**Python 2.6 or 2.7

SNOLab Computer Day 2014

Brian Mong

Python is a programming language, *Pythonic* is a philosophy

- Python is a scripting/general purpose language with some general principals
  - Easy to read/write
  - Fun to program (spam, spam, spam, spam, eggs, spam spam …)
- Interactive shell, just type python, ipython, etc.
- Scripts are run as: `python script.py`
  - `python –i scripy.py` *gives you prompt after script runs*
  - `python script.py args` *pass arguments to your script*
- Try this:
  - Run python
  - `import this`

Basics - Types

- Integer
- Float
- Tuple
- List
- Dictionary

- *Custom Class*

```
>>> 3/4 #Integer division
0
>>> 3./4 #float division
0.75
>>> a = (1,2,3) #Tuple
>>> a[1]
2
>>> b = [1,2,3] #List
>>> b[1]
2
>>> c = {'a':1,'b':2,'c':3} #Dict
>>> c['b']
2
>>> type(3)
<type 'int'>
>>> type(3.)
<type 'float'>
>>> type((1,2,3))
<type 'tuple'>
>>> type([1,2])
<type 'list'>
>>> type({})
<type 'dict'>
```

Basics2 - Python Cheat Sheet

- `True/False/None`
- `range(5) = [0,1,2,3,4]`
- `enumerate(['a','b']) = (0,'a'),(1,'b')`
- `for i in [1,2,3,4]:`
  - `continue / break`
- `if 'a' in ['a','b','c'] or 'a' == "A":`
- `elif 1.0 != 1 and type(1) == int:`
- `else:`
- `"""Multiline'z "String'z\"""`
- C-like string replacement
  - `print "Hello %s %d, %0.2f"%('World',123,99.9901)`
    `Hello World 123, 99.99`

Basics3

Lookup methods available on an object with dir()

```
>>> c
{'a': 1, 'c': 3, 'b': 2}
>>> dir(c)
['__class__', '__cmp__', '__contains__', '__delattr__',
'__delitem__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__',
'__init__', '__iter__', '__le__', '__len__', '__lt__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__setitem__', '__sizeof__',
'__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys',
'get', 'has_key', 'items', 'iteritems', 'iterkeys',
'itervalues', 'keys', 'pop', 'popitem', 'setdefault',
'update', 'values', 'viewitems', 'viewkeys', 'viewvalues']
>>> c.keys()
['a', 'c', 'b']
>>> c.has_key(3)
False
```

Note: Methods with **__xxx__** are not meant to be used by you!

Question1

- Look at `dir(())` and `dir([])` and determine the difference between a tuple and a list.

```
>>> dir(())
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__getslice__', '__gt__',
'__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'count', 'index’]
>>> dir([])
['__add__', '__class__', '__contains__', '__delattr__',
'__delitem__', '__delslice__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__getslice__',
'__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__reversed__', '__rmul__', '__setattr__', '__setitem__',
'__setslice__', '__sizeof__', '__str__', '__subclasshook__',
'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```

Python Structure

Python uses indention to distinguish blocks of a script

The standard is 4 spaces per level, no tabs!

```
def square_eggs(x):
    return x**2

square_spam = lambda x: x**2

y = [1,2,3,4,5]
spam  = []
eggs = []
for i in y:
    spam.append(square_spam(i))
    eggs.append(square_eggs(i))
    print i,i**2

