

Overview of Python

Python and PyMC Workshop

Michael J. Conroy

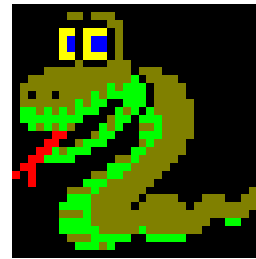
USGS/ University of Georgia

What is Python?

- Programming language that is
 - Open-source
 - Object-oriented
 - Interpreted scripting language for C
- Similar languages:
 - Java
 - Perl
 - Ruby

Why Python?

- Easy to use
- Free
 - = open
 - = no cost
- Powerful and flexible
- Object oriented
- A real programming language **Binomial.py**
- Third party modules (some written in FORTRAN, C, etc.)
- We miss Monty Python



Modules include

- Numpy
 - Specify and manipulate array data
 - Linear algebra
- Scipy
 - Supplements Python Numeric
 - Features of Numpy plus
 - Optimization
 - Statistical distribution methods
 - Integration
 - Genetic algorithms
 - Many others

Jargon busting

- Object- any data stored. Includes:
 - Data types (numbers, strings, lists, etc.)
 - Classes & functions (user-defined, built in)
- Type- internal representation of object + method and operations it supports
 - Mutable- can be changed
 - Immutable- cannot be changed
- Attribute- property or value associated with object
- Method- function that performs an operation on an object when invoked
- Class- group of functions/methods
- Module-collection of classes/definitions saved in a .py file
- Exceptions- error raised by program
 - Built in
 - User defined

Basic programming

- Pass data as objects
- Use indentation blocks to define functions, classes, loops, etc
- Take advantage of object oriented nature
 - member-objects inherit attributes of objects of which they are a member
- Take advantage
 - User defined methods, classes
 - Importing of user-defined or 3rd party modules

Simple example

```
# Some data, in a list
```

```
my_data = [12,5,17,8,9,11,21]
```

Data object

```
# Function for calculating the mean of some data
```

```
def mean(data):
```

```
# Initialize sum to zero
```

function

```
    sum_x = 0.0
```

```
# Loop over data
```

```
    for x in data:
```

Method and loop definition by indentation

```
        # Add to sum
```

```
        sum_x += x
```

```
        # Divide by number of elements in list, and return
```

```
        return sum_x/len(data)
```

```
mean(my_data)
```

Invoking the function

Running python

Starting python

- Interactively using Python command line
 - Type in commands one by one
 - Import modules and run methods interactively
- Run script from an editor (e.g., IDLE)
 - Reserved/ key words colourized
 - Good for debugging
- Use Python shell in editor
- Point and click python script (compiles and runs)

From command line

- MS-DOS command (set to working directory)
- Type 'python'
- At prompt enter python commands

```
>>>a=4**2+5
```

```
>>>a
```

```
21
```

RESERVED!

- and
- assert
- break
- class
- continue
- def
- del
- elif
- else
- except
- exec
- finally
- for
- from

..more reserved

- global
- if
- import
- in
- is
- lambda
- not
- or
- pass
- print
- raise
- return
- try
- while

Operators & delimiters

- Basic math: `+`, `-`, `*`, `**`, `/` (with numeric objects)
- Incrementing: `+=`, `-=`, `*=`, `**=`, `/=`
- Comparison: `==`, `>`, `<`, `>=`, `<=`, `!=`, `<>`
- Strings, lists, tuples:
 - Concatenate, copy: `+`, `*`
 - String formatting: `%`
 - Slicing: `[:]`

Python types

- Numbers- integers, long integers, floating, complex
- Sequences- strings, tuples, lists
- Mapping objects- dictionaries
- Callable objects- functions, methods, classes
- Modules
- Files

Assignments & Mutability

- When program makes assignment `a=b` creates new reference
- Immutable objects:
 - Creates a copy of `b`
e.g. `b=(1,2,3,4)`
`a=b`
`print a` → (1,2,3,4)
- Mutable objects:
 - Eg. `b=[1,2,3,4]`
`a=b`
`a[1]=20`
`print b` → [1,20,3,4] **CHANGES THE ORIGINAL OBJECT**

Literals

```
#python debugger
>>import pdb
>>from numpy import *
#some types
>>print 42 #integer
42
>>print 'what is the answer to the meaning of the universe?' #string
what is the answer to the meaning of the universe?
>>print 67.0J #imaginary
67j
>>print 67.0J+1. #complex
(1+67j)
>>print """string over 2
>>lines""""
string over 2
lines
```

Tuples

```
#tuples
```

```
>>a=(34,45,90,100,'y')
```

```
>>print 'a=',tuple_
```

```
a= (34, 45, 90, 100, 'y')
```

```
#slicing
```

```
>>print 'fifth element of a=', a[4]
```

```
fifth element of a =y
```

```
>>print 'last 2 elements of tuple', a[-2:]
```

```
last 2 elements of tuple (100, 'y')
```

```
>>print 'all but last 2 elements of tuple', a[:-2]
```

```
all but last 2 elements of tuple (34, 45, 90)
```

```
>>try:
```

```
>> a[4]=5
```

```
>>except TypeError:
```

```
>> print "You can't do this because tuples are immutable!"
```

← Good use of error exception!

