Introduction to Python

Douglas Eck
University of Montreal

Much of this material adapted from Vitaly Shmatikov CS 345
Powerpoint slides found at:
www.cs.utexas.edu/~shmat/courses/cs345_spring08/21python.ppt.

Thanks Vitaly!

Quote of the Day

"Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered."

- Guido van Rossum

Python

- Object-oriented scripting language
  - "Multi-paradigm": imperative and functional features
  - Modules, classes, exceptions
- Interpreted, interactive
  - Facilitates rapid edit-test-debug development cycle
- Dynamically (but strongly) typed
- Extensible and portable
  - Built-in interfaces to many C libraries
  - Add functionality by writing new modules in C/C++
  - Runs on Unix, Mac, Windows…

Scripting Languages

- "Scripts" vs. "programs"
  - Interpreted vs. compiled
  - One script == a program
  - Many {*.c,*.h} files == a program

- Higher-level "glue" language
  - Put together program/library components
  - Orchestrate larger-grained computations

Running Python Interpreter

Interactive shell:

```
shell ~>python
Python 2.5 (r25:51908, Nov  6 2007, 15:55:44)
[GCC 4.1.2 20070925 (Red Hat 4.1.2-27)] on linux2
Type "help", "copyright", "credits" or "license" for
   more information.
>>> print 'Hello world'
Hello world
```

Files:

--contents of file helloworld.py--

```
!/usr/bin/env python %optional; for executable file
print 'Hello world'
```

```
shell ~>python helloworld.py
Hello world
shell ~>helloworld.py
Hello world
```

Ipython: Enhanced

http://ipython.scipy.org/moin/

Enhanced interactive python shell. Has command completion, command memory, help, etc. Strongly recommended.

```
341 eckdoug@fringant ~>ipython
Python 2.5 (r25:51908, Nov  6 2007, 15:55:44)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.1 -- An enhanced Interactive Python.
?       -> Introduction to IPython's features.
%magic  -> Information about IPython's 'magic' % functions.
help    -> Python's own help system.
object? -> Details about 'object'. ?object also works, ?? prints more.
```

Python Help

```
In [1]: help()

Welcome to Python 2.5!  This is the online help utility.

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://www.python.org/doc/tut/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules",
"keywords", or "topics".  Each module also comes with a one-line summary
of what it does; to list the modules whose summaries contain a given word
such as "spam", type "modules spam".
```

Keywords and Modules

```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

and                 else                import              raise
assert              except              in                  return
break               exec                is                  try
class               finally             lambda              while
continue            for                 not                 yield
def                 from                or
del                 global              pass
elif                if                  print
```

Built-in Types

- Numbers
  - Integers: C "long" and arbitrary precision
    - Decimal, decimal long, octal and hex literals
  - Boolean: True, False and bool(x) -> bool
  - Floating point: C "double"
  - Complex numbers
- Sequences
  - String, List, Tuple
- Dictionary (association list)
- File
- Matrix and Array (via Numpy)

Integers

```
>>> int
<type 'int'>
>>> long
<type 'long'>
>>> 1024
1024
>>> 99999999999999999999999999999999
99999999999999999999999999999999L
>>> int("12345")
12345
>>> long("12345")
12345L
>>> int(0xff)
255
>>> a=100
>>> b=200
>>> a+1, b+2
(101, 202)     # this is a tuple
>>> print a+1, b+2
101 202
>>> int(3.14), abs(-12), (2*1024/4), 8%2, 8/2)
(3, 12, 512, 0, 4)
>>> int('0100',10), int('0100', 8), int('0x40',16)
(100, 64, 64)
>>> float(3), 2**(2+2), pow(2,4)
(3.0, 16, 16)
```

