ROOT tutorial, part 2
TSelectors and pyROOT

Attilio Andreazza Università di Milano
Caterina Doglioni Université de Genève

HASCO school – 18/07/2012

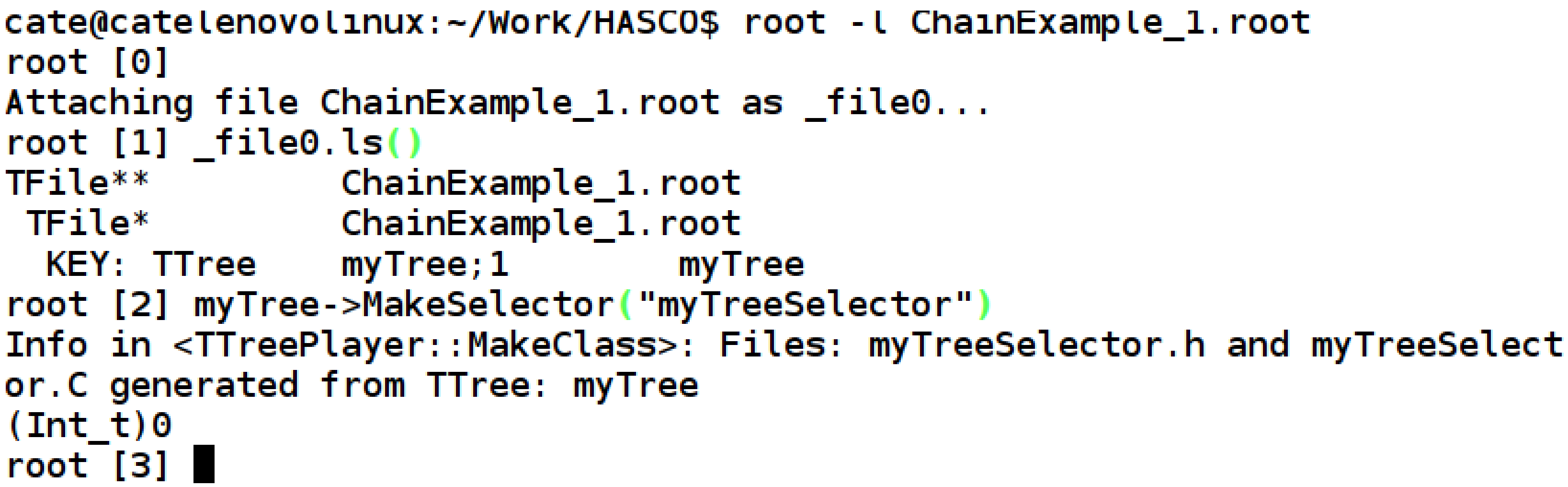
Data storage and more: Ttrees (2)

ROOT Tutorial
HASCO school – 17/07/2012

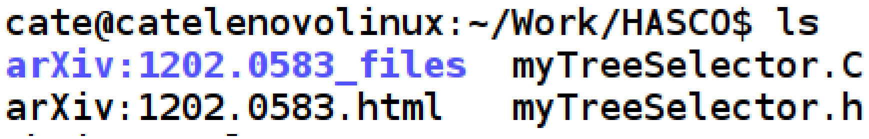
Reading a TTree: TSelector

Large trees → impractical to Scan()/Draw() by hand
Use **class** built from TTree (in interactive/compiled C++):
takes care of **reading out** branches/**looping** on entries

To make a TSelector out of a TTree



```
cate@catelenovolinux:~/Work/HASCO$ root -l ChainExample_1.root
root [0]
Attaching file ChainExample_1.root as _file0...
root [1] _file0.ls()
TFile**         ChainExample_1.root
 TFile*         ChainExample_1.root
  KEY: TTree    myTree;1        myTree
root [2] myTree->MakeSelector("myTreeSelector")
Info in <TTreePlayer::MakeClass>: Files: myTreeSelector.h and myTreeSelect
or.C generated from TTree: myTree
(Int_t)0
root [3] 
```

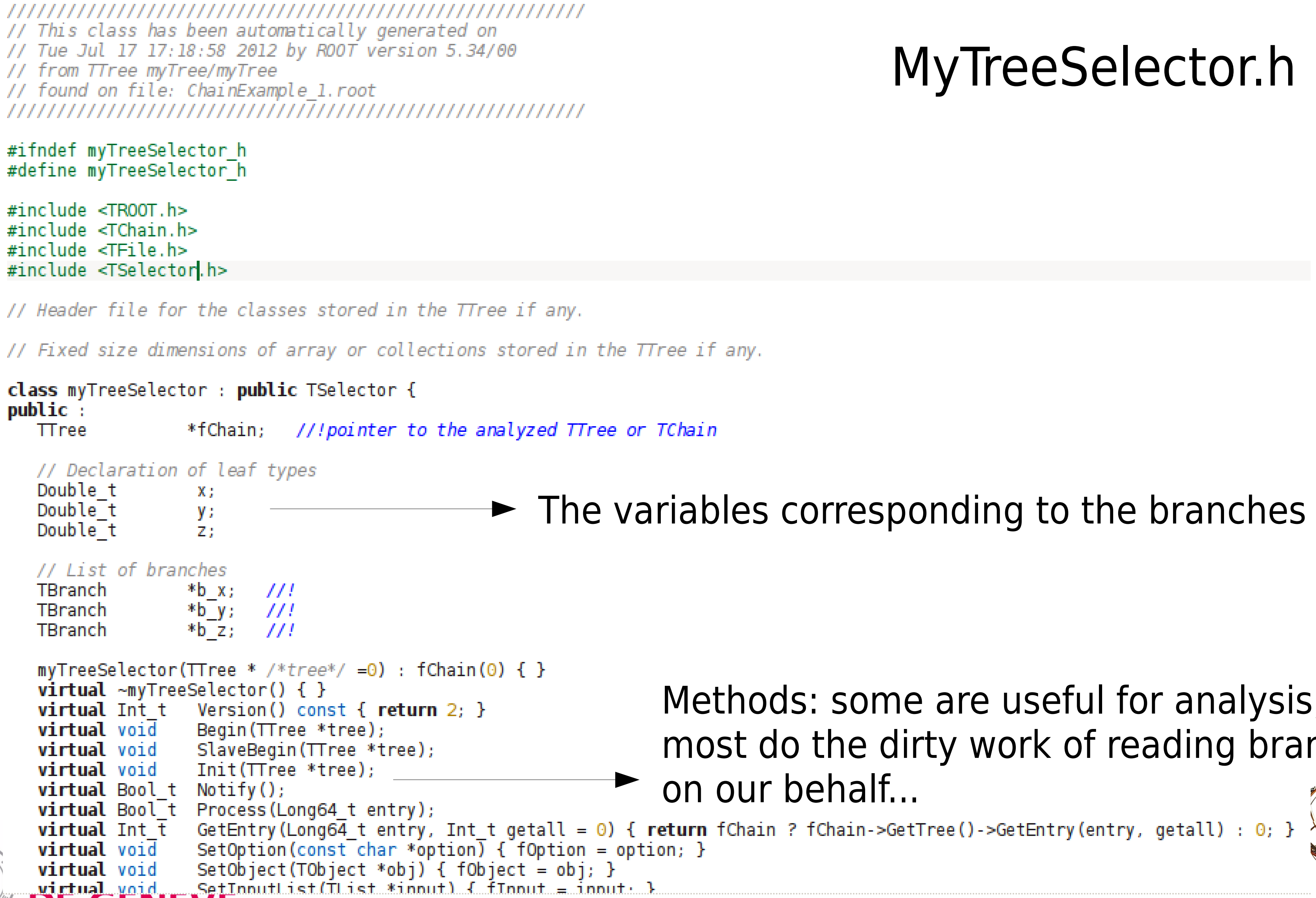


```
cate@catelenovolinux:~/Work/HASCO$ ls
arXiv:1202.0583_files  myTreeSelector.C
arXiv:1202.0583.html   myTreeSelector.h
```

→Two new files...let's open them!