print "spam == eggs?", spam==eggs
```

```
$ python indentions.py
1 1
2 4
3 9
4 16
5 25
spam == eggs? True
$
```

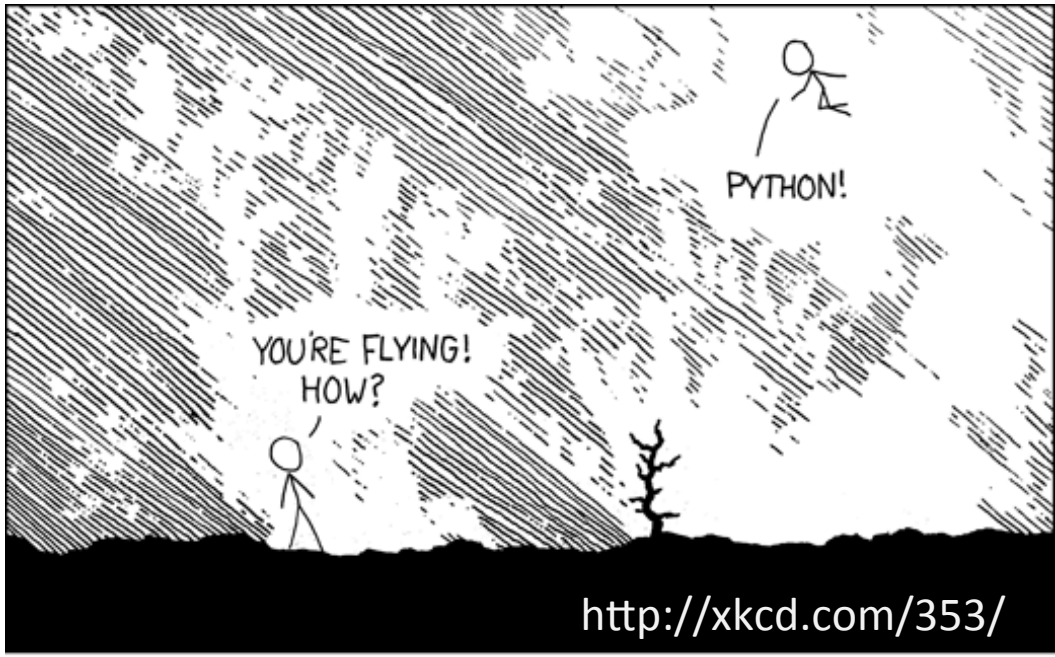
Import

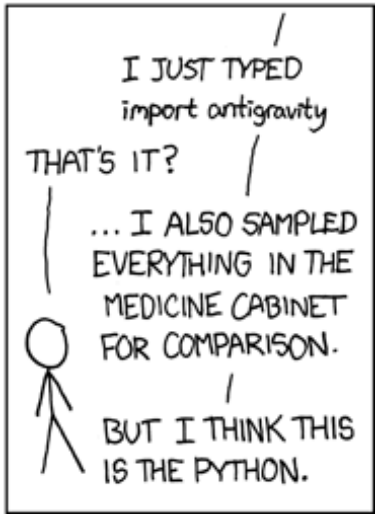
Usually someone has already written functions that do what you want.

The hardest part is deciding how you want to import…

namespace

```
import module  #pythonic
module.fun()

from module import fun
fun()

import module.fun as x
x()
```





Classes

Classes are how one defines new object types that contain

- data structure
- methods that act on that structure