Arithmetic Operators

```
>>> help("int")
int(x [, base]) -> integer
x.__abs__() <==> abs(x)
x.__add__(y) <==> x+y
x.__and__(y) <==> x&y
x.__cmp__(y) <==> cmp(x,y)
x.__coerce__(y) <==> coerce(x, y)
x.__div__(y) <==> x/y
x.__divmod__(y) <==> xdivmod(x, y)y
x.__float__() <==> float(x)
x.__floordiv__(y) <==> x//y
x.__getattribute__('name') <==> x.name
x.__hash__() <==> hash(x)
x.__hex__() <==> hex(x)
x.__int__() <==> int(x)
x.__invert__() <==> ~x
x.__long__() <==> long(x)
x.__lshift__(y) <==> x<<y
x.__mod__(y) <==> x%y
x.__mul__(y) <==> x*y
x.__neg__() <==> -x
x.__nonzero__() <==> x != 0
x.__oct__() <==> oct(x)
x.__or__(y) <==> x|y
x.__pos__() <==> +x
x.__pow__(y[, z]) <==> pow(x, y[, z])
x.__radd__(y) <==> y+x
x.__rand__(y) <==> y&x
x.__rdiv__(y) <==> y/x
x.__rdivmod__(y) <==> ydivmod(y, x)x
x.__repr__() <==> repr(x)
x.__rfloordiv__(y) <==> y//x
x.__rlshift__(y) <==> y<<x
x.__rmod__(y) <==> y%x
x.__rmul__(y) <==> y*x
x.__ror__(y) <==> y|x
y.__rpow__(x[, z]) <==> pow(x, y[, z])
x.__rrshift__(y) <==> y>>x
x.__rshift__(y) <==> x>>y
x.__rsub__(y) <==> y-x
x.__rtruediv__(y) <==> y/x
x.__rxor__(y) <==> y^x
x.__str__() <==> str(x)
x.__sub__(y) <==> x-y
x.__truediv__(y) <==> x/y
x.__xor__(y) <==> x^y
```

Booleans

```
>>> bool
<type 'bool'>
>>> True, False
(True, False)
>>> bool(0), bool(1), bool(-1)
(False, True, True)
>>> bool(3.14), bool("hello, world")
(True, True)
>>> True & True, True & False, False & False
(True, False, False)
>>> True | True, True | False, False | False
(True, True, False)
>>> True ^ True, True ^ False, False ^ False
(False, True, False)
>>> str(True), repr(True)
('True', 'True')
>>> print False
False
>>> print True
True
```

Floating Point

```
>>> float
<type 'float'>
>>> 3.14
3.1400000000000001
>>> print 3.14
3.14
>>> repr(3.14)
'3.1400000000000001'
>>> str(3.14)
'3.14'
>>> 3.14//2, 3.14/2
(1.0, 1.5700000000000001)
>>> import math
>>>> math.pi, math.e
(3.1415926535897931, 2.7182818284590451)

>>> help ("math")   # check it out to see what functions are available
```

Strings

String = immutable sequence of characters

```
>>> help (“str”)  # check out help to learn more
>>> str.upper("abcdefghijklmnopqrstuvwxyz")
'ABCDEFGHIJKLMNOPQRSTUVWXYZ’
>>> ””   # empty string
''
>>> print "'"
'
>>> print '"'
"
>>> "a %s parrot" % 'dead’  # inline string formatting
'a dead parrot’
>>> "monty " + 'python'
'monty python’
>>> 'm' in 'monty python'
True
>>> 'm' in "Monty Python"
False
>>> s=“monty python”
>>>len(s), s.find(‘py’), s.split()
(12, 6,  ['monty', 'python'])
>>> s[0],s[1],s[2:3],s[4:]
('m', 'o', 'n', 'y python’)
>>> s[6:99]
‘python’
```

Lists

List = mutable sequence

```
>>> help (“list”)  # check out help to learn more
>>> []   #empty list
[]
>>> x = [1, "this", 3.14]
>>> x
[1, 'this', 3.1400000000000001]
>>> len(x)
3
>>>  for i in x:   # iterate over the list x
...     print i
1
this
3.14
>>> x + ['a', 2, []]
[1, 'this', 3.1400000000000001, 'a', 2, []]
>>> x.append(5)
>>> x.reverse()
>>> x
[5, 3.1400000000000001, 'this', 1]
>>> x*2
[5, 3.1400000000000001, 'this', 1, 5, 3.1400000000000001, 'this', 1]
>>> x.sort()
>>> x
[1, 3.1400000000000001, 5, 'this']
```