TSelector interface

MyTreeSelector.h



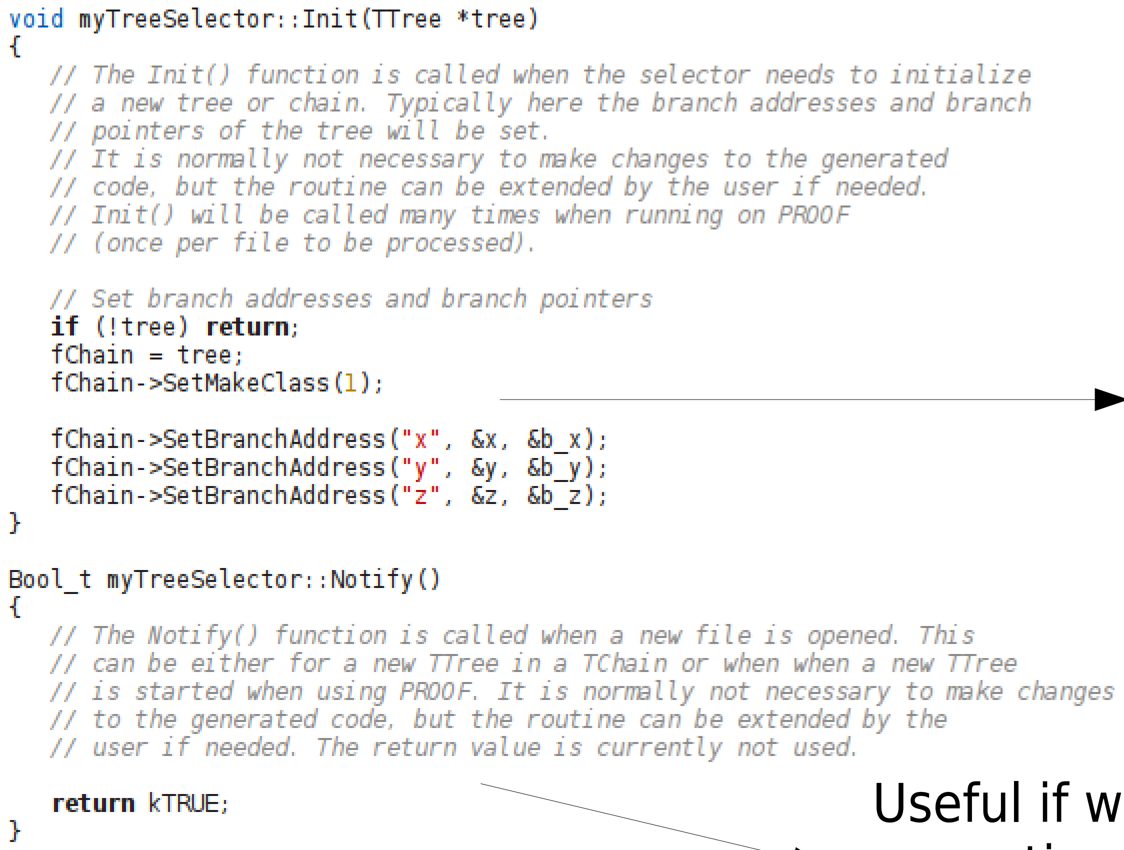
```
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Tue Jul 17 17:18:58 2012 by ROOT version 5.34/00
// from TTree myTree/myTree
// found on file: ChainExample_1.root
//////////////////////////////////////////////////////////

#ifndef myTreeSelector_h
#define myTreeSelector_h

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TSelector.h>

// Header file for the classes stored in the TTree if any.

// Fixed size dimensions of array or collections stored in the TTree if any.

class myTreeSelector : public TSelector {
public :
   TTree          *fChain;   //!pointer to the analyzed TTree or TChain

   // Declaration of leaf types
   Double_t        x;
   Double_t        y;
   Double_t        z;

   // List of branches
   TBranch        *b_x;   //!
   TBranch        *b_y;   //!
   TBranch        *b_z;   //!

   myTreeSelector(TTree * /*tree*/ =0) : fChain(0) { }
   virtual ~myTreeSelector() { }
   virtual Int_t   Version() const { return 2; }
   virtual void    Begin(TTree *tree);
   virtual void    SlaveBegin(TTree *tree);
   virtual void    Init(TTree *tree);
   virtual Bool_t  Notify();
   virtual Bool_t  Process(Long64_t entry);
   virtual Int_t   GetEntry(Long64_t entry, Int_t getall = 0) { return fChain ? fChain->GetTree()->GetEntry(entry, getall) : 0; }
   virtual void    SetOption(const char *option) { fOption = option; }
   virtual void    SetObject(TObject *obj) { fObject = obj; }
   virtual void    SetInputList(TList *input) { fInput = input; }
```

The variables corresponding to the branches

Methods: some are useful for analysis, most do the dirty work of reading branches on our behalf...

TSelector interface & implementation

MyTreeSelector.h



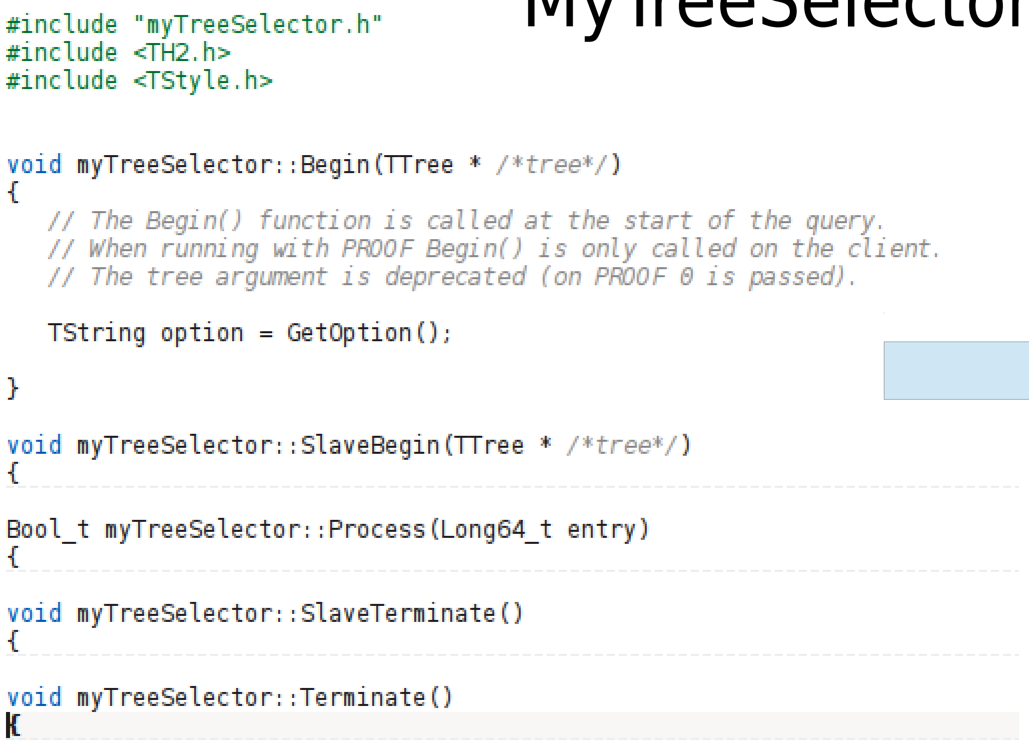
```
void myTreeSelector::Init(TTree *tree)
{
   // The Init() function is called when the selector needs to initialize
   // a new tree or chain. Typically here the branch addresses and branch
   // pointers of the tree will be set.
   // It is normally not necessary to make changes to the generated
   // code, but the routine can be extended by the user if needed.
   // Init() will be called many times when running on PROOF
   // (once per file to be processed).

   // Set branch addresses and branch pointers
   if (!tree) return;
   fChain = tree;
   fChain->SetMakeClass(1);

   fChain->SetBranchAddress("x", &x, &b_x);
   fChain->SetBranchAddress("y", &y, &b_y);
   fChain->SetBranchAddress("z", &z, &b_z);
}

Bool_t myTreeSelector::Notify()
{
   // The Notify() function is called when a new file is opened. This
   // can be either for a new TTree in a TChain or when when a new TTree
   // is started when using PROOF. It is normally not necessary to make changes
   // to the generated code, but the routine can be extended by the
   // user if needed. The return value is currently not used.

   return kTRUE;
}
```

Where the dirty work gets done!
This function gets called behind
the scenes by the base class

Useful if we need to do something
every time we open a new file
(e.g. print file name, attach metadata trees)

TSelector implementation

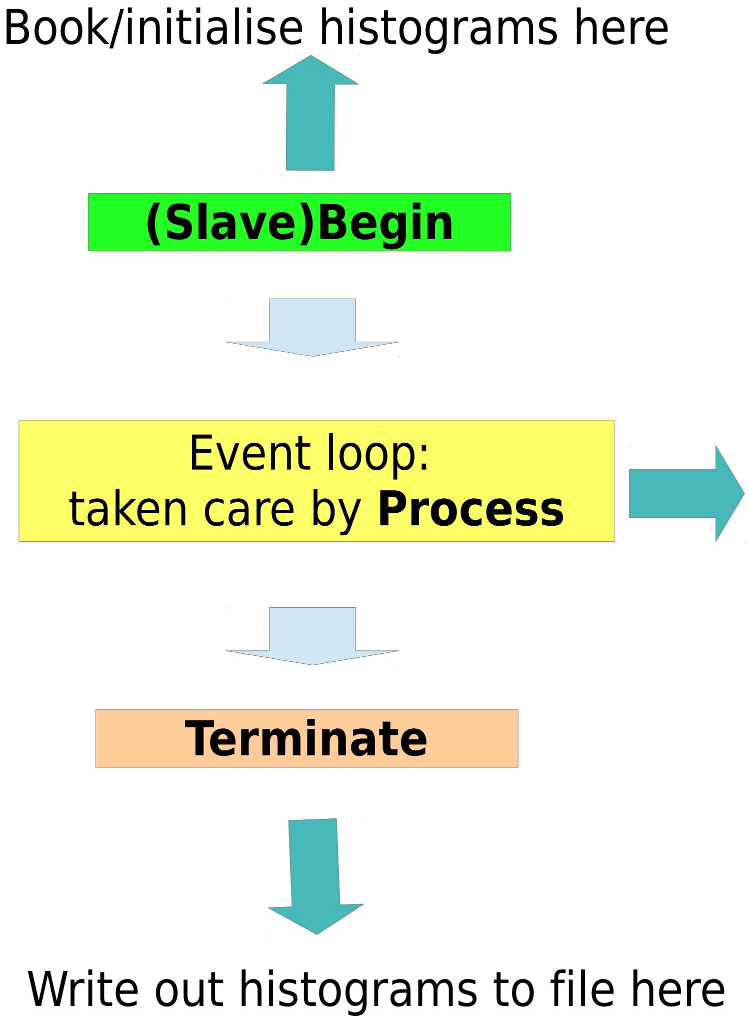
MyTreeSelector.C



```
#include "myTreeSelector.h"
#include <TH2.h>
#include <TStyle.h>


void myTreeSelector::Begin(TTree * /*tree*/)
{
   // The Begin() function is called at the start of the query.
   // When running with PROOF Begin() is only called on the client.
   // The tree argument is deprecated (on PROOF 0 is passed).

   TString option = GetOption();

}

void myTreeSelector::SlaveBegin(TTree * /*tree*/)
{

Bool_t myTreeSelector::Process(Long64_t entry)
{

void myTreeSelector::SlaveTerminate()
{

void myTreeSelector::Terminate()
{
```

All methods are empty!
Up to the user to do what he/she wants in them...

How does a TSelector work?