```
class MyClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'hello world'
x = MyClass()
```

In this example:

`i` is an *attribute*

`f` is a *method*

`x` is an *instance*

Pythonists would recommend making sure a list or dictionary isn't suitable first

PyROOT

- PyROOT wraps ROOT to give you a better interactive shell
- PyROOT calls ROOT functions compiled in C, so there isn't any performance loss (for normal use)
- Access PyROOT by: `import ROOT`
  - `Generally replace (->,::) with .`
  - ROOT commands available in ROOT namespace
    - `c1 = ROOT.Tcanvas(…)`
    - `h1 = ROOT.TH1D("hist","hist",100,0,1)`
    - `h1.Draw()`
    - `tree = ROOT.TChain(…)`

Verses CINT ROOT syntax:

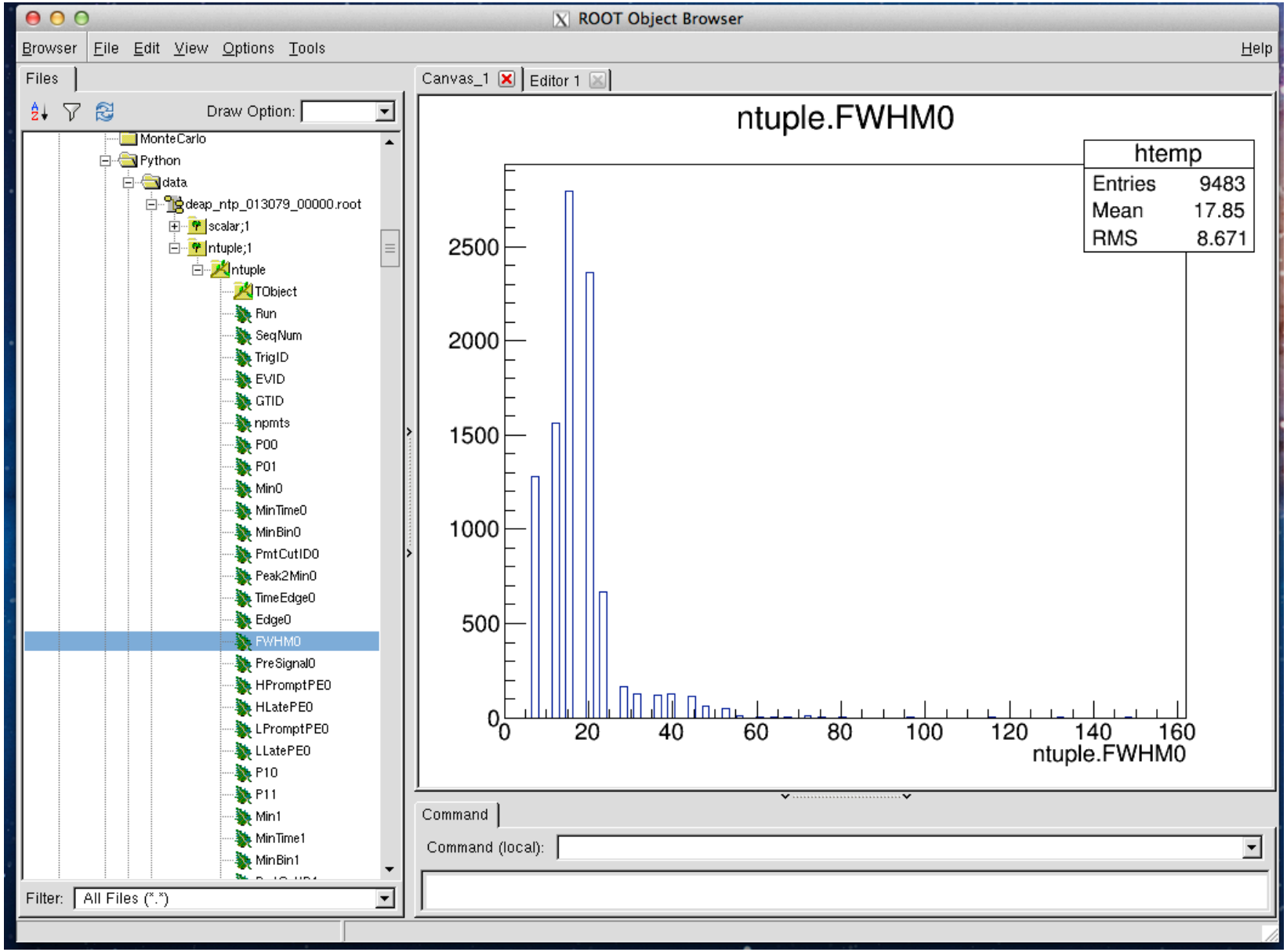
```
TH1D *h1 = new TH1D("hist","hist",100,0,1);
```

PyROOT ReExample