Tuples

Tuple = immutable sequence

```
>>> help (“tuple”)  # check out help to learn more
>>> ()   #empty tuple
()
>>> x = (1,'a','bcd')
>>> ‘a’ in x
True
>>> x[0],x[1],x[:-1]
(1, 'a', (1, 'a'))
>>> (1,2) + (3,4)
(1, 2, 3, 4)
>>> (1,2)*2
(1, 2, 1, 2)
>>> y = (20)   # not a 1 item tuple!
>>> y
20
>>> y = (20,)
>>> y
(20,)
>>> y=(20,15,9,1,-5)
>>> tmp = list(y)   # convert tuple to a list
>>> tmp.sort()
>>> y=tuple(tmp) # convert sorted list back to a tuple
>>> y
(-5, 1, 9, 15, 20)
```

Dictionaries

Dictionary = (unordered) set of <key,value> pairs

```
In [37]: d = {'spam' : 1, 'ham' : 2, 'eggs' : 3 }

In [38]: d.values()
Out[38]: [3, 2, 1]

In [39]: d.keys()
Out[39]: ['eggs', 'ham', 'spam']

In [54]: d.has_key('ham')
Out[54]: True

In [52]: d.get('bacon',-1)
Out[52]: -1

In [53]: d.get('ham',-1)
Out[53]: 2
```

Files

```
>>> help (“file”)  # check out help to learn more
>>> output = open(‘/tmp/test’, ‘w’)  # open tmp file for writing
>>> input = open(‘/etc/passwd’, ‘r’) # open file for reading
>>> s = input.read()          # read entire file into string s
>>> s = input.readline()      # read next line
>>> l = input.readlines()     # read entire file into list of line strings
>>> output.write(s)           # write string s
>>> output.write(l)           # write all lines in l
>>> output.close()
>>> input.close()
```

```
#You can iterate over a file object
#This example prints every line in a file
fh = open(‘/etc/passwd’,’r’)
for l in fh :
    print l.strip()   #strip() removes leading and trailing whitespace
fh.close()
```

Matrix / Array

```
from numpy import *
x=array([[1,2,3,4,5],[6,7,8,9,10]])
In [4]: x
Out[4]:
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
In [5]: type(x)
Out[5]: <type 'numpy.ndarray'>
In [6]: y = x * 1.2
In [7]: y
Out[7]:
array([[  1.2,   2.4,   3.6,   4.8,   6. ],
       [  7.2,   8.4,   9.6,  10.8,  12. ]])
In [9]: x.shape
Out[9]: (2, 5)
In [10]: y=random.random((5,4))
In [11]: dot(x,y)
Out[11]:
array([[  6.91690408,   3.26093504,  10.50287798,  10.49733007],
       [ 17.70706734,   9.52690047,  29.55506377,  27.36516467]])
```

The matrix and array types are provided by numpy.

Beware! Unlike matlab, matrix and array are distinct types.

This and other details will be discussed in a separate numpy/ scipy tutorial

Typing / Safety

- Variables are dynamically typed
- Variables are strongly typed
- E.g cannot do string operations on numbers or vice versa

```
In [1]: x=10
In [2]: y='twenty'
In [3]: type(x)
Out[3]: <type 'int'>
In [4]: type(y)
Out[4]: <type 'str'>
In [5]: x+y
---------------------------------------------------------------------------
<type 'exceptions.TypeError'>                Traceback (most recent call last)
/u/eckdoug/<ipython console> in <module>()
<type 'exceptions.TypeError'>: unsupported operand type(s) for +: 'int' and 'str'
```

Typing / Safety

- Cannot index past end of list
- Must append to end of a list
- Variables must be set before being referenced

```
In [17]: x=['a','b','c']

In [18]: x[10]
---------------------------------------------------------------------------
<type 'exceptions.IndexError'>                Traceback (most recent call last)

/u/eckdoug/<ipython console> in <module>()

<type 'exceptions.IndexError'>: list index out of range
```

Assignment Forms

- Basic form
  - `a = 'spam'`
- Tuple positional assignment
  - `a, b = 2, 'ham'`
  - `a == 2, b == 'ham' => (True, True)`
- List positional assignment
  - `[a,b] = [3,'eggs']`
  - `a==3, b=='eggs' => (True, True)`
- Multi-assignment
  - `a = b = 10`