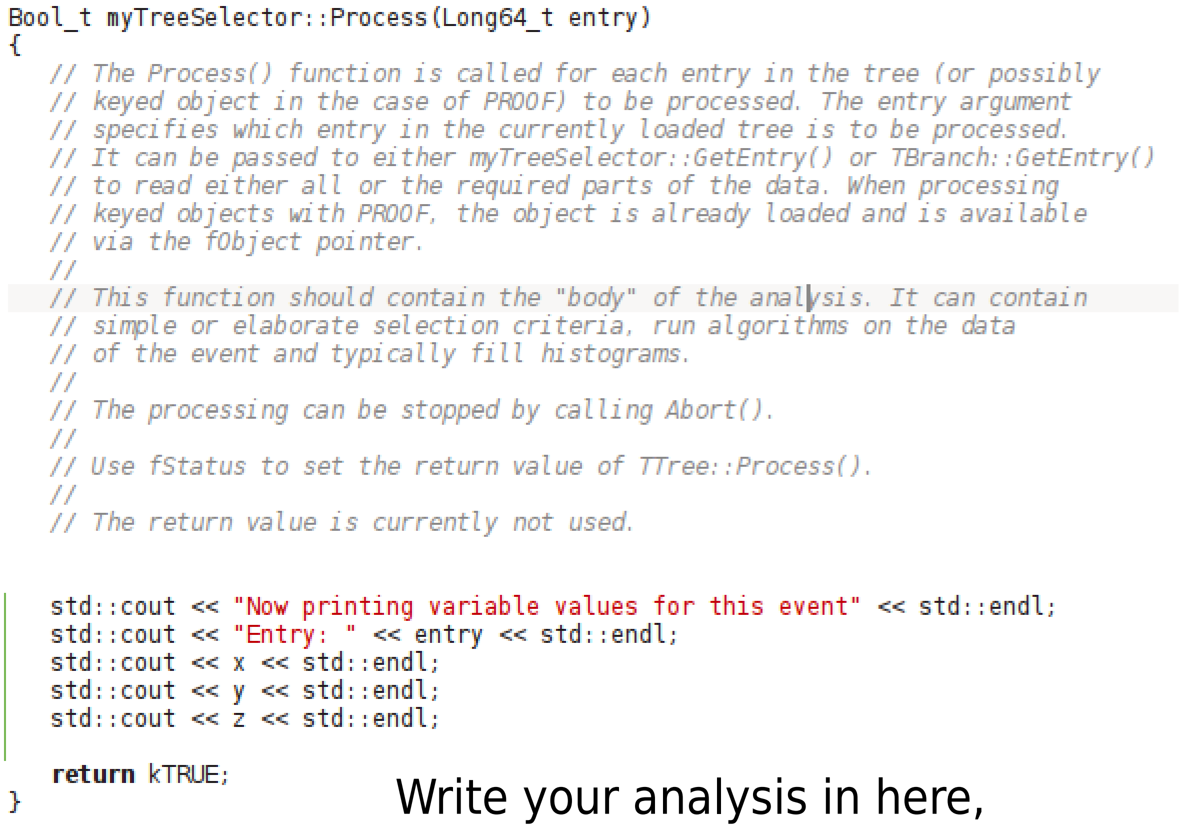
Book/initialise histograms here

(Slave)Begin

Event loop:
taken care by **Process**

Terminate

Write out histograms to file here

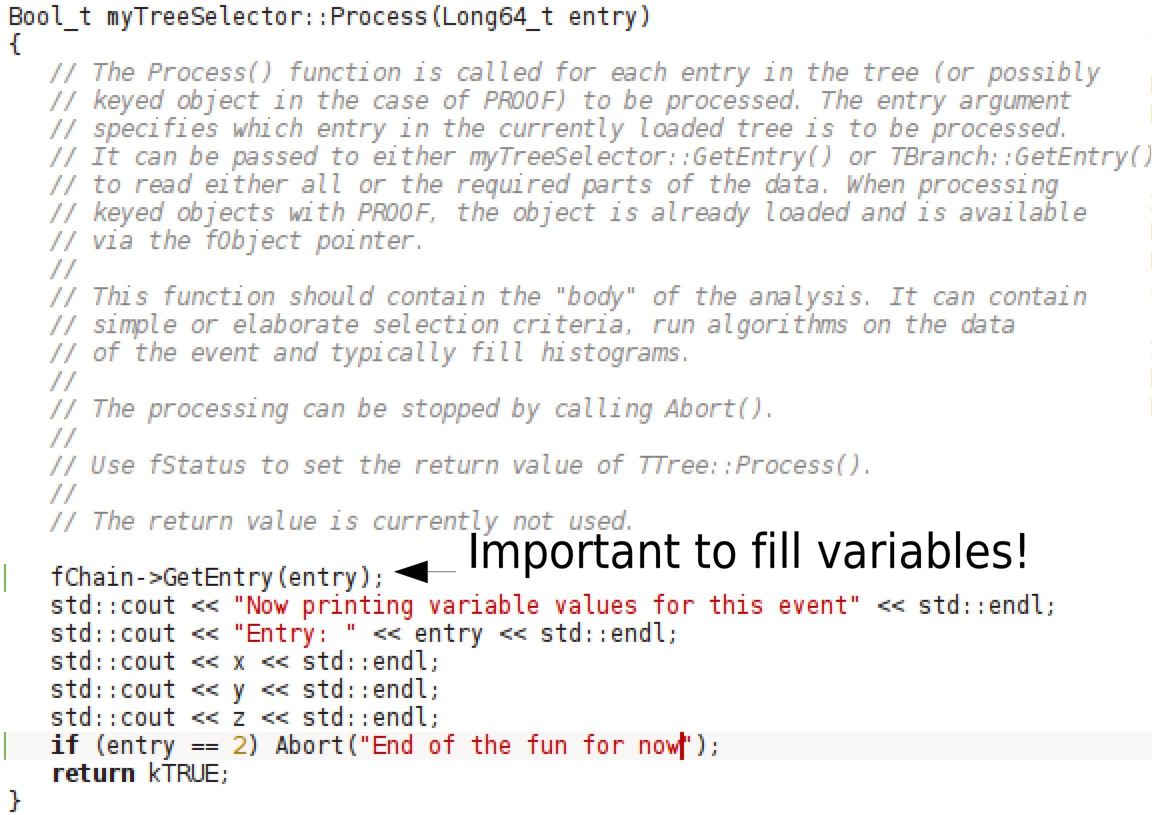


```
Bool_t myTreeSelector::Process(Long64_t entry)
{
   // The Process() function is called for each entry in the tree (or possibly
   // keyed object in the case of PROOF) to be processed. The entry argument
   // specifies which entry in the currently loaded tree is to be processed.
   // It can be passed to either myTreeSelector::GetEntry() or TBranch::GetEntry()
   // to read either all or the required parts of the data. When processing
   // keyed objects with PROOF, the object is already loaded and is available
   // via the fObject pointer.
   //
   // This function should contain the "body" of the analysis. It can contain
   // simple or elaborate selection criteria, run algorithms on the data
   // of the event and typically fill histograms.
   //
   // The processing can be stopped by calling Abort().
   //
   // Use fStatus to set the return value of TTree::Process().
   //
   // The return value is currently not used.


   std::cout << "Now printing variable values for this event" << std::endl;
   std::cout << "Entry: " << entry << std::endl;
   std::cout << x << std::endl;
   std::cout << y << std::endl;
   std::cout << z << std::endl;

   return kTRUE;
}
```

Write your analysis in here, taking advantage of easy access of variables: they will be filled for you in the same way we did when filling the TTree

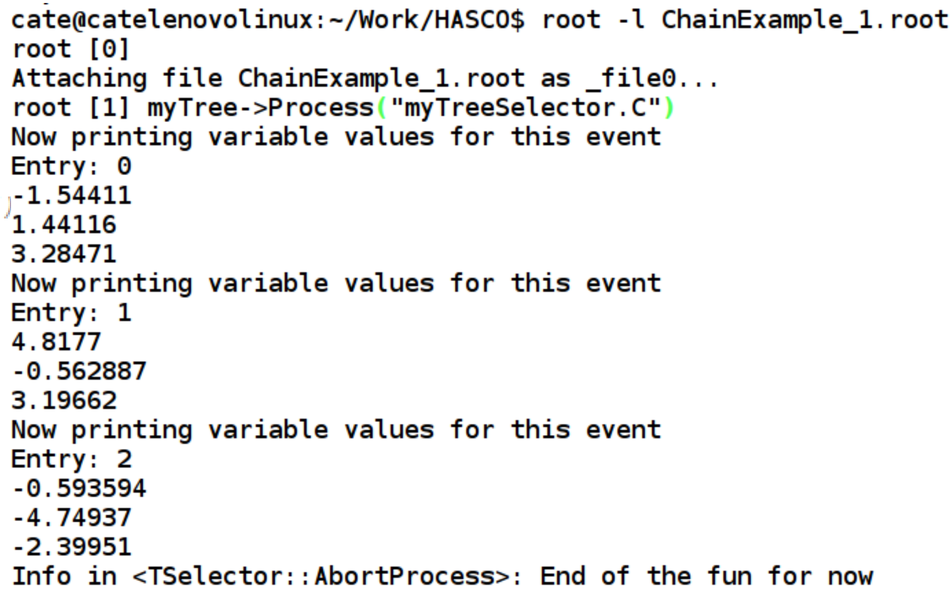
How do we use a TSelector in ROOT?

Interactive: very simple (as written in .C file)
Can also use makefile...recommended (faster)

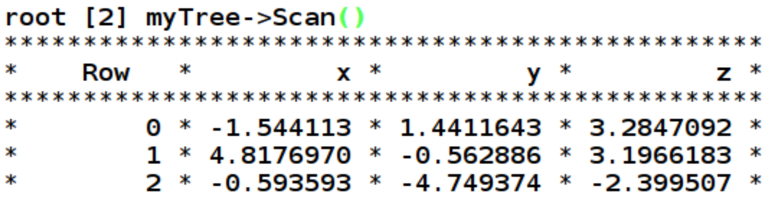


```
Bool_t myTreeSelector::Process(Long64_t entry)
{
   // The Process() function is called for each entry in the tree (or possibly
   // keyed object in the case of PROOF) to be processed. The entry argument
   // specifies which entry in the currently loaded tree is to be processed.
   // It can be passed to either myTreeSelector::GetEntry() or TBranch::GetEntry()
   // to read either all or the required parts of the data. When processing
   // keyed objects with PROOF, the object is already loaded and is available
   // via the fObject pointer.
   //
   // This function should contain the "body" of the analysis. It can contain
   // simple or elaborate selection criteria, run algorithms on the data
   // of the event and typically fill histograms.
   //
   // The processing can be stopped by calling Abort().
   //
   // Use fStatus to set the return value of TTree::Process().
   //
   // The return value is currently not used.

   fChain->GetEntry(entry);
   std::cout << "Now printing variable values for this event" << std::endl;
   std::cout << "Entry: " << entry << std::endl;
   std::cout << x << std::endl;
   std::cout << y << std::endl;
   std::cout << z << std::endl;
   if (entry == 2) Abort("End of the fun for now");
   return kTRUE;
}
```

Important to fill variables!



```
cate@catelenovolinux:~/Work/HASCO$ root -l ChainExample_1.root
root [0]
Attaching file ChainExample_1.root as _file0...
root [1] myTree->Process("myTreeSelector.C")
Now printing variable values for this event
Entry: 0
-1.54411
1.44116
3.28471
Now printing variable values for this event
Entry: 1
4.8177
-0.562887
3.19662
Now printing variable values for this event
Entry: 2
-0.593594
-4.74937
-2.39951
Info in <TSelector::AbortProcess>: End of the fun for now
```



```
root [2] myTree->Scan()
************************************************
*    Row   *         x *         y *         z *
************************************************
*        0 * -1.544113 * 1.4411643 * 3.2847092 *
*        1 * 4.8176970 * -0.562886 * 3.1966183 *
*        2 * -0.593593 * -4.749374 * -2.399507 *
```

pyROOT

Inspiration taken from a tutorial by Daniel Short (Oxford)

Usual disclaimer:
following slides are biased
by current use of pyROOT,
here only introducing
basics needed for the hands-on

A more complete set of lectures (Glasgow)
The pyROOT tutorials in ROOT

ROOT Tutorial
HASCO school – 18/07/2012

Why PyROOT?