You can't do this because tuples are immutable!

Lists

```
#lists
```

```
>>a=[34,45,90,100,'y']
```

```
>>b=a
```

```
>>print 'a=',a
```

```
a =[34, 45, 90, 100, 'y']
```

```
>>b[2]='r'
```

```
>>print 'lists are mutable'
```

```
lists are mutable
```

```
>>print 'a=',a
```

```
a= [34, 45, 'r', 100, 'y']
```

List expansion

```
#expanding lists
```

```
>>print [3,2]
```

```
>>'print expanded lists'
```

```
expanded lists
```

```
>>print [3,2]*3
```

```
[3, 2, 3, 2, 3, 2]
```

```
>>print [3,2]+[4]
```

```
[3, 2, 4]
```

List methods

```
#list methods
```

```
>>print 'list methods'
```

```
>>a=[1,2,3,4,5,6,7,8,9]
```

```
>>print a
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
#append x to end of a
```

```
>>x=4
```

```
>>a.append(x)
```

```
>>print a
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 4]
```

```
>>b=[20,30,40]
```

```
>>a.extend(b)
```

```
>>print a
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 4, 20, 30, 40]
```

More list methods

```
>>print a.count(4)
```

```
2
```

```
>>a.remove(4)
```

```
>>print a
```

```
[1, 2, 3, 5, 6, 7, 8, 9, 4, 20, 30, 40]
```

```
>>a.sort()
```

```
>>print a
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 20, 30, 40]
```

```
>>a.reverse()
```

```
>>print a
```

```
[40, 30, 20, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Removed

First occurrence of 4



Strings

- Sequences of characters indexed by integers 0, 1, 2,.....

```
>>a='Now is the Time'
```

```
>>print a
```

```
Now is the Time
```

```
>>#slicing
```

```
>>print a[1:5]
```

```
ow i
```

```
>>print 'error exception if index not in string'
```

(some)String Methods

```
>>#split
>>lineofdata='1234,567,8910'
>>print 'Split the line at the commas'
>>x,y,z=lineofdata.split(",")
>>print x,y,z
1234 567 8910
```

Dictionaries

- Array of objects indexed by keys
- Create in { }
- Access by []
- Not sorted by keys (but can be sorted)

Dictionaries

#dictionaries

#empty dictionary

```
>>wife={ }
```

#filled dictionary

```
>>wife={'fred':'wilma', 'barney':'betty' }
```

```
>>print wife
```

```
{'barney': 'betty', 'fred': 'wilma' }
```

```
>>print wife['fred']
```

```
wilma
```

#iterate over keys and get entry

```
>>for hubby in wife:
```

```
>>    print hubby,wife[hubby]
```

```
barney betty
```

```
fred wilma
```

Loops, controls

- Conditionals
 - if, elif, else
- Loops
 - for, while
- Exceptions
 - try, except

Conditionals

```
>>x=2
>>if x<2:
>>    print 'x is less than 2'
>>elif x>2:
>>    print 'x>2'
>>else:
>>    print 'x=2'
x=2
```

loops

```
>>x=0
>>while x<11:
>>  print 'x=',x, 'x**2=', x**2
>>  x+=1
x= 0 x**2= 0
x= 1 x**2= 1
x= 2 x**2= 4
x= 3 x**2= 9
x= 4 x**2= 16
x= 5 x**2= 25
x= 6 x**2= 36
x= 7 x**2= 49
x= 8 x**2= 64
x= 9 x**2= 81
x= 10 x**2= 100
```

Same result from

```
for x in range(11):
    print x, x**2
```

List comprehension

- Shortcut way of creating and manipulating lists in 1 step

```
#list comprehension
```

```
>>print 'list comprehension'
```

```
>>x=[i**2 for i in range(10)]
```

```
>>print x
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
#can operate on existing list
```

```
>>z=[1j*y for y in x]
```

```
>>print z
```

```
[0j, 1j, 4j, 9j, 16j, 25j, 36j, 49j, 64j, 81j]
```

List comprehension

- Combine with other list methods for more complicated problems

Eg. Sum from $x=0, \dots, 99$, $x+x^{**2}+\text{sqrt}(x)$

Method 1:

```
#use zip to put together 3 lists and sum them
```

```
>>a=arange(0,100)
```

```
>>b=[x**2 for x in a]
```

```
>>c=[sqrt(x) for x in a]
```

```
>>d=sum([x+y+z for x,y,z in zip(a,b,c)])
```

```
print d
```

```
333961.462947
```

Method 2 (easier for this case):