Blocks and Indentation

- Block structure determined by indentation
  - No begin … end or { … }
  - To place statements in the same block, must indent them at the same level
  - Example:

```
if n == 0:
    …… return 1
else:
    …… return n*fact(n-1)
```

Conditional Statement

```
>>> if 1:
... print True
  File "<stdin>", line 2
    print True
        ^
IndentationError: expected an indented block
>>> if 1:
...     print True
...
True
>>> choice='ham'
>>> if choice=='spam':
...     print 1
... elif choice=='ham':
...     print 2
... elif choice=='eggs':
...     print 3
... else:
...     print "None"
...
2
```

“While” Loops

```
>>> x='spam'
>>> while x:
...     print x,
...     x=x[1:]
...
spam pam am m

>>> a=0; b=10
>>> while a < b:
...     print a,
...     a += 1
...
0 1 2 3 4 5 6 7 8 9
```

“For” Loops (1)

```
In [13]: range(5)
Out[13]: [0, 1, 2, 3, 4]

In [14]: for x in range(5) :
   ....:     print x
   ....:
0
1
2
3
4

In [16]: x=['a','b','c']

In [17]: for i in x :
   ....:     print i
   ....:
a
b
c
```

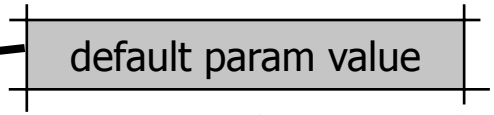
“For” Loops (2)

```
In [74]: a=['a','b','c']
In [75]: b=[1,2,3]
In [76]: for (i,j) in zip(a,b) :
   ....:     print i,j
a 1
b 2
c 3

In [78]: for (ctr,val) in enumerate(a) :
   ....:     print ctr,val
0 a
1 b
2 c

In [85]: for i in range(0,100,10) :
    print i,
   ....:
0 10 20 30 40 50 60 70 80 90

In [82]: for i in range(5,-1,-1) :  #help range for more
    print i,
   ....:
5 4 3 2 1 0
```

Functions

```
>>> def mul(x,y): return x * y
...

>>> mul(3,4)
12

>>> mul(math.pi,2.0)
6.2831853071795862

>>> mul([1,2],2)
[1, 2, 1, 2]

>>> mul("boo", 5)
'booboobooboobooʼ

>>> def fact(n):
...     m=1
...     while n > 0: m*=n; n-=1
...     return m
...
>>> fact(5)
120
```

A more complex function




```
def wordfreq(fn, K=20) :
    '''prints top K most frequent words in text file fn'''
    fh=open(fn,'r')

    #make a dictionary with key=word, val=count
    word2count=dict()
    for l in fh :
        words = l.strip().lower().split(' ') #split into words
        for w in words :
            if len(w)>0 :
                count = word2count.get(w,0)
                word2count[w]=count+1

    #make a dictionary with key=count, val = list of words
    count2word=dict()
    for (k,v) in word2count.iteritems() :
        wordlist = count2word.get(v,list())
        wordlist.append(k)
        count2word[v]=wordlist
    keys=count2word.keys()
    keys.sort()       #sort
    keys=keys[::-1]   #invert
    keys=keys[0:K]    #take first K keys
    for (ctr,key) in enumerate(keys) :
        print '%i %s appears %i times' % (ctr+1,count2word[key],key) #print
    fh.close()
```

Calling from command line

will run from command line in linux/unix using `wordfreq.py` instead of `python wordfreq.py`

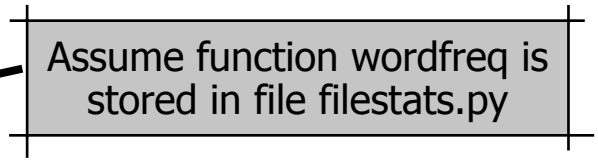
```
#!/usr/bin/env python
import sys

def wordfreq(fn, K=20) :
    '''prints top K most frequent words in text file fn'''
    fh=open(fn,'r')

    #----shortened-----

    fh.close()

def die_with_usage() :
    print 'wordfreq.py <file> prints the top K most frequent words in <file>'
    sys.exit(0)
```

package "sys" provides system functions including argv (command line arguments)

This block only executes when called from command line

```
if __name__=='__main__' :
    if len(sys.argv)<=1 :
        die_with_usage()
    wordfreq(sys.argv[1])
```

