Importing Root:

```
from ROOT import TFile
```

Opening files:

```
inputFile = TFile("input/path/file.root", "READ")
```

Reading ROOT **file** contents:

```
inputFile.ls()
```

Getting and looking at **TTrees**:

```
ttreeVar = inputFile.Get("desiredTTree") #the names are listed from line 3
ttreeVar.ls() #does't give very helpful information
ttreeVar.Print() #should work but doesn't always

for branch in ttreeVar.GetListOfBranches(): #this does work always, it seems
    print(branch.GetName())
```

Starting to print histograms:

```
from ROOT import TCanvas
c1 = TCanvas("name", "title", widthnum, heightnum)

ttreeVar.Draw("branchName")
c1.Draw()
```

We can also import the histogram class and begin to do better:

```
from ROOT import TH1D #the histogram class
from ROOT import kRed #a colour :)

hist1 = TH1D("name", "title;x-axis name; y-axis name", bins, minimum, maximum)
hist2=TH1D("name2", "title2;x-axis name; y-axis name", bins, minimum, maximum) #for
    example
ttreeVar.Draw("branchName>>hist1")
ttreeVar.Draw("branch2Name>>hist2")

c2 = TCanvas("name", "title", widthnum, heightnum)
hist1.Draw() #keeps default color
hist2.SetLineColor(kRed)
hist2.Draw("same") #draws on same canvas rather than overwriting

c2.Draw()
```

In addition, it is possible to subtract event details in the histogram, for example

```
ttreeVar.Draw("branchName - branch2Name>>hist1")
```

And to put multiple histograms on one plot. We use a very "nice" cd method

```
c4 = TCanvas("c4", "c4", 1000, 500)
c4.Divide(columns, rows)
c4.cd(1) #go to the first box
ttreeVar.Draw("branchName")
c4.cd(2)
ttreeVar.Draw("branchName2")
c4.Draw()
```

To go about it in a weird way, we can cull values from the histogram itself

```
hist3=TH1D("hist3", "title", bins, minimum, maximum)
ttreeVar.Draw("branchName-branchName2>>hist3", "branchName-branchName2 < upperLimit")
mean=hist3.GetMean()
print(mean, "ns")
```

Or we can simply do our calculations in the histogram argument and read the mean or whatever on the output. The application of the above is that we complete the following workflow

1. Using the detectors placed at 0 distance, we can read their time display by making a histogram and subtracting their values (each is a TBranch) and reading the mean
2. We can then place them at an unknown distance x and add the mean 0 delay on, to get the ToF
3. We can then calculate from the mean or in the histogram argument itself the distance
4. If there are some data we want to cut off then we can use as in line **38**

Recall that ROOT is a **column-oriented DBMS** (database management system), meaning that when we define (as is standard) columns as *types* of data, and esp. in our case rows as *events*, we put together all the events from a column together (e.g. we read top to bottom and then right to left).

However, we are still interested in getting particular events. The easiest way is to iterate using a for loop

```
i=0 #In exemplar cases, we want a counter so we don't print all 300,000 events
for event in ttreeVar:
    print(j.branchName)
    i+=1
    if i>5:
        break
```

There are instances where j.branchName will not be a printable object, but rather some sort of array. This can be easily countered

```
for event in ttreeVar:
    for j in event.branchName:
        print(j)
```

Or in a more complicated format:

```
counter = 0
myArray = []
for event in step1:
    print("Event", counter, "has", event.pixel0_numHits, "hits")
    for idx in range(event.pixel0_numHits):
        print("  Hit", idx, ": X =", event.pixel0_hitX[idx], " Y =", event.pixel0_hitY
            [idx])
    print("  Cluster X =", event.pixel0_clusterX, " Y =", event.pixel0_clusterY)
    counter += 1
    myArray += [event.pixel0_clusterX]
    if counter == 5:
        break

print(myArray)
```

Pretend we want to iterate through a bunch of branches named "pixel1_clusterX", "pixel2_clusterX", ... , "pixel5_clusterX. We can either write it all or

```
pixelList= []
index_var=0
for event in ttreeVar:
    pixelList[index_var]=[]
```

```
    for j in range(1,6):
        pixelList.append(getattr(event, "pixel" + str(j)+"_clusterX"))
    index_var+=1
```

The important part of this is we can invoke `getattr(event, "branchName")`, which, because now it is a string name and not hard-coded, allows us more flexibility.
Back to histograms. When aligning, we want to see correlations. This is relatively easy to do with 2-d histograms:

```
ttreeVar.Draw("branchName:branch2Name")
#Or with conditions that both have a hits
ttreeVar.Draw("branchName:branch2Name", "branchName_len()>0 && branch2Name_len()>0")
#note that _len() is not a real method
```

But why not add some color? There is a whole list of different options you can choose at this url: `https://root.cern/doc/master/classTHistPainter.html#HP01a`. A good option is `colz`.