...to avoid worrying about types and strings!

```
TTree * t = (TTree*) myFile->Get(“myTree”)
```

vs

```
t=myFile->Get(“myTree”)
```

```
TString s = TString::Form(“My string is %c of chars
and numbers, like %d”), “made”, 200)
cout << s.Data() << endl;
```

vs

```
s = “My string is”+” made ”+of chars and numbers,
like”+str(200)
print s
```

Python is a powerful language...

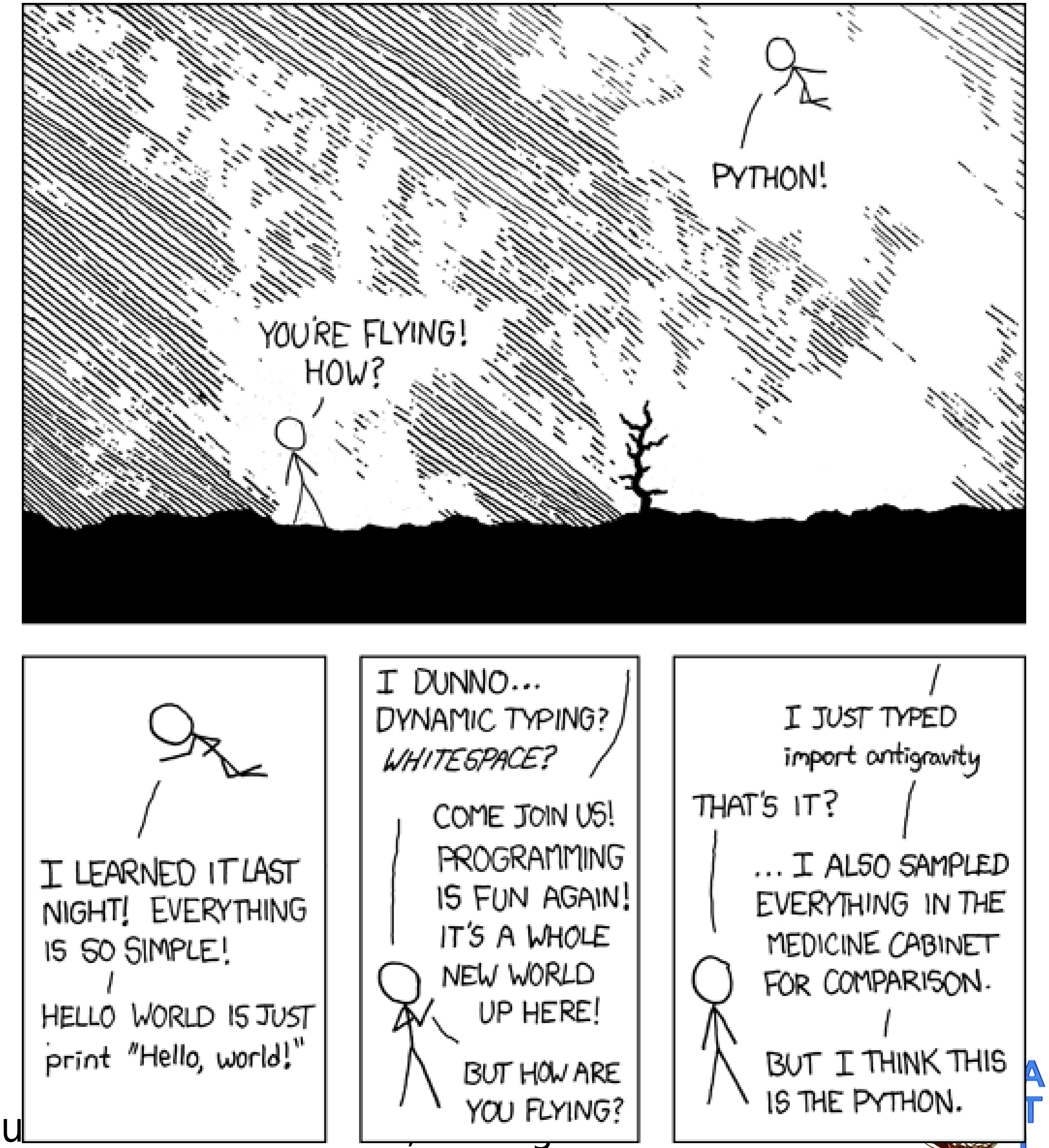
Formatting histograms in python does not do the language justice...

Python:

- High level interpreted programming language
- Object-oriented too!
- Some people write entire analyses using pyROOT and derivatives...can be done!
- We will be using it for **formatting plots**
  → advantage: ROOT macros treating data don't get polluted with string, axes renaming, colors treatment etc

Useful properties:

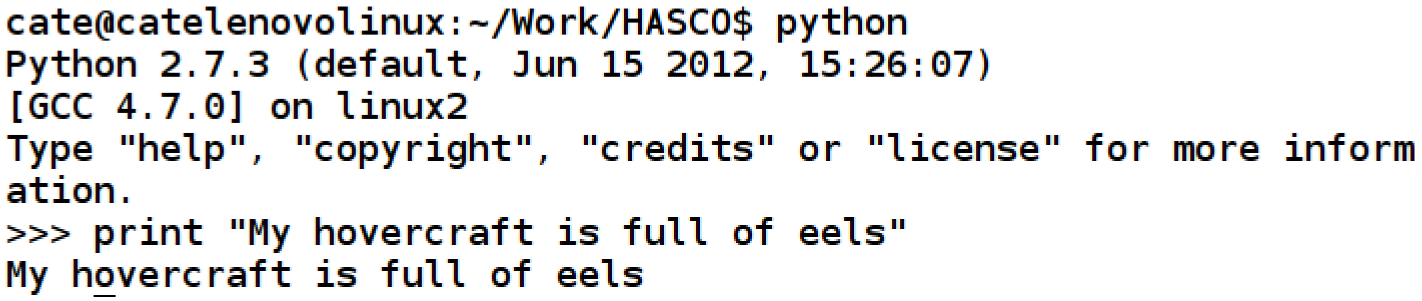
- Everything is a reference (no pointers...)
- Automatic garbage collection (this sometimes clashes with ROOT's...)
- Built-in help and reference listing
- Strongly typed
-


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

http://xkcd.com/353/

Using Python

Interactive console

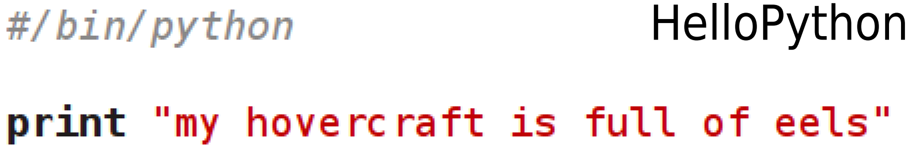


```
cate@catelenovolinux:~/Work/HASCO$ python
Python 2.7.3 (default, Jun 15 2012, 15:26:07)
[GCC 4.7.0] on linux2
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> print "My hovercraft is full of eels"
My hovercraft is full of eels
```

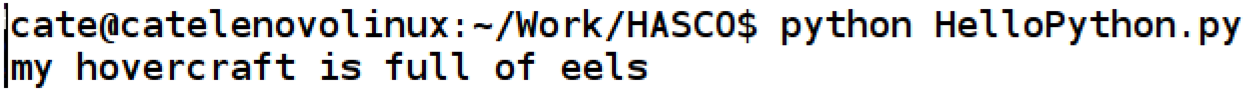
To quit session: **CTRL-D**

Precompiled scripts

HelloPython.py



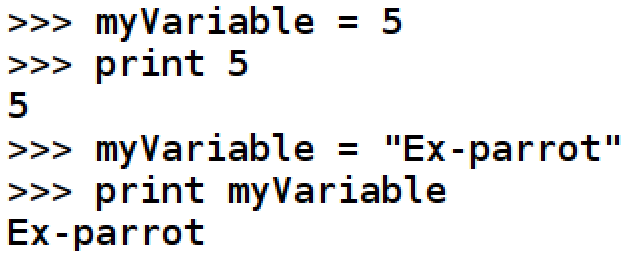
```
#/bin/python

print "my hovercraft is full of eels"
```



```
cate@catelenovolinux:~/Work/HASCO$ python HelloPython.py
my hovercraft is full of eels
```

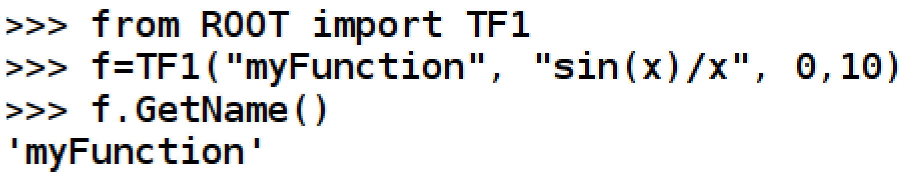
Essential Python (1)