Importing packages and functions




```
In [4]: from filestats import wordfreq
In [5]: wordfreq('great_beagle.txt',4)
1 ['the'] appears 24903 times
2 ['of'] appears 13757 times
3 ['and'] appears 12438 times
4 ['a'] appears 9241 times


In [6]: import filestats as FS
In [7]: FS.wordfreq('great_beagle.txt',4)
1 ['the'] appears 24903 times
2 ['of'] appears 13757 times
3 ['and'] appears 12438 times
4 ['a'] appears 9241 times

In [8]: from filestats import *
.....
```

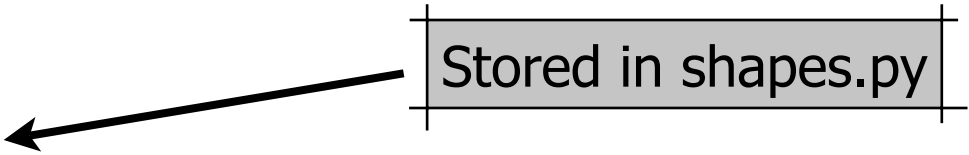
Examples using default args

```
def f(a=-1, b=-1, c=-1) :
    print 'a=%s b=%s c=%s' % (str(a),str(b),str(c))

In [9]: f()
a=-1 b=-1 c=-1

In [11]: f(c=200,a=23)
a=23 b=-1 c=200
```

```
def stringshuffle(s, doWord=False) :
    '''shuffles a string;
       If doWord is true, shuffles individual words instead of entire string'''
    import numpy.random as R
    if doWord :
        phrase = s.split(' ')
    else :
        phrase = [s]
    for (ctr,w) in enumerate(phrase) :
        l = list(w)
        R.shuffle(l)
        phrase[ctr]=string.join(l,'')

    return string.join(phrase,' ')
```

Object Features

- Object-oriented features
  - All methods and instance variables are public
  - Multiple inheritance




```
class Point:
    '''A point'''
    def __init__(self, x, y): # constructor
        '''Set name of person.'''
        self.x = x
        self.y = y

    def position(self) :      # a method
        '''Return position'''
        return [self.x,self.y]

    def __repr__(self) :      # objects string representation
        return 'Point <x=%2.1f, y=%2.1f>' % (self.x,self.y)
```

Using a class

```
In [5]: import shapes
In [6]: help shapes.Point
Help on class Point in module shapes:

class Point
 |  A point
 |
 |  Methods defined here:
 |
 |  __init__(self, x, y)
 |      Set name of person.
 |
 |  __repr__(self)
 |
 |  position(self)
 |      Return position

In [7]: x=shapes.Point(10,20)
In [8]: print x
Point <x=10.0, y=20.0>
In [9]: x.position()
Out[9]: [10, 20]
```

Contents of shapes.py :

```
class Point:
    '''A point'''
    def __init__(self, x, y):
        '''Set name of person.'''
        self.x = x
        self.y = y

    def position(self) :
        '''Return position'''
        return [self.x,self.y]

    def __repr__(self) :
        return 'Point <x=%2.1f, y=%2.1f>'
                % (self.x,self.y)
```

Inheritance

```
import math

class Point
    '''A point'''
    def __init__(self, x, y): # == C++ constructor
        '''Set name of person.'''
        self.x = x
        self.y = y

    def position(self) :
        '''Return position'''
        return [self.x,self.y]

    def __repr__(self) :
        return 'Point <x=%2.1f, y=%2.1f>' % (self.x,self.y)

class Circle(Point) :                    #class Circle defined as subclass of Point
    '''A circle'''
    def __init__(self,x,y,r) :
        Point.__init__(self,x,y)         #constructor for super-class Point
        self.r=r

    def area(self) :
        '''Return area'''
        return 2.0 * math.pi * self.r  #note the use of value “pi” from package math
    def __repr__(self) :
        return 'Circle <x=%2.1f, y=%2.1f, r=%2.1f>' % (self.x,self.y,self.r)
```

lambda, map, reduce, filter

```
>>> mul = lambda x, y: x * y
>>> mul
<function <lambda> at 0x4788b0>
>>> apply(mul, (2,3))
6
```

