Beginner's python 2

Anja

LHCb Starterkit 2021

Modules

Provide functions/constants/data types/etc.

How do I get them?

- create a virtual environment
- install using `pip install`

⇒ in most cases it's easiest to take an environment from cvmfs, for example:

```
$ source /cvmfs/sft.cern.ch/lcg/views/LCG_97python3/x86_64-centos7-gcc8-opt/setup.sh
```

⇒ All we need today is in the LbEnv you should get by default on lxplus.

How do I use them?

```
1  import module
2  module.function() # function call
3
4  from module import f1, f2
5  f1() # function call
6
7  # import with an alias
8  import module.function as modfun
```

Modules

Examples

```
1  import argparse                    # handle command line arguments
2  import math                        # mathematical functions
3  import numpy as np                 # numerical computations
4  import matplotlib.pyplot as plt # plotting library
```

Often several modules contain functions that can do the same thing:

```
1  from math import cos,pi
2  cos(pi) # returns -1
3
4  from numpy import cos,pi
5  cos(pi) # returns -1
```

Tasks

1. Create 1000 random integers between 0 and 10 using `numpy.random.randint`.
2. Use numpy to calculate the mean and standard deviation of your sample.
3. Plot a histogram of your data.

Reaction Put the "thumbs up" in Zoom.

Bonus Draw events from a random Gaussian and repeat 1-3.
Change the number of bins in the histogram.
Move the legend to another place.
Change the colours in your plot.
How do you plot a data point with errorbars?

Useful commands

```
1  import numpy as np
2  # numpy random number generator
3  from numpy.random import randint
4
5  # plotting
6  import matplotlib.pyplot as plt
7  plt.hist(data,label="hist") # normalized histogram of data
8  plt.legend()                # plot the legend (based on the lables)
9  plt.xlabel("xlabel")
10 plt.ylabel("ylabel")
11 plt.savefig("filename.png") # save the plot
12 plt.close()                 # close the current plot
```

Classes

Example: we look at $X \to K\pi$ and want to calculate the invariant mass of X:

$$p_X^\mu = p_K^\mu + p_\pi^\mu \qquad \Rightarrow \qquad m_X^2 = p_X^\mu (p_X)_\mu$$

It would be nice if we could do something like this:

```
1  K  = FourVector(pxK, pyK, pzK, eK)  # define kaon vector
2  Pi = FourVector(pxPi,pyPi,pzPi,ePi) # define pion vector
3
4  X = K + Pi                          # add the vectors
5  m2_X = X.M2()                       # invariant mass squared
```

Classes

```
K  = FourVector(pxK, pyK, pzK, eK)  # define kaon vector
Pi = FourVector(pxPi,pyPi,pzPi,ePi) # define pion vector

X = K + Pi                          # add the vectors
m2_X = X.M2()                       # invariant mass squared
```

```
class FourVector:
    # a generic four-vector object
    # created by calling FourVector(px,py,pz,E)

    def __init__(self, px, py, pz, E):
        # this is the initializer and is automatically
        # called when a new object is created
        self.px   = px
        self.py   = py
        self.pz   = pz
        self.E    = E

    def M2(self):
        # calculate the invariant mass squared of the vector
        return self.E ** 2 - (self.px ** 2 + self.py ** 2 + self.pz ** 2)

    def __add__(self, other):
        # a function to add 4-vectors
        new_px = self.px + other.px
        new_py = self.py + other.py
        new_pz = self.pz + other.pz
        new_E  = self.E  + other.E
        return FourVector(new_px, new_py, new_pz, new_E)
```

Inheritance

In the detector, we measure the momentum, not the mass or the energy.
$\Rightarrow$ Define a class that is a 4-vector based on the momentum and a mass-hypothesis.

```
1   class Particle(FourVector):
2       # a 4-vector of a particle
3       # based on 3-momentum and a given mass
4
5       def __init__(self, px, py, pz, M):
6           # initializer for the Particle class
7           # first initialize a FourVector object
8           super().__init__(px,py,pz,0)
9           # then set the mass and calculate the
10          # energy based on the mass hypothesis
11          self.setmass(M)
12
13      def energy(self):
14          # calculate the energy
15          return np.sqrt(self.M ** 2 + self.px ** 2 + self.py ** 2 + self.pz ** 2)
16
17      def setmass(self,mass):
18          # set the mass variable
19          self.M = mass
20          # calculate the energy based on the given mass
21          self.E = self.energy()
```

Example

```
1   p = Particle(0,0,2,0.140) # set the p 4-vector for a pion
2   p.E                       # energy of a pion with momentum (0,0,2)
3   p.setmass(0.494)          # change the hypothesis to a kaon
4   p.E                       # energy of a kaon with momentum (0,0,2)
```

Tasks

1. Copy `/afs/cern.ch/user/a/anbeck/public/particle.py` to the directory you want to work in. You can now treat this like a module called `particle`:
   ```
   from particle import FourVector,Particle
   ```

2. Add a function to the FourVector class that returns the total momentum.

3. Create two particles with $\vec{p}_1 = (0, 0, 1), \vec{p}_2 = (0, 0, 3)$. Calculate the invariant mass of their parent assuming a) both are kaons and b) both are pions.