Python **works out variable types** while running
→ no need for declaration!



```
>>> myVariable = 5
>>> print 5
5
>>> myVariable = "Ex-parrot"
>>> print myVariable
Ex-parrot
```

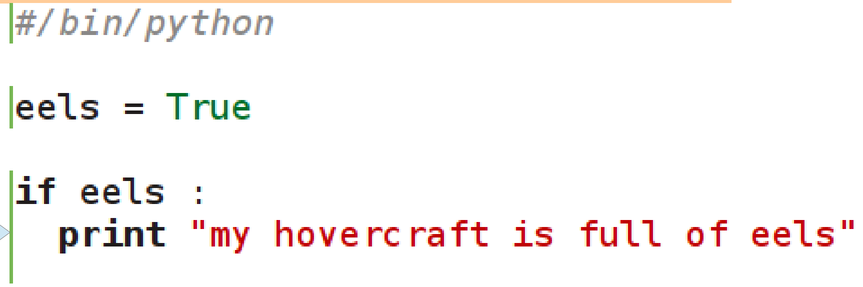
Python can use external libraries and functions (=modules)



```
>>> from ROOT import TF1
>>> f=TF1("myFunction", "sin(x)/x", 0,10)
>>> f.GetName()
'myFunction'
```

→ First hint of pyROOT

Python **cares about whitespace**



```
#/bin/python

eels = True

if eels :
  print "my hovercraft is full of eels"
```

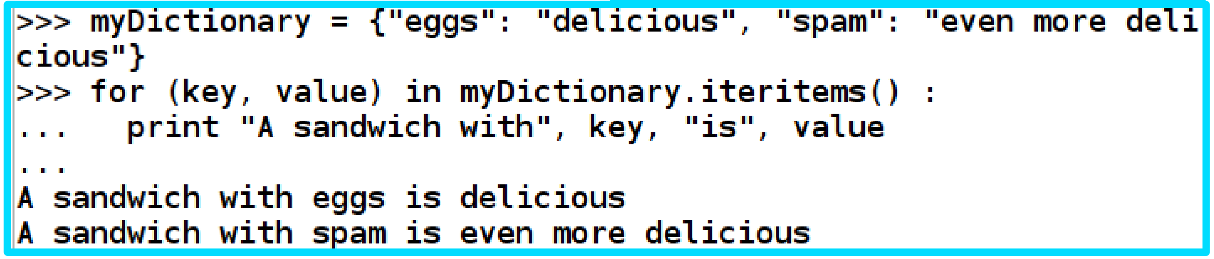
Need to **indent** in case of
if statements/for loops…

Essential Python by example (2)

For loops and xrange

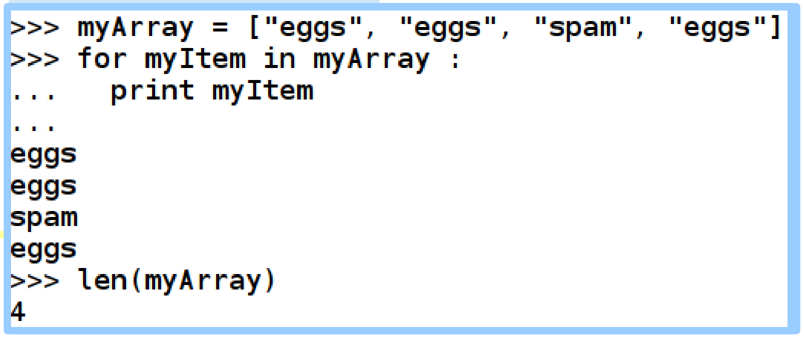
```
>>> for i in xrange(1,10) :
...    print i
...
1
2
3
4
5
6
7
8
9
```

Dictionaries



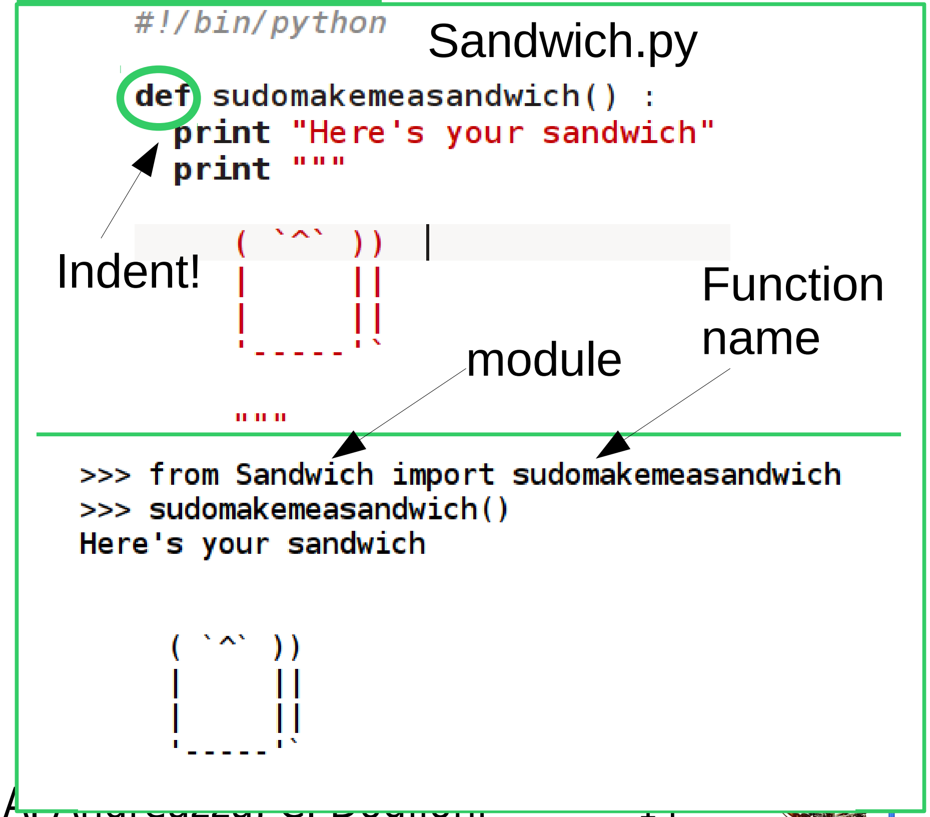
```
>>> myDictionary = {"eggs": "delicious", "spam": "even more deli
cious"}
>>> for (key, value) in myDictionary.iteritems() :
...    print "A sandwich with", key, "is", value
...
A sandwich with eggs is delicious
A sandwich with spam is even more delicious
```

Lists



```
>>> myArray = ["eggs", "eggs", "spam", "eggs"]
>>> for myItem in myArray :
...    print myItem
...
eggs
eggs
spam
eggs
>>> len(myArray)
4
```

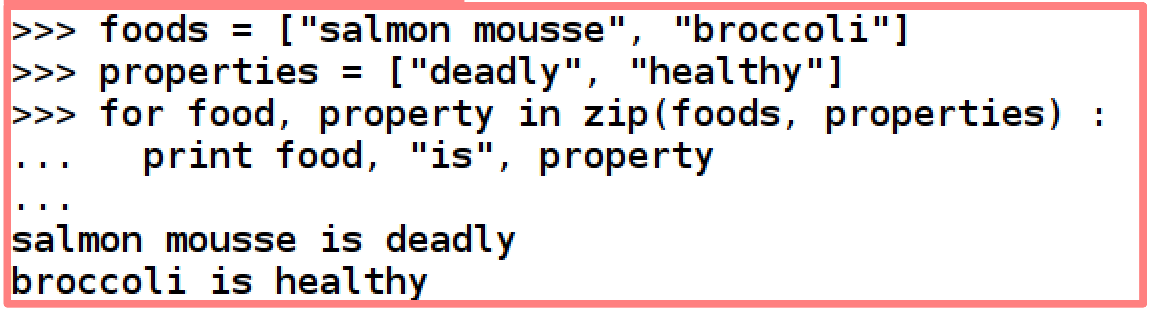
Functions



```
#!/bin/python

def sudomakemeasandwich() :
    print "Here's your sandwich"
    print """

( `^` ))
|      ||
|      ||
'-----'`

"""
```

```
>>> from Sandwich import sudomakemeasandwich
>>> sudomakemeasandwich()
Here's your sandwich


  ( `^` ))
  |      ||
  |      ||
  '-----'`
```

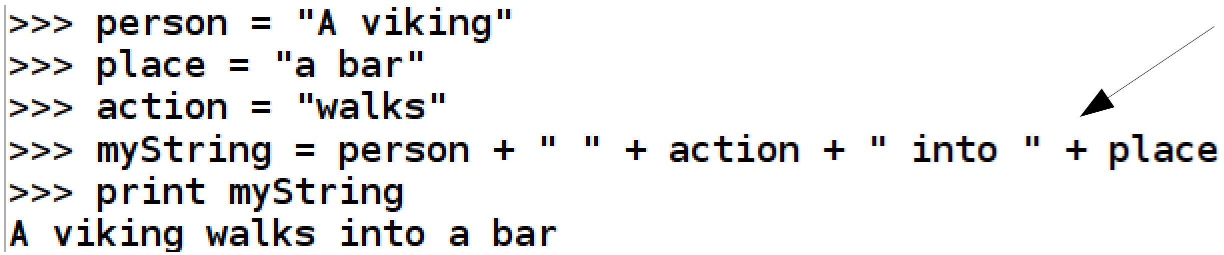
The **zip** function



```
>>> foods = ["salmon mousse", "broccoli"]
>>> properties = ["deadly", "healthy"]
>>> for food, property in zip(foods, properties) :
...    print food, "is", property
...
salmon mousse is deadly
broccoli is healthy
```

Strings in Python (1)