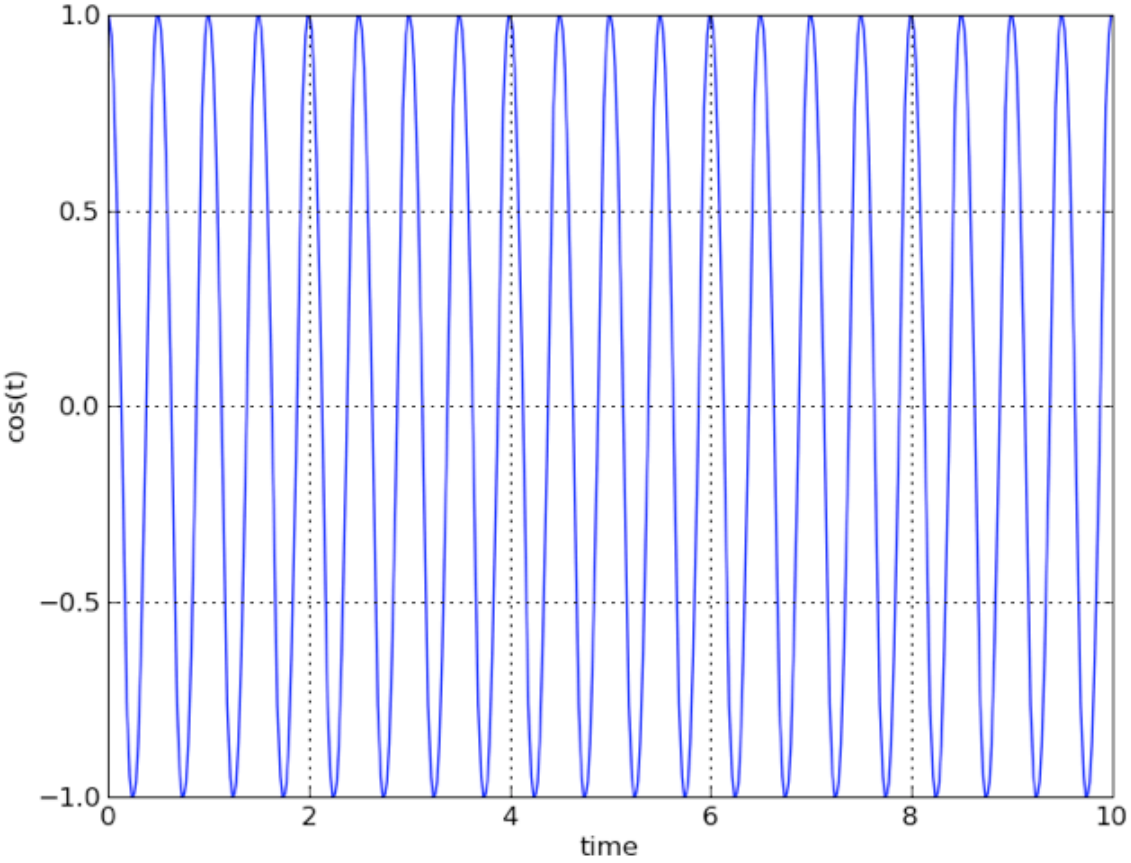
```
>>> map (lambda x:x*x, [1,2,3,4,5])
[1, 4, 9, 16, 25]
>>> map (string.lower,['Hello','Goodbye']
[hello, goodbye]
```

```
>>> reduce (lambda x,y: x+y, [1,2,3,4,5])
15
>>> reduce (lambda x,y:x*y, [1,2,3,4,5])
120
>>> def fact(n): return reduce (lambda x,y:x*y, range(1,n+1))
...
>>> fact(5)
120
```

```
>>> filter (lambda x:x>0, [-5,-4,-3,-2,-1,0,1,2,3,4,5])
[1, 2, 3, 4, 5]
```

Plotting

```
from numpy import *
import pylab
In [1]: secs=10.0;fs=1000; t=arange(fs*secs)/fs; freq=2;
In [2]: pylab.plot(t,cos(t*2*pi*freq))
In [3]: pylab.ylabel('cos(t)'); pylab.xlabel('time')
In [4]: pylab.grid('on')
#With ipython, this is enough; with python add: pylab.show()
```

