>> from xml.etree import ElementTree
>>> tree = ElementTree.parse("segments.xml")
>>> root = tree.getroot()
>>> root
<Element {http://biodas.org/documents/das2}SEGMENTS at 596300>
>>> root.tag
'{http://biodas.org/documents/das2}SEGMENTS'
>>>
```

Namespaces use {Clark}notation

# Children

```
<?xml version="1.0" encoding="UTF-8"?>
<das:SEGMENTS xmlns:das="http://biodas.org/documents/das2">
  <das:FORMAT name="das2xml" />
  <das:SEGMENT uri="segment/chr1" title="Chromosome 1" length="246127941"
    reference="http://dalkescientific.com/human35v1/chr1" />
</das:SEGMENTS>
```

```
>>> root
```

```
<Element {http://biodas.org/documents/das2}SEGMENTS at 596300>
```

```
>>> len(root)
```

```
2
```

```
>>> root[1]
```

```
<Element {http://biodas.org/documents/das2}SEGMENT at 5963f0>
```

```
>>> root[1].attrib
```

```
{'length': '246127941', 'uri': 'segment/chr1', 'reference': 'http://
dalkescientific.com/human35v1/chr1', 'title': 'Chromosome 1'}
```

```
>>> len(root[1])
```

```
0
```

```
>>>
```

# Searching

'find\*' methods in the tree and each node  
Uses a subset of XPath

```
>>> root = ElementTree.parse("ucla_segments.xml")
>>> for ele in root.findall(
    "{http://biodas.org/documents/das2}SEGMENT"):
...     print repr(ele.attrib["title"]), int(ele.attrib["length"])
...
'Chromosome 1' 246127941
'Chromosome 2' 243615958
'Chromosome 3' 199344050
'Chromosome 4' 191731959
'Chromosome 5' 181034922
...
...
...
```

# Text content

```
<?xml version="1.0"?>
<eSearchResult>
  <Count>2</Count>
  <RetMax>2</RetMax>
  <RetStart>0</RetStart>
  <IdList>
    <Id>9390282</Id>
    <Id>8744570</Id>
  </IdList>
</eSearchResult>
```


```
>>> tree = ElementTree.parse("esearch_small.xml")
>>> for hit in tree.findall("IdList/Id"):
...     print repr(hit.text)
...
'9390282'
'8744570'
>>>
```

# Iterative parsing

Entrez search of PubMed returns 900 records in 5MB  
Process a record at a time → feedback, less memory

```
>>> from cStringIO import StringIO
>>> f = StringIO("""<book><title>Heidi</title>
...     <author>Johanna Spyri</author></book>""")
>>>
>>> for (event, elem) in ElementTree.iterparse(f, ["start", "end"]):
...     print event, elem
...
start <Element book at 5969e0>
start <Element title at 596b48>
end <Element title at 596b48>
start <Element author at 596af8>
end <Element author at 596af8>
end <Element book at 5969e0>
>>>
```

Default is ["end"]  
Others events are  
"start-ns" and "end-ns"



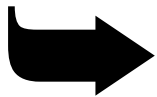
# Modify while iterating

```
def show_authors(count, article):
    print "Authors of article", count
    fields = []
    for author in article.findall(
        "MedlineCitation/Article/AuthorList/Author"):
        fields.append( (author.find("Initials").text + " " +
            author.find("LastName").text) )
    print " ", ", ", ".join(fields).encode("utf8")

filename = "/Users/dalke/ftps/sigpath/schema/ncbi-sample.xml"
counter = itertools.count(1)

for (event, ele) in ElementTree.iterparse(filename):
    if event == "end" and ele.tag == "PubmedArticle":
        show_authors(counter.next(), ele)
        ele.clear()
```

```
<Author>
<LastName>B<C3><B6>ttger</LastName>
<ForeName>B</ForeName>
<Initials>B</Initials>
</Author>
```



```
Authors of article 6
    CL Yee, R Yang, B Böttger, TE Finger, JC Kinnamon
```

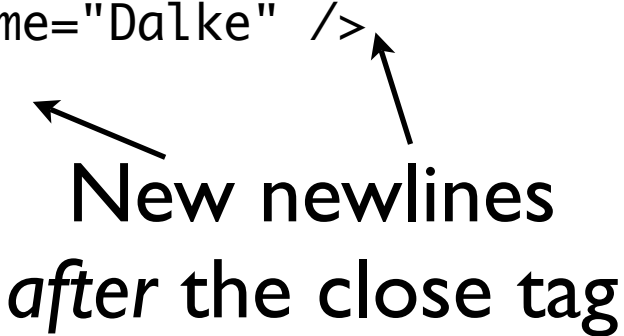
# Creating a tree

```
>>> root = ElementTree.Element("AuthorList")
>>> ElementTree.SubElement(root, "Author",
    {"forename": "Andrew", "surname": "Dalke"})
<Element Author at 58ffd0>
>>> ElementTree.SubElement(root, "Author",
    {"forname": "Craig", "surname": "Smith"})
<Element Author at 588698>
>>> del root[1]
>>> ElementTree.SubElement(root, "Author",
    {"forename": "Craig", "surname": "Smith"})
<Element Author at 588698>
>>>
```

# Writing XML