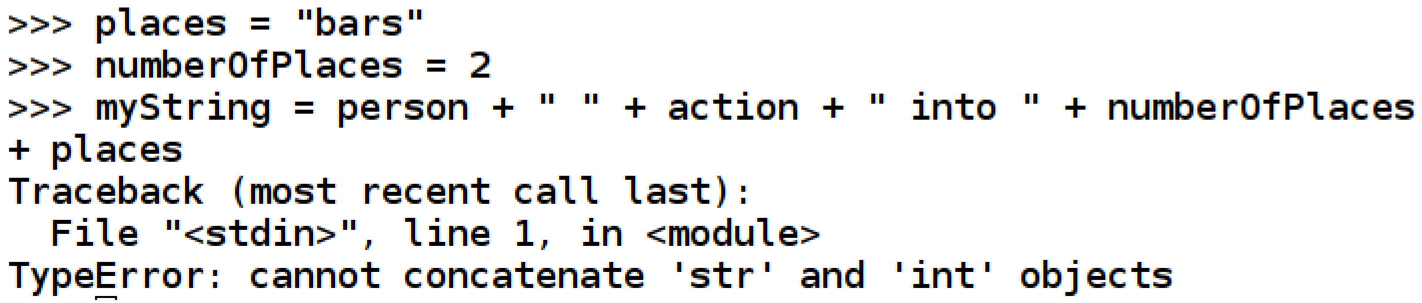
Building a string



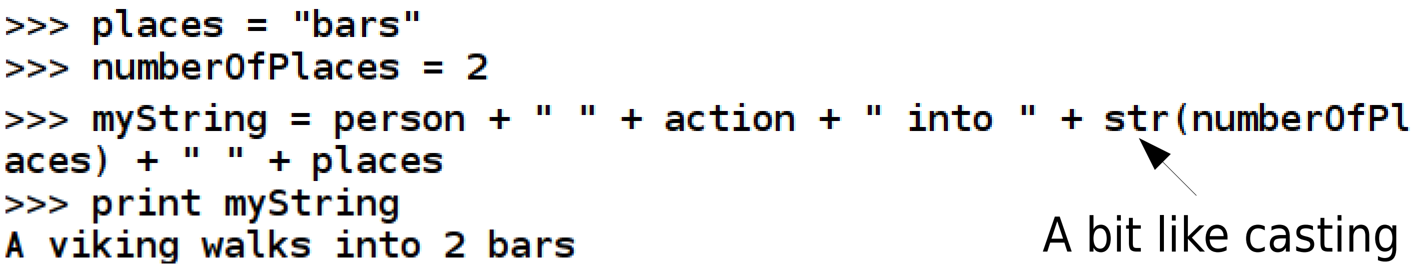
```
>>> person = "A viking"
>>> place = "a bar"
>>> action = "walks"
>>> myString = person + " " + action + " into " + place
>>> print myString
A viking walks into a bar
```

Very easy!

Numbers are not strings (or: python knows what type a variable is)



```
>>> places = "bars"
>>> numberOfPlaces = 2
>>> myString = person + " " + action + " into " + numberOfPlaces
+ places
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

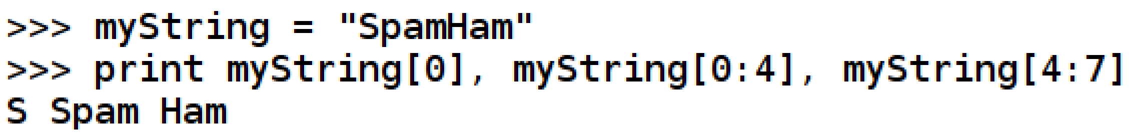


```
>>> places = "bars"
>>> numberOfPlaces = 2
>>> myString = person + " " + action + " into " + str(numberOfPl
aces) + " " + places
>>> print myString
A viking walks into 2 bars
```

A bit like casting in C++....

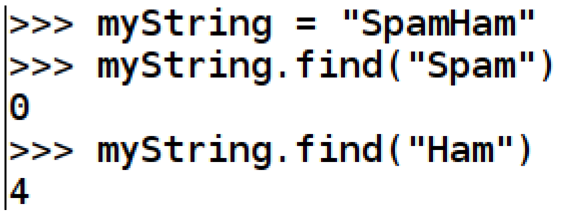
Strings in Python (2)

A string is an array of characters

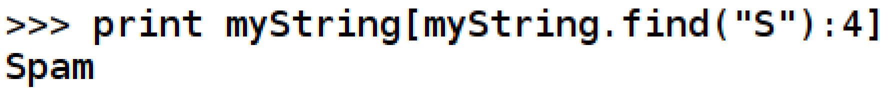


```
>>> myString = "SpamHam"
>>> print myString[0], myString[0:4], myString[4:7]
S Spam Ham
```

Finding substrings

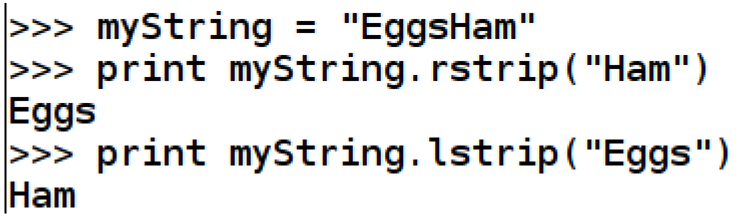


```
>>> myString = "SpamHam"
>>> myString.find("Spam")
0
>>> myString.find("Ham")
4
```



```
>>> print myString[myString.find("S"):4]
Spam
```

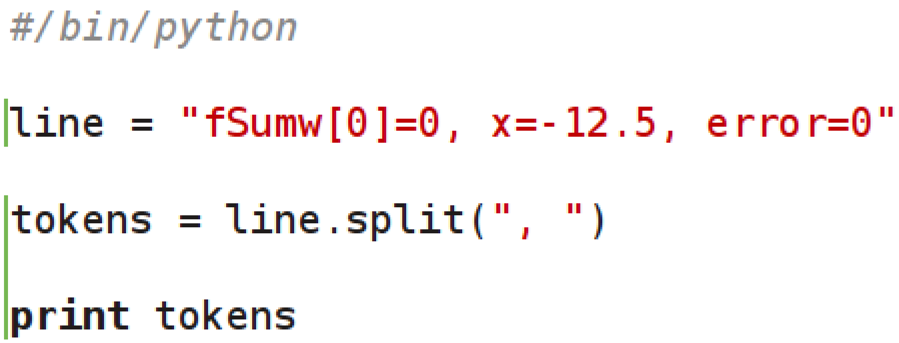
Removing parts of strings



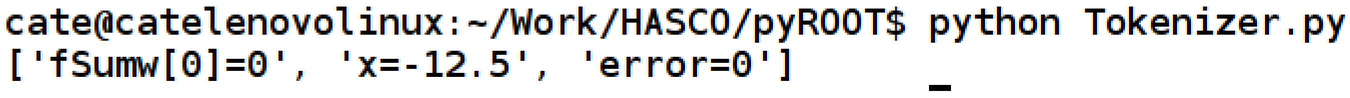
```
>>> myString = "EggsHam"
>>> print myString.rstrip("Ham")
Eggs
>>> print myString.lstrip("Eggs")
Ham
```

Strings in Python (3)

Tokenizing a string

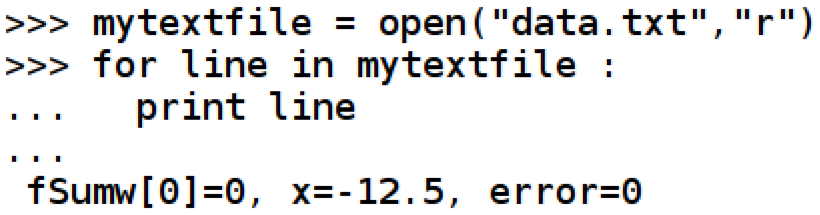


```
#/bin/python

line = "fSumw[0]=0, x=-12.5, error=0"

tokens = line.split(", ")

print tokens
```



```
cate@catelenovolinux:~/Work/HASCO/pyROOT$ python Tokenizer.py
['fSumw[0]=0', 'x=-12.5', 'error=0']
```

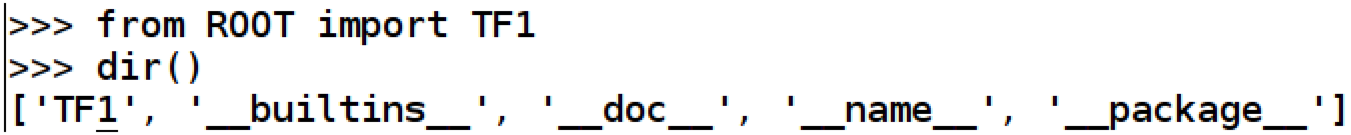
Getting a string from a text file



```
>>> mytextfile = open("data.txt","r")
>>> for line in mytextfile :
...   print line
...
 fSumw[0]=0, x=-12.5, error=0
```

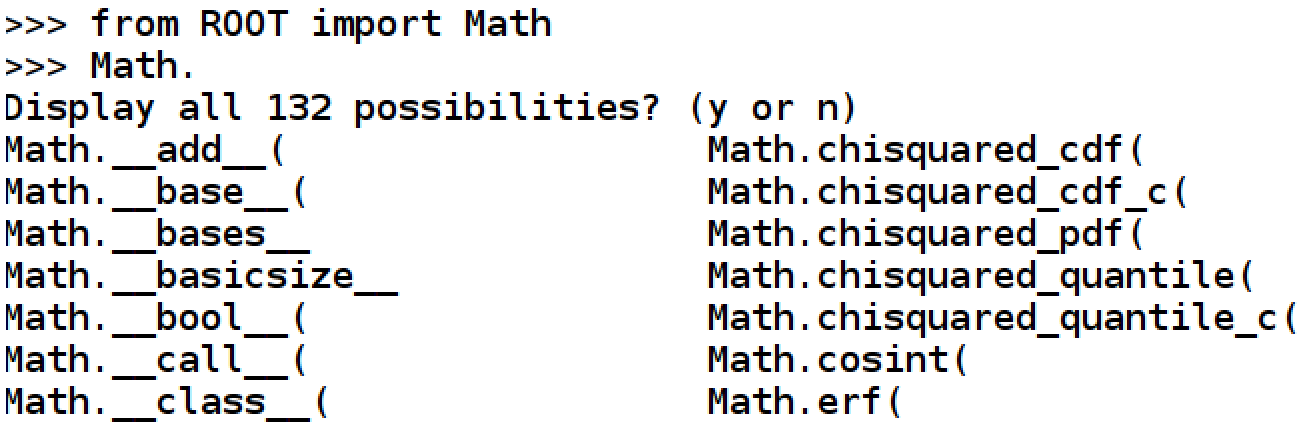
Essential pyROOT (1)

Import ROOT classes as modules (can check what there is with `dir()` function)