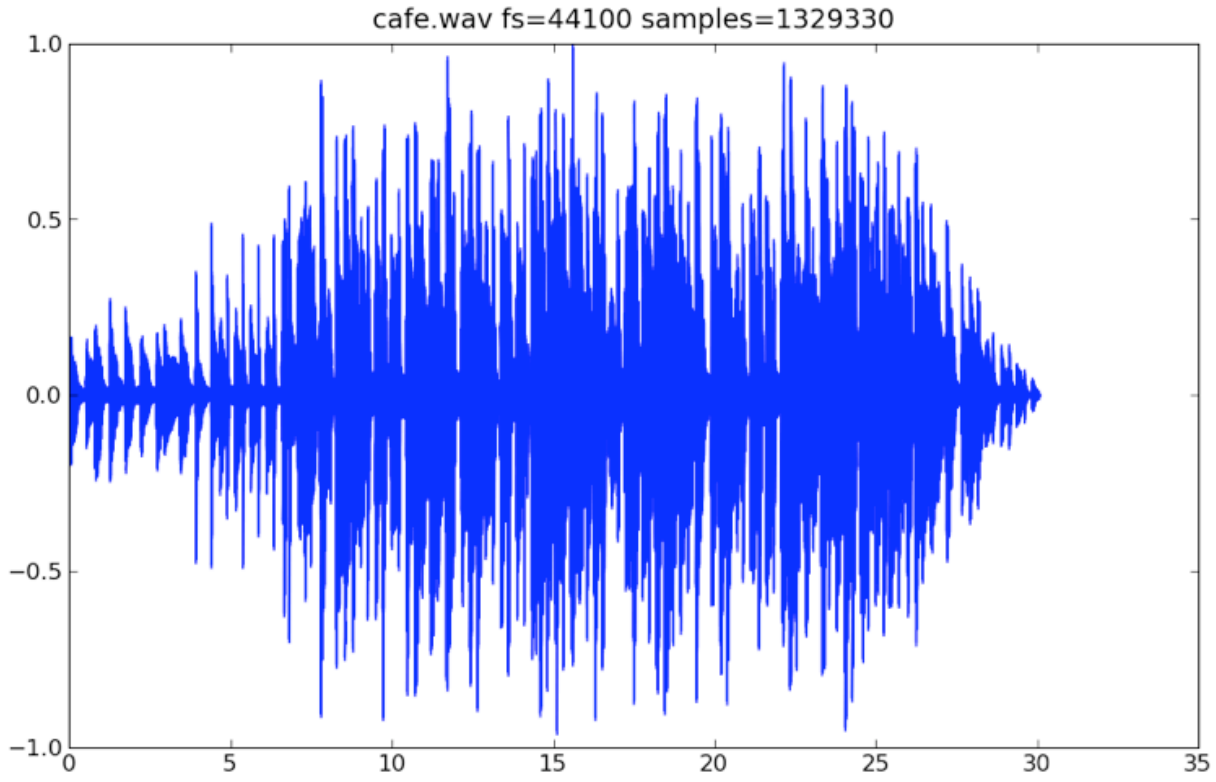


Plotting is provided by matplotlib (a.k.a. pylab)

More details given in a later plotting tutorial

Read Wave File

```
import scipy.io.wavfile as S
import pylab as P
fn='cafe.wav'
(fs,x)=S.read('cafe.wav')
x=x/32768.0  #normalize between -1 and 1 rather than as PCM 16-bit values
t=arange(len(x))/float(fs)
P.plot(t,x)
P.title('%s fs=%i samples=%i' % (fn,fs,len(x)))
```



Write Wave File

```
import scipy.io.wavfile as S
secs=10.0;fs=44100; t=arange(fs*secs)/fs; freq=440;
fn='a440.wav'
x=cos(t*2*pi*freq) * .7          #attenuate the signal (1s make clicks)
xout=(x*32768.0).astype(int16)   #write out as PCM 16-bit ints
S.write(fn,fs,xout)


#This creates a wav file 10sec long containing 440hz pure sinusoid
```

- Primitive wave I/O in python wave module
- Better I/O supported by scipy.io.wave
- Even better wave support from scikits.audiolab
- More details given in a later plotting tutorial