```
>>> tree = ElementTree.ElementTree(root)
>>> tree.write("/dev/tty")
<AuthorList><Author forename="Andrew" surname="Dalke" /><Author forename="Craig" surname="Smith" /></AuthorList>>>>
>>> root[0].tail = "\n"
>>> root[1].tail = "\n"
>>> tree.write("/dev/tty")
<AuthorList><Author forename="Andrew" surname="Dalke" />
<Author forename="Craig" surname="Smith" />
</AuthorList>>>>
>>> root.text="\n"
>>> root.tail = "\n"
>>> tree.write("/dev/tty")
<AuthorList>
<Author forename="Andrew" surname="Dalke" />
<Author forename="Craig" surname="Smith" />
</AuthorList>
>>>
```

**New newlines  
after the close tag**





```
from xml.sax import saxutils
```

# XMLGenerator

```
class Name(object):
```

```
    def __init__(self, forename, surname):
```

```
        self.forename = forename
```

```
        self.surname = surname
```

```
names = [Name("Andrew", "Dalke"), Name("Craig", "Smith")]
```

```
gen = saxutils.XMLGenerator(encoding="utf-8")
```

```
gen.startDocument()
```

```
gen.startElement("AuthorList", {})
```

```
gen.characters("\n")
```

```
for name in names:
```

```
    gen.startElement("Author",
```

```
        {"forename": name.forename, "surname": name.surname})
```

```
    gen.endElement("Author")
```

```
    gen.characters("\n")
```

```
gen.endElement("AuthorList")
```

```
gen.characters("\n")
```

```
gen.endDocument()
```

**Takes SAX2 events**

**I prefer this approach even though it is more error prone**

# from `__future__` import `with_statement`

```
class Document(object):
    def __init__(self, out=None, encoding="iso-8859-1"):
        self.gen = saxutils.XMLGenerator(out, encoding)
    def __enter__(self):
        self.gen.startDocument()
        return self.gen
    def __exit__(self, type, value, traceback):
        self.gen.endDocument()
        del self.gen
```

```
class Element(object):
    def __init__(self, gen, tag, d={}):
        self.gen, self.tag, self.d = gen, tag, d
    def __enter__(self):
        self.gen.startElement(self.tag, self.d)
    def __exit__(self, type, value, traceback):
        self.gen.endElement(self.tag)
```

```
class ElementNL(Element):
    def __exit__(self, type, value, traceback):
        Element.__exit__(self, type, value, traceback)
        self.gen.characters("\n")
```

**Use context generators  
to simplify generation**



# other modules

My abstract said I would talk about  
subprocess, csv and optparse

and might talk about  
bisect, heapq and textwrap

What time is it now?

# subprocess

Replaces `popen`, `system`, `popen2.*`, `commands.*`, `pipes.*`  
(Did anyone ever use the last two?)

Much easier to use. Does exactly what you want.  
(to within limits of the OS)

# CSV

```
for line in open(filename):  
    words = line.split("\t")  
    ...
```

The csv version is a  
bit more complex

```
for words in csv.reader(open(filename),  
                        quoting=csv.QUOTE_NONE, delimiter="\t"):  
    ...
```

## Clear advantage with quoted fields

```
>>> for row in csv.reader(  
...     ["'0'Higgins, Bernard',22-8-1778"],  
...     quotechar="'", doublequote=True):  
...     print row  
...  
["'0'Higgins, Bernard", '22-8-1778']  
>>>
```

Supports Excel dialects - for more complex Excel  
spreadsheets, try **pyExcelerator**

# optparse

Getopt is very easy to understand and start with.

```
args, filenames = getopt.getopt(sys.argv[1:],  
                                "f:q", ["file=", "quiet"])
```

Maintenance problems with larger programs. Aspects of a parameter are in separate locations.

```
parser = OptionParser()  
parser.add_option("-f", "--file", dest="filename",  
                 help="write report to FILE", metavar="FILE")  
parser.add_option("-q", "--quiet",  
                 action="store_false", dest="verbose", default=True,  
                 help="don't print status messages to stdout")  
  
(options, args) = parser.parse_args()
```

# Retrieving SWISS-PROT records

Common protein sequence data format.

A file contains many records.

Each record starts with “ID” followed by the id

```
ID  100K_RAT          STANDARD;          PRT;    889 AA.  
AC  Q62671;  
DT  01-NOV-1997 (Rel. 35, Created)  
DT  01-NOV-1997 (Rel. 35, Last sequence update)  
...
```

Given a record's ID, fetch the record  
Minimize complexity / scale down



# A fixed-width offset table

```
grep --byte-offset '^ID' sprot40.dat |  
  awk '{printf("%-10s%9d\n", $2,  
              substr($1, 1, length($1)-3))}' >  
  sprot40.offsets
```

20 bytes per line (including newline)

```
100K_RAT          0  
104K_THEPA       4568  
108_LYCES        7430  
10KD_VIGUN       9697  
110K_PLAKN      12169  
...  
ZYG_CHICK 320663597  
ZYG_HUMAN 320666761  
ZYG_MOUSE 320670231
```

Records were already  
in ID sorted order

How to find the start byte given an id?

# bisect

```
class Lookup(object):
    def __init__(self, infile):
        self.infile = infile
        self.infile.seek(0, 2) # seek to end
        n = self.infile.tell()
        assert n % 20 == 0
        self.num_records = n // 20

    def __len__(self):
        return self.num_records

    def __getitem__(self, i):
        self.infile.seek(20*i)
        return self.infile.read(20)

    def search(self, name):
        i = bisect.bisect_left(self, name)
        rec = self[i]
        if rec[:10] != name.ljust(10):
            raise KeyError(name)
        return rec
```

List-like lookup of  
fixed-size records



Implements a binary  
search algorithm