```
>>> from ROOT import TF1
>>> dir()
['TF1', '__builtins__', '__doc__', '__name__', '__package__']
```

Tab-completion works here as well:

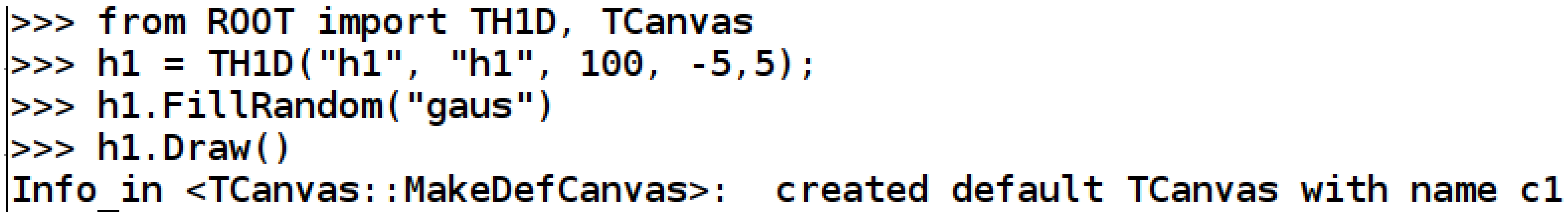


```
>>> from ROOT import Math
>>> Math.
Display all 132 possibilities? (y or n)
Math.__add__(                   Math.chisquared_cdf(
Math.__base__(                  Math.chisquared_cdf_c(
Math.__bases__                  Math.chisquared_pdf(
Math.__basicsize__              Math.chisquared_quantile(
Math.__bool__(                  Math.chisquared_quantile_c(
Math.__call__(                  Math.cosint(
Math.__class__(                 Math.erf(
```

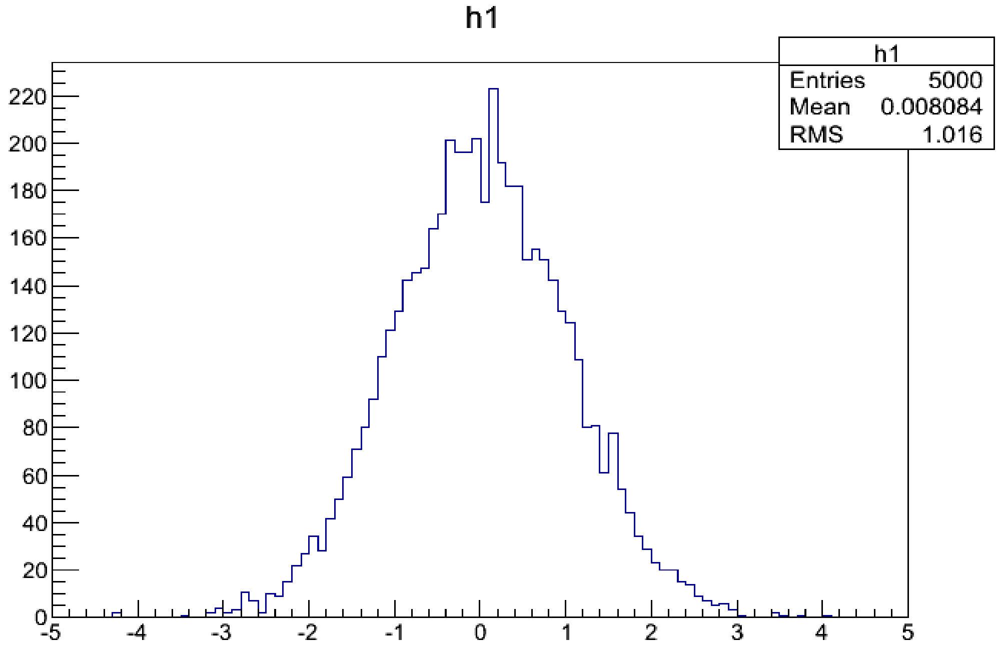
Essential pyROOT (2)

Instantiating an object in python works slightly differently wrt C++

In general, use ROOT classes in the same way as in CINT,
without worrying about **.** Vs → as in python everything is a reference

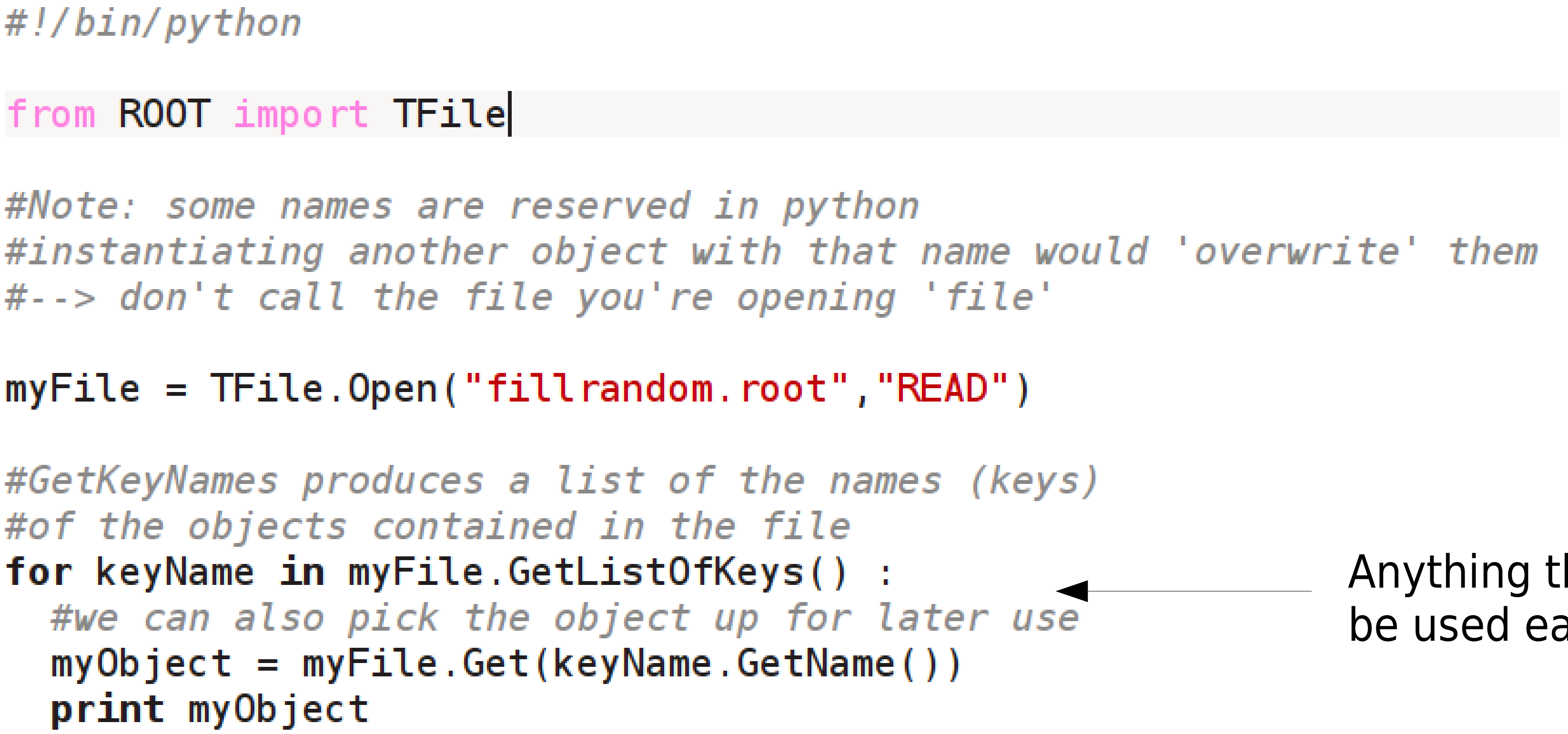


```
>>> from ROOT import TH1D, TCanvas
>>> h1 = TH1D("h1", "h1", 100, -5,5);
>>> h1.FillRandom("gaus")
>>> h1.Draw()
Info_in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```



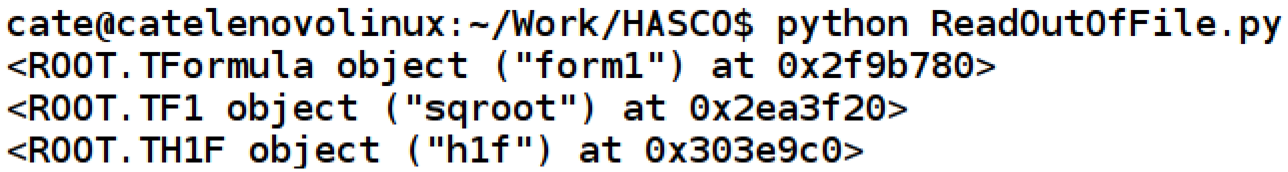
Reading objects out of a file

pyROOT advantage: easy use of TLists



```
#!/bin/python

from ROOT import TFile

#Note: some names are reserved in python
#instantiating another object with that name would 'overwrite' them
#--> don't call the file you're opening 'file'

myFile = TFile.Open("fillrandom.root","READ")

#GetKeyNames produces a list of the names (keys)
#of the objects contained in the file
for keyName in myFile.GetListOfKeys() :
  #we can also pick the object up for later use
  myObject = myFile.Get(keyName.GetName())
  print myObject
```

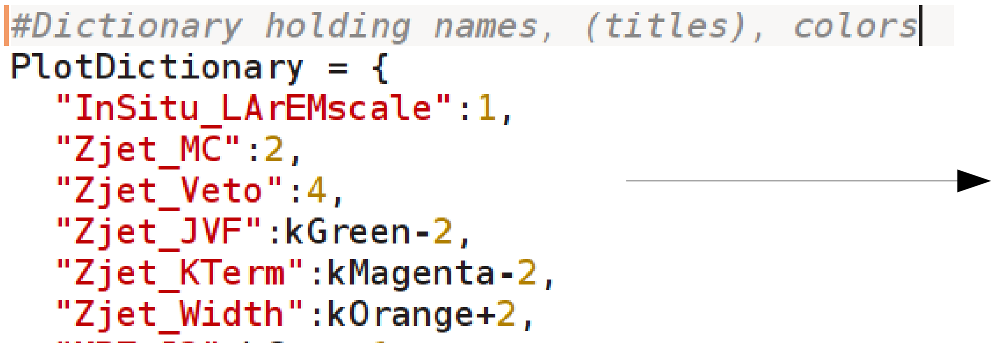
Anything that is a list can be used easily in a loop



```
cate@catelenovolinux:~/Work/HASCO$ python ReadOutOfFile.py
<ROOT.TFormula object ("form1") at 0x2f9b780>
<ROOT.TF1 object ("sqroot") at 0x2ea3f20>
<ROOT.TH1F object ("h1f") at 0x303e9c0>
```