```
>>d=sum([x+x**2+sqrt(x) for x in arange(0,100)])
```

```
>>print d
```

```
333961.462947
```

Getting data into python

- Direct entry (command, script)
- File object (reading from external file)
- Online data object (e.g., Mysql)
- Python shelve method
- R or other objects (requires additional module)

Reading data from a file

```
#reading data from files- method 1-- load data into a list nobs*nvariables
#create a file object snork
>>snork= open('snorkel.csv')
#set up an empty list called
>>data = []
>>for line in snork:
>>    gradient,mndepth,mnwidth,crx,capt,known,ef = line.split(',')
>>    data.append([float(gradient),float(mndepth),float(mnwidth),float(crx),int(capt),
                    int(known),int(ef)])
>>print data[0] #prints the first record
[9.5, 0.14999999999999999, 4.0300000000000002, 0.60499999999999998, 4, 40, 0]
```

Reading data from a file

```
#method load data into dictionary
>>wsdata = open('watershed.csv')
>>WS = {}
>>>for line in wsdata:
>>  wshd_id,temp,precip,str_cons,felsic,carbonate,forest,range1,urban,drain_den,sqkm,geo_road,anadac\
>>                                     = line.split(',') #line continuation
>>  WS[wshd_id] = [int(wshd_id),float(temp),float(precip),float(str_cons),float(felsic),
>>                float(carbonate),float(forest), float(range1),float(urban),float(drain_den),
>>                float(sqkm),float(geo_road),float(anadac)]
>>print 'data for watershed 50'
data for watershed 50
>>print WS['50']
[50, 7.070000000000000003, 93.54000000000000006, 0.0,
62.31000000000000002, 12.23, 94.73999999999999995, 0.0, 0.0,
1.090000000000000001, 56.21892939, 1.49, 0.0]
```


Outputting data

- Print
- Write to file object
- Shelve
- Send to another program

Example

```
>>>outdata=open('output.csv','w')
>>>outdata.write(('s\n') %('gradient, depth,width'))
>>>snork= open('snorkel.csv')
>>>for line in snork:
>>>  gradient,mndepth,mnwidth,crx,capt,known,ef = line.split(',')
>>>  outdata.write(('s,%s,%s\n') %(gradient, mndepth,mnwidth))
outdata.close()
snork.close()
```

header



Reading from relational databases

- Efficient storage, manipulation, query, and retrieval
- Requires third-party modules
 - E.g., MySQLdb

Example

```
>>> import MySQLdb # import database package, and connect
>>> db = MySQLdb.connect
(db='okanagan',host='fisher.forestry.uga.edu',user='guest',passwd='2phast')
>>> cur = db.cursor() # create a cursor object to mediate
# communication with database
# run query
>>> cur.execute('SELECT nestid, species, site, treename FROM nests
WHERE year=96')
10L
>>> data = cur.fetchall() # fetch data, and assign to variable
>>> data
((334L, 'WAVI', 'INK', 'Alder'), (335L, 'WAVI', 'VEN', 'Willow'),
(336L, 'WAVI', 'VEN', 'Alder'), (337L, 'WAVI', 'VEN', 'Lodgepole Pine'),
(338L, 'WAVI', 'VEN', 'Lodgepole Pine'), (339L, 'WAVI', 'VEN', 'Alder'),
(340L, 'WAVI', 'Bart', 'Alder'), (341L, 'WAVI', 'Bart', 'Alder'), (342L,
'WAVI', 'Bart', 'Alder'), (343L, 'WAVI', 'Bart', 'Willow'))
```

Functions

- Numeric, string, dictionary, or any other data operation
- Treated as objects
- Callable from main program
 - () creates an instance

Example

```
>>x=[45, 50, 60, 65, 75]
>>def mean(data):
>>    return sum(data)/float(len(data))
    #NOTE! without 'float' treats as integer division
>>mean(x)
59.0
#short cut method using lambda
>>xbar=lambda data:sum(data)/float(len(data))
>>xbar(x)
59.0
```

Using functions together

```
>>>def fact(n):
```

```
>>  x=1
```

```
>>  for i in range(n,1,-1):
```

```
>>      x*=i
```

```
>>  return x
```

```
>>>def comb(n,x):
```

```
>>  y=fact(n)/fact(x)/fact(n-x)
```

```
>>  return y
```

```
>>>print '10 choose 5=',comb(10,5)
```

```
10 choose 5 = 252
```

Classes

- Set of attributes associated with a collection of objects
 - Class variables
 - Methods (functions)
- Objects in class refer the class object by `self.classname` convention
- Class just defines attributes
 - Use requires invoking an *instance*
- Central to object-oriented programming
 - All objects within class that refer to the class by `self` are available outside class
 - Other objects are local unless defined globally

Class example

```
>>>class combine:
>>     def fact(self,n):
>>         x=1
>>         for i in range(n,1,-1):
>>             x*=i
>>         return x
>>     def comb(self,n,x):
>>         y=fact(n)/fact(x)/fact(n-x)
>>         return y
>>>#class instance
>>>c=combine()
>>>#invoking the bound method
>>>print c.fact(10)
3628800
>>>print c.comb(10,2)
42
```

Inheritability

- New class that specialize or modifies the behavior of an existing class
 - Original class= base class or *superclass*
 - New class = derived class or *subclass*
- Subclasses inherit all the attributes of superclass
- Definitions within subclass instance override same superclass definition

Example

Superclass mammal

```
>>>class mammal:
>>>    #class representing mammals
>>>    name='mammal'
>>>    def lactate(self):
>>>        print 'mammals lactate'
>>>    def hair(self):
>>>        print 'mammals have hair'
>>>    def endothermy(self):
>>>        print 'mammals are endothermic'
>>>mammal().endothermy()
mammals are endothermic
```

Subclasses- marsupial and placental (sub-subclass)

```
>>class marsupial(mammal):
>>  #inherits all attributes of class mammal
>>  name='marsupial'
>>  def placenta(self):
>>      print 'no placenta'
>>class placental(marsupial):
>>  #inherits from marsupial and mammal
>>  #replaces name and placenta
>>  name='placental'
>>  def placenta(self):
>>      print 'has a placenta'
>>print placental.name
placental
>>placental().hair()
mammals have hair
>>placental().placenta()
has a placenta
```

Some special methods

- Enumerate
- Matrix algebra
- Simulating random variables

Enumerate

```
#enumerate over and print index and value
```

```
>>print 'x=',x
```

```
x= [45, 50, 60, 65, 75]
```

```
>>print 'enumerate x'
```

```
>>for i,j in enumerate(x):
```

```
    >> print i,j
```

```
0 45
```

```
1 50
```

```
2 60 ← entry
```

```
3 65
```

```
4 75
```

index

More useful example

```
>> hist='1001020'  
>> print 'hist', hist  
hist 1001020  
>> print 'enumerate hist'  
enumerate hist  
>> for i,j in enumerate(hist):  
>>     print i,j  
0 1  
1 0  
2 0  
3 1  
4 0  
5 2  
6 0
```

Use to turn history string into list of indice
and corresponding values

Matrix operations

- Numpy or scipy
- Define matrix objects
- Matrix operations
 - *, +, etc
 - Methods
 - Transpose
 - Inverse
- Special functions
 - E.g., linalg → determinant, eigenanalysis

Example

```
>>inv=linalg.inv
>>det=linalg.det
>>eig=linalg.eig
>>A=mat([[5,6,7],[6,7,9],[9,10,11]])
>>B=inv(A)
>>print 'A*B=',A*B
A*B= [[ 1.000000000e+00 -4.44089210e-16 -1.22124533e-15]
 [ 0.000000000e+00  1.000000000e+00 -7.77156117e-16]
 [ 1.77635684e-15 -4.44089210e-16  1.000000000e+00]]
>>print 'eigenvalues', eig(A)[0]
>>eigenvalues [ 23.92647089 -0.24550115 -0.68096974]
>>print 'eigenvectors',eig(A)[1]
eigenvectors [[-0.43712941 -0.50675339  0.01959232]
 [-0.53812871  0.80449292 -0.76691223]
 [-0.72064927 -0.309826  0.64145286]]
```

Simulating random variables

- `Numpy.random`
- E.g. `binomial`
 - `binomial(n,p)` simulates single binomial
 - `binomial(n,p,m)` simulates vector of m binomials with parameters n, p

Examples

```
>>n=100
```

```
>>p=0.5
```

```
>>print 'single binomial variate with n=',n, 'and p=',p,'x=',binomial(n,p)  
single binomial variate with n= 100 and p= 0.5 x= 46
```

```
>>m=10
```

```
>>print m, ' binomial variates', binomial(n,p,m)
```

```
>10 binomial variates [55 47 51 46 49 49 51 45 48 47]
```

```
>a=2
```

```
>b=2
```

```
>m=10
```

```
>x=[binomial(n,beta(a,b)) for i in range(m)]
```

```
>print '10 beta-binomial rvs', x
```

```
10 beta-binomial rvs [30, 24, 61, 74, 40, 42, 80, 56, 28, 50]
```

Debuggery

- Idle or other colourized editor
 - Colour-coded key words
 - Syntax checks
 - Run from editor
- Python debugger (pdb)
 - Insert breaks into program
 - Program executes to break
 - Can proceed line-by-line
- Use of exceptions
 - Trap and print errors
 - Avoid error termination