```
import ROOT
ROOT.gROOT.SetStyle("Plain")
ROOT.gStyle.SetPalette(1)

mychain = ROOT.TChain("ntuple")
myhist = ROOT.TH1D("hist","hist Name",100,0,1)
myfit = ROOT.TF1("fitf","gaus")
c1 = ROOT.TCanvas()

dataCuts = "Zfit > 0 && Zfit < 7 && MinBin0 > 650 && MinBin0 < 700 "
dataCuts+= "&& MinBin1 > 650 && MinBin1 < 700 && abs(MinBin0 - MinBin1) < 10 "
dataCuts+= "&& EventCutID == 1 && Fprompt > 0 && HLatePE0 > 0 && HLatePE1 > 0"

mychain.Add("data/deap_ntp_013079_*.root")
print "Loaded %d entries"%mychain.GetEntries()
moar_cut = " && TotalPE > 120 && TotalPE < 240"
mychain.Draw("Fprompt>>hist", dataCuts + " && TotalPE > 120 && TotalPE < 240")

myhist.Fit("fitf","LL","",0.68,1)
c1.SetLogy(1)
```

Also see: pyroot_generic.py

>>>ROOT.TBrowser()



ROOT tree.Scan

- tree.Scan(“var1:var2”,cuts)

```
>>> t.Scan("TrigID:MinBin0:Zfit",cuts)
************************************************
*     Row   *     TrigID *   MinBin0 *       Zfit *
************************************************
*         3 *          0 *       664 * 5.2510962 *
*         7 *          0 *       666 * 4.8705420 *
*        27 *          0 *       665 * 4.8093390 *
*        42 *          0 *       667 * 6.2409100 *
*        93 *          0 *       667 * 5.7747392 *
*       141 *          0 *       667 * 4.7239689 *
*       143 *          0 *       666 * 4.4660582 *
*       156 *          0 *       667 * 2.6396508 *
*       182 *          0 *       664 * 5.4336285 *
```

ROOT Tricks

- tree.Draw is your best friend
  - Fastest way to do anything, including get data

```
[bmong@science1 Python]$ python
Python 2.6.6 (r266:84292, Nov 21 2013, 12:39:37)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import ROOT
>>> t = ROOT.TChain("ntuple")
>>> t.Add("data/*")
14
>>> cuts = "Zfit > 0 && Zfit < 7 && MinBin0 > 650 && MinBin0 < 700 "
>>> cuts+= "&& MinBin1 > 650 && MinBin1 < 700 && abs(MinBin0 - MinBin1) < 10 "
>>> cuts+= "&& EventCutID == 1 && Fprompt > 0 && HLatePE0 > 0 && HLatePE1 > 0"
>>> cuts+= “&& TotalPE > 120 && TotalPE < 240"
>>> t.GetEntries()
1396683L
>>> t.Draw(">>+elist",cuts,"goff")
56733L
>>>
```

ROOT Tricks – Part doux

- tree.Draw is your best friend
  - Fastest way to do operations on your trees

```
>>> elist = ROOT.gDirectory.Get("elist")
>>> elist.GetN()
56733
>>> for i in range(elist.GetN()):
...  t.GetEntry(elist.GetEntry(i))
...
```

Use event lists to reduce data before looping over entries

- TTree:Draw@2
  - You can also get size of ntuple, Max$() Sum$(), and more with Draw

But Wait… There's More

`matplotlib` - matlab like plotting syntax

`numpy` - matrices, vectors & linalg oh my

`scipy` - bunch of random scientific functions

`iminuit` - minuit wrapper

Tons of others….

Parting wisdom

Save useful python functions to a file and import them when needed:

```
import sys
sys.path.append("/path/to/myfunctions.py")
import myfunctions
```

example: bungfun.py

Exercises

- Try converting a ROOT script to PyROOT
- Read random walk example provided
- Convert a ROOT.TH1 to only python objects
  - hint1: Read TH1.GetBin*(), TH1.GetNBinsX()
  - hint2:
```
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.show()
```
  - Note: science1 and most other remote servers require you to run matplotlib with `matplotlib.use("Agg")`