Phys105 – Week 3

