PyROOT

doing data analysis with Python and ROOT

What is Python?

- a dynamic programming language
- object-oriented
- expressive language + powerful standard library
- suited for many tasks
- extendable in other languages, especially C or C++
- garbage collection

What is PyROOT?

- an extension to python
- written by Wim Lavrijsen
- allows the user to use ROOT classes from Python
- let's you get the best of both worlds

more about Python

- interpreted language
- write it like a shell script or use the interpreter like a shell
- weak typing
- blocks are implemented via indentation
- yields tidy code
- spaces are no tabs

```
#!/usr/bin/python
print("Hello World")
```

```
>>> x = 1
>>> print x
1
>>> x = "I'm a string"
>>> print x
I'm a string
```

```
>>> if 2 > 1:
...     print "all right"
...     print "2>1"
... else:
...     print "something is wrong"
...
all right
2>1
```

some important Python built-in types:

• numeric types:	int	C's long
	float	C's double
	long	integer with unlimited precision
	complex	real and imaginary are double
• sequence types:	str	strings
	list	[1,2,"string", 3+4j]
• file objects		returned by functions to access files, urls, ...

• casting if needed:

```
>>> x = 1.3
>>> str(x)
'1.3'
>>> list(str(x))
['1', '.', '3']
>>> x = "34.6E2"
>>> float(x)
3460.0
```

sequence types (lists, strings, arrays)

- python lists have no fixed size and can contain any object

- convenient:

```
>>> a = [3,5,1,4,2]
>>> a + ["a",7]
[3, 5, 1, 4, 2, 'a', 7]
>>> 5*[0]
[0, 0, 0, 0, 0]
>>> a[1:4]
[5, 1, 4]
>>> a[1:]
[5, 1, 4, 2]
>>> a.sort()
>>> a
[1, 2, 3, 4, 5]
>>> 3 in a
True
>>> len(a)
5
```

```
for i in a:
...     print i
...
1
2
3
4
5
>>>
```

convenient but not too efficient

- most of the time we deal with arrays of one type (doubles)
- ROOT-methods require arrays of one type

```
>>> from array import array
>>> ar = array("d", a)
>>> ar
array('d', [1.0, 2.0, 3.0, 4.0, 5.0])
>>> ar[0]
1.0
```

- a first use of PyROOT:

```
>>> from ROOT import TGraph
>>> from array import array
>>> x = array("d", [1,2,3,4,5])
>>> y = array("d", [3,2,6,3,7])
>>> g = TGraph(len(x), x,y)
>>> g.Draw("AL*")
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```

string manipulation:

- some often used methods:
  - split()
  - strip()

```
>>> s = "Hello World !"
>>> s.split()
['Hello', 'World', '!']
>>> s.split("l")
['He', '', 'o Wor', 'd !']
```

```
>>> s = "      Hello World!    \n"
>>> s.strip()
'Hello World!'
```

- string substitution:

```
>>> a = 1
>>> " a has value %7.5f" % a
' a has value 1.00000'
>>> " %2d + %2d = %2d " % (1,1,2)
'  1 +  1 =  2 '
```

Examples: reading files & create graphs

reading the output of a command:

```
>>> import os
>>> f = os.popen("date")
>>> for l in f:
...     print l
...
Mi 22. Nov 14:31:22 CET 2006
```

reading from an URL:

```
>>> import urllib
>>> f = urllib.urlopen("http://www.google.de")
>>> for l in f:
...    print l
...
<html><head><meta http-equiv="content-type"
content="text/html; charset=ISO-8859-
1"><title>Google</title><style><!--


...
```

plotting a python function:

```
>>> from ROOT import TF1
>>> def myfunc(x):
...     return x[0]**2
...
>>> f = TF1("myf", myfunc, -2., 2.)
>>> f.Draw()
```

using in ROOT defined structs:

```
#!/usr/bin/python
from ROOT import *
gROOT.ProcessLine("struct mystruct { Int_t a; Int_t b; };")
from ROOT import mystruct
s = mystruct()
s.a = 1
s.b = 2
print s.a, s.b
```

using in ROOT defined classes:

```
class SimpleClass {
    public:
    SimpleClass(Long_t a, Long_t b) {
      this->a = a; this->b = b;};
    Long_t a;
    Long_t b;

    Long_t add() { return a+b; };
};
```

```
echo .L simpleclass.C+ | root -b
```

```
from ROOT import gSystem
gSystem.Load("simpleclass_C")
from ROOT import SimpleClass
x = SimpleClass(1,2)
print x.add()
```

... and extend them!

```
#!/usr/bin/python

from ROOT import gSystem

gSystem.Load("simpleclass_C")

from ROOT import SimpleClass

x = SimpleClass(1,2)
print x.add()


def sub(self):
    return self.a - self.b

SimpleClass.sub = sub

print x.sub()
```

read a Tchain and fill a histogram:

```
from ROOT import *
from glob import glob

files = glob("/home/eike/desy/baikal/datafiles/ralf*.root")
chain = TChain("h77")

for f in files:
    chain.Add(f)

entries = chain.GetEntries()

h = TH1D("h", "phi dist", 360, 0, 360)

for i in xrange(entries): # xrange yields an iterator from 0..entries-1
    chain.GetEntry(i)
    h.Fill(chain.phi_nt[0]) # [0] subtility in root 5.12

h.Draw()
gApplication.Run()
```

write a TTree:

```
from ROOT import *
from array import array

f = TFile("tree.root", "recreate")
t = TTree("t", "sample tree")

n = array("f", [0])
u = array("f", [0])

t.Branch('normal', n, 'normal/F')
t.Branch('uniform', u, 'uniform/F')

for i in xrange(10000):
    n[0] = gRandom.Gaus()
    u[0] = gRandom.Uniform()
    t.Fill()

f.Write()
f.Close()
```

this was only the tip of the iceberg:

- http://docs.python.org/lib/
  - more string methods
  - file testing
  - dealing with dates and calendars
  - networking:
    - processing mails, access smtp-servers
    - reading the content of urls, access ftp servers
    - parsing xml (html-) files
    - accessing databases

- python has bindings for many other c-libraries (e.g. QT)

- numpy & pylab:
  - packages for numerical calculations with n-dimensional arrays
  - optimization-problems
  - *and much more...*

- pyROOT: http://wlav.web.cern.ch/wlav/pyroot/index.html