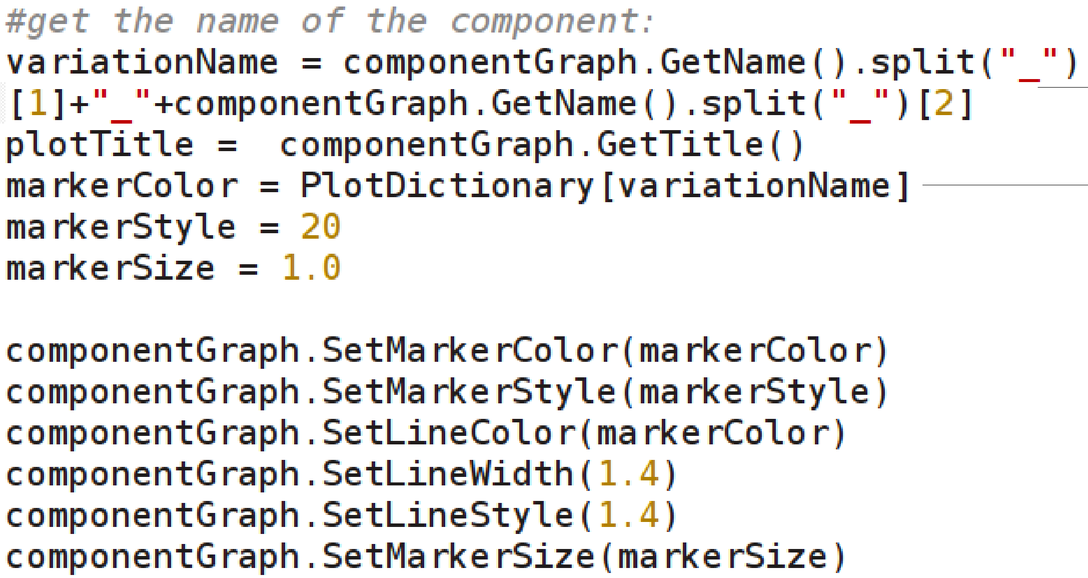
Formatting many histograms

An example of how I use dictionaries...



```
#Dictionary holding names, (titles), colors
PlotDictionary = {
  "InSitu_LArEMscale":1,
  "Zjet_MC":2,
  "Zjet_Veto":4,
  "Zjet_JVF":kGreen-2,
  "Zjet_KTerm":kMagenta-2,
  "Zjet_Width":kOrange+2,
```

Key: something that can be easily obtained from the graph name
Value: color of that plot



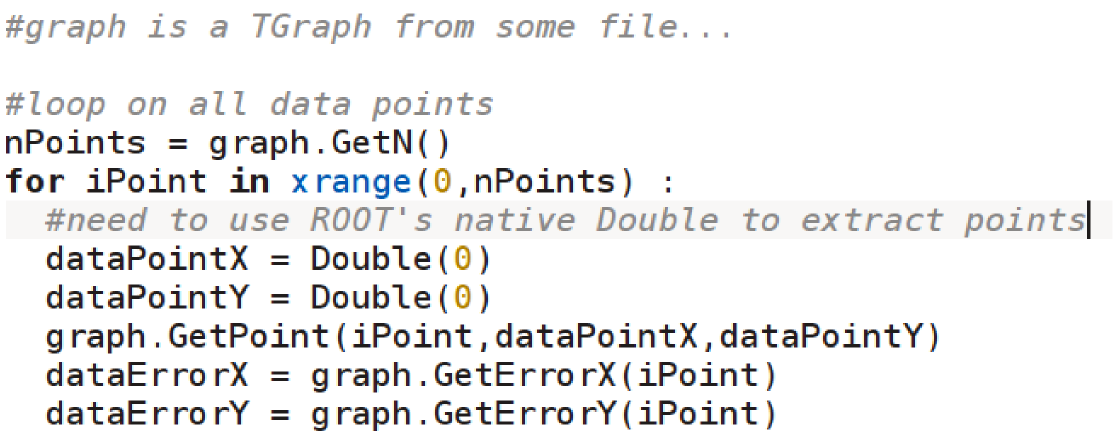
```
#get the name of the component:
variationName = componentGraph.GetName().split("_")
[1]+"_"+componentGraph.GetName().split("_")[2]
plotTitle =  componentGraph.GetTitle()
markerColor = PlotDictionary[variationName]
markerStyle = 20
markerSize = 1.0

componentGraph.SetMarkerColor(markerColor)
componentGraph.SetMarkerStyle(markerStyle)
componentGraph.SetLineColor(markerColor)
componentGraph.SetLineWidth(1.4)
componentGraph.SetLineStyle(1.4)
componentGraph.SetMarkerSize(markerSize)
```

Some string magic to obtain the 'key' above

Assigning the right color from the dictionary

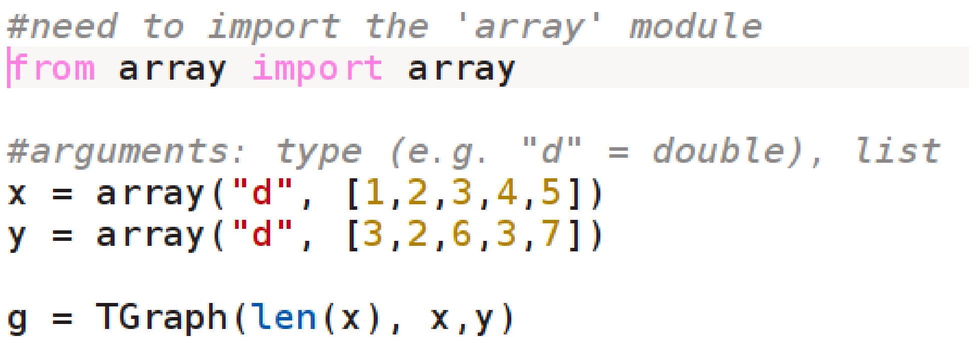
TGraphs

Reading out points from a TGraph



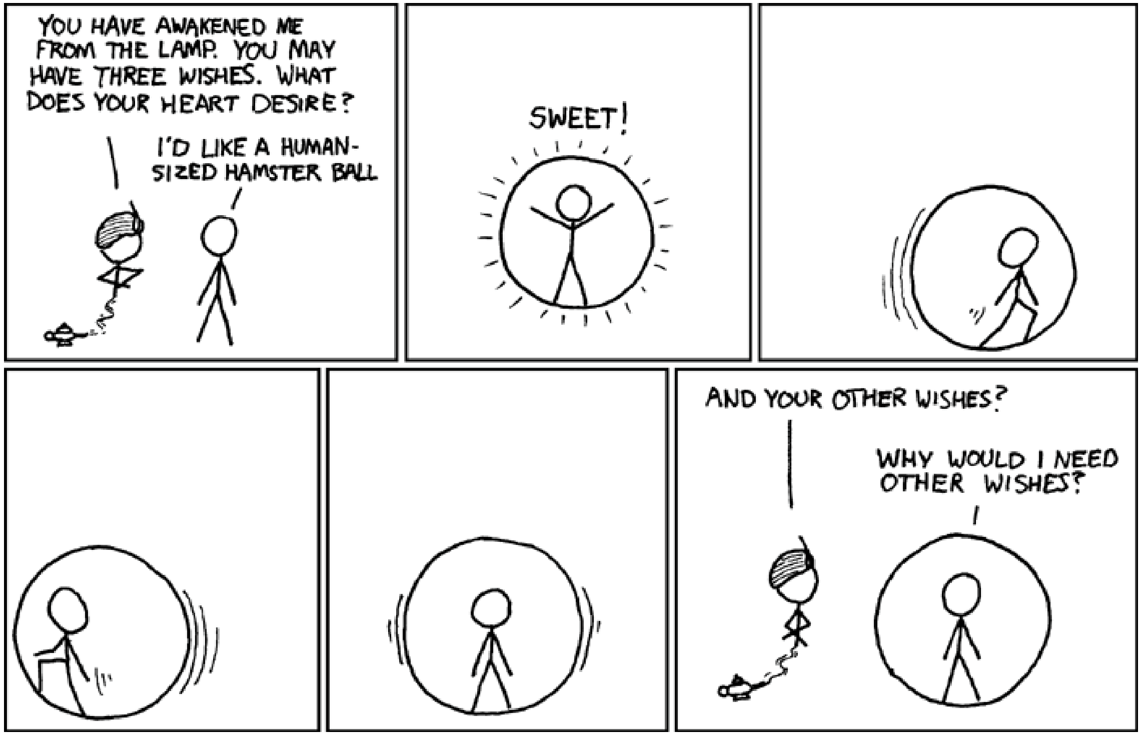
```
#graph is a TGraph from some file...

#loop on all data points
nPoints = graph.GetN()
for iPoint in xrange(0,nPoints) :
  #need to use ROOT's native Double to extract points
  dataPointX = Double(0)
  dataPointY = Double(0)
  graph.GetPoint(iPoint,dataPointX,dataPointY)
  dataErrorX = graph.GetErrorX(iPoint)
  dataErrorY = graph.GetErrorY(iPoint)
```

Creating a TGraph from ROOT arrays



```
#need to import the 'array' module
from array import array

#arguments: type (e.g. "d" = double), list
x = array("d", [1,2,3,4,5])
y = array("d", [3,2,6,3,7])

g = TGraph(len(x), x,y)
```

Enjoy the excursion!


YOU HAVE AWAKENED ME FROM THE LAMP. YOU MAY HAVE THREE WISHES. WHAT DOES YOUR HEART DESIRE?
I'D LIKE A HUMAN-SIZED HAMSTER BALL
SWEET!
AND YOUR OTHER WISHES?
WHY WOULD I NEED OTHER WISHES?