Reaction Put the "thumbs up" in Zoom.

Looking at data

Data usually comes in ROOT files.

A common python option is to load the data into a pandas DataFrame.

```
1  import uproot
2
3  # open the root file
4  f = uproot.open("filename.root")
5
6  # access a tree in the file
7  t = f["treename"]
8
9  # convert the tree into a pandas DataFrame object
10 df = t.arrays(library="pd")
```

A pandas DataFrame is a table with many useful tools to analyse and manipulate data.

Pandas

Access columns

```
1  df["column"]
```

Selections

```
1  df[df["column"]>0]                    # select rows with boolean array df["column"]>0
2  df.query("column > 0", inplace=True) # select rows based on string "column>0"
3  df[df["column"].isin(["A","B","C"])] # df[df["B_BKGCAT"].isin([0,10,50])]
```

inplace=True changes df, while inplace=False returns a new data frame

Adding new columns

```
1  df.eval("column = columnA + columnB", inplace=True) # add based on string
2  df.loc[:,"column"] = array                          # insert array
```

And many other things

```
1  # histogtram
2  plt.hist(df["column"])
3  df["column"].hist()
4
5  # mean
6  np.mean(df["column"])
7  df["column"].mean()
```

Tasks

1. Open this file and find the names of the tree and the branches:
   `root://eospublic.cern.ch//eos/opendata/lhcb/AntimatterMatters2017/data/B2HHH_MagnetDown.root`
   Hint: look at `f.keys()` to find the treename

2. Look at these commands: (in a script use `print(command)`)
   ```
   df.columns
   df.head(3)
   df["B_FlightDistance"]
   ```

3. Histogram the B flight distance and save the plot.

4. Add a column with the B momentum ($p_B = \left|\sum_h \vec{p}_h\right|$).
   Add a column with the B mass assuming that all hadrons are kaons (494 MeV).

5. How does the B mass distribution change when requiring `ProbK>0.1` for all hadrons?

   This removes events that are not $B \to KKK$ because a low `ProbK` means that the particle did not look very much lika a kaon in the detector.

Reaction Put the "thumbs up" in Zoom.

Bonus How do 5. and 6. change, when you assume that all hadrons are pions?

Solution in `/afs/cern.ch/user/a/anbeck/public/solution3.py`.

Command line arguments

Often, we want to use our scripts with several files, like MagnetUp and MagnetDown or different years of data taking.

Instead of hard-coding the filename in the script, we can give the filename as a command line argument.

```
1  import argparse
2
3  # define a parser
4  parser = argparse.ArgumentParser()
5
6  # add necessary argument
7  parser.add_argument("data",type=str,help="Input ROOT files.")
8
9  # add optional argument
10 parser.add_argument("--name",type=str,action="store",default="plot",
11                                help="File ending for the output.")
12
13 # put them in a list
14 arguments = parser.parse_args()
15
16 # use them
17 datafile_name = arguments.data
18 plot_suffix = arguments.name
```

This file is run using

```
$ python scriptname.py filename.root --name myName
```

Tasks

1. Add a necessary argument to hand over the filename.
2. Add an optional argument that takes a string and adds it as a suffix to the plots.
3. Run your script with the MagnetUp and the MagnetDown files in `root://eospublic.cern.ch//eos/opendata/lhcb/AntimatterMatters2017/data/`

Bonus Add `nargs="+"` to the filename argument. Now you can give one or several datafiles and `args.data` is a list. (You'll need to change how the data is read into the data frame.)

Solution in `/afs/cern.ch/user/a/anbeck/public/solution4.py`.

Log into lxplus: `$ ssh -X username@lxplus.cern.ch`

Then you have two options:

A) Interactive shell

`$ ipython`

You start straight away with immediate feedback.

B) Script

Open an editor and write a script

`$ emacs myFile.py &`

Then run it:

`$ python myFile.py`

To look at a plot do one of those:

Add `plt.show()` after creating the plot in the script

`$ eog picture.png &`

`$ evince picture.pdf &`

If LbEnv is not available, try

`$ source /cvmfs/sft.cern.ch/lcg/views/LCG_97python3/x86_64-centos7-gcc8-opt/setup.sh`

And to get the data frame, instead of

```
df = t.arrays(library="pd")
```

use

```
df = t.pandas.df()
```