Some exercises about accessing data with numpy

Most of the data analysis for high energy physics is done using a software toolkit called ROOT, which is mainly C++ and can be integrated with Python and R. It is quite cumbersome to install/run (at least for now). So we will use a package called uproot, that reads/writes ROOT files without having ROOT or C++.

So let's install it! In principle you can install uproot with Conda, but saves time to do this inside the notebook for this exercise.

Add ! in the beginning to run a shell command within the notebook.

In [ ]:
```
#Uncomment to try. This should be fine but safer the following way.
#!pip install uproot

import sys
!{sys.executable} -m pip install uproot
```

You should see something like this:

```
Collecting uproot
  Using cached https://files.pythonhosted.org/packages/a9/f0/ffd47865b5fb2693c78d5067343ea3fe1933b3042cc1e3380c2498c4bb7c/uproot-3.2.14-py2.py3-none-any.whl
Collecting numpy>=1.13.1 (from uproot)
  Using cached https://files.pythonhosted.org/packages/ff/7f/9d804d2348471c67a7d8b5f84f9bc59fd1cefa148986f2b745 52f8573555/numpy-1.15.4-cp36-cp36m-manylinux1_x86_64.whl
Collecting awkward>=0.5.6 (from uproot)
  Using cached https://files.pythonhosted.org/packages/03/8d/62912260aed6de6bca1ea818e028b181fa797f35c3a3d47153a7b57be339/awkward-0.5.6-py2.py3-none-any.whl
Collecting cachetools (from uproot)
  Using cached https://files.pythonhosted.org/packages/76/7e/08cd3846bebeabb6b1cfc4af8aae649d90249b4aeed080bddb5297f1d73b/cachetools-3.0.0-py2.py3-none-any.whl
Collecting uproot-methods>=0.2.0 (from uproot)
  Using cached https://files.pythonhosted.org/packages/df/20/0718588689e3b6b408ba11beb401f3d9f8fefc92efb5de9253140d4575e6/uproot_methods-0.2.10-py2.py3-none-any.whl
Collecting lz4 (from uproot)
Installing collected packages: numpy, awkward, cachetools, uproot-methods, lz4, uproot
Successfully installed awkward-0.5.6 cachetools-3.0.0 lz4-2.1.2 numpy-1.15.4 uproot-3.2.14 uproot-methods-0.2.10
```

Import uproot and numpy

In [ ]:
```
import uproot as up
import numpy as np
```

Next up is accessing the ROOT file. In the zip file you have an example Monte Carlo sample with 10000 events. We will work with bigger samples later. We look at the variables stored in the ROOT file (there are a lot of them here, we will discuss them later).

In [ ]:
```
file = up.open("skim_BsToJPsiPhi_2017_DCAP_10000.root")
print(file.keys())
data = file["PDtree"]
print(data.keys())
```

We would like to have these set of variables as numpy arrays, for example number of tracks,track transverse momentum, track eta,track phi and track charge.

In [ ]:
```
ntracks = data.arrays(["nTracks"]) #this is now a Python dictionary
print(type(ntracks))
print(ntracks.keys())
Ntracks = ntracks[b'nTracks'] # this is now a numpy array
print(type(Ntracks),Ntracks.shape)
```

Other track variable arrays to look at:

In [ ]:
```
TrkPt = data.arrays(["trkPt"])[b'trkPt']
TrkEta = data.arrays(["trkEta"])[b'trkEta']
TrkPhi = data.arrays(["trkPhi"])[b'trkPhi']
TrkCharge = data.arrays(["trkCharge"])[b'trkCharge']
```

In some cases we have arrays of arrays and these arrays (inside the main array) don't necessarily have the same length, so cannot be expressed as numpy arrays. Uproot loads them into jagged arrays. Now what does that mean? Let's have a look at one of them.

In [ ]:
```
print(type(TrkEta))
for i,eta in enumerate(TrkEta[:5]): #first 5 arrays
    print('##################### event number: ',i+1) #to make it visible
    for eta2 in eta:
        print(eta2)
```

Jagged arrays are backed by numpy so we can do operations on them pretty fast and still preserve the same structure.

Numpy exercise: Calculation with numpy arrays.

In [ ]:
```
#TO DO: Create two arrays with the z impact parameter of tracks (variable
#       (variable: tipEz). Divide tipDz values by the tipEz values.
TipDz = data.arrays(["tipDz"])[b'tipDz']
TipEz = data.arrays(["tipEz"])[b'tipEz']
print(TipDz.shape)
print(TipEz.shape)
s = TipDz/TipEz
print(s.shape, type(s)) #the result is another jagged array with the same


#TO DO: Get the maximum (minimum/mean/etc) of all elements in the TrkPt ja
TrkPt_flat = np.concatenate(TrkPt)
TrkPt_flatmax = np.max(TrkPt_flat)
print(TrkPt_flatmax)

#TO DO: Get maximum Pt per event (per array) in the TrkPt jagged array (us
#       and not np.max). What shape does the result have? Divide the jagge
TrkPt_max = TrkPt.max()
print(TrkPt_max.shape)
TrkPt_scale = TrkPt/TrkPt_max
```

Let's have a look at muon variables.

In [ ]:
```
Nmuons = data.arrays(["nMuons"])[b'nMuons']
MuoE = data.arrays(["muoE"])[b'muoE']
print(Nmuons[5]) #element registered in the 6th array
print(len(MuoE[5])) #number of elements
```

In [ ]:
```
MuoPt = data.arrays(["muoPt"])[b'muoPt']
MuoEta = data.arrays(["muoEta"])[b'muoEta']
MuoPhi = data.arrays(["muoPhi"])[b'muoPhi']
MuoCharge = data.arrays(["muoCharge"])[b'muoCharge']
```

Matplotlib exercise: Visualize some events.

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
#TO DO: Plot MuoPt vs MuoEta, MuoPt vs MuoPhi and MuoEta vs MuoPhi for sor
#       or plt.scatter). I set the marker size for the last plot with the
#       the marker.

event_range = range(0,15)
plt.figure(figsize=(16,3))
plt.subplot(131)
[plt.scatter(MuoEta[e],MuoPt[e],c = 'b') for e in event_range]
plt.ylabel("Muon Pt")
plt.xlabel("Muon Eta")
plt.grid()
plt.subplot(132)
[plt.scatter(MuoPhi[e],MuoPt[e],c = 'b') for e in event_range]
plt.ylabel("Muon Pt")
plt.xlabel("Muon Phi")
plt.grid()
plt.subplot(133)
[plt.scatter(MuoEta[e],MuoPhi[e],c = 'b', s = MuoPt[e]*10) for e in event_
plt.xlabel("Muon Eta")
plt.ylabel("Muon Phi")
plt.grid()
plt.show()
```

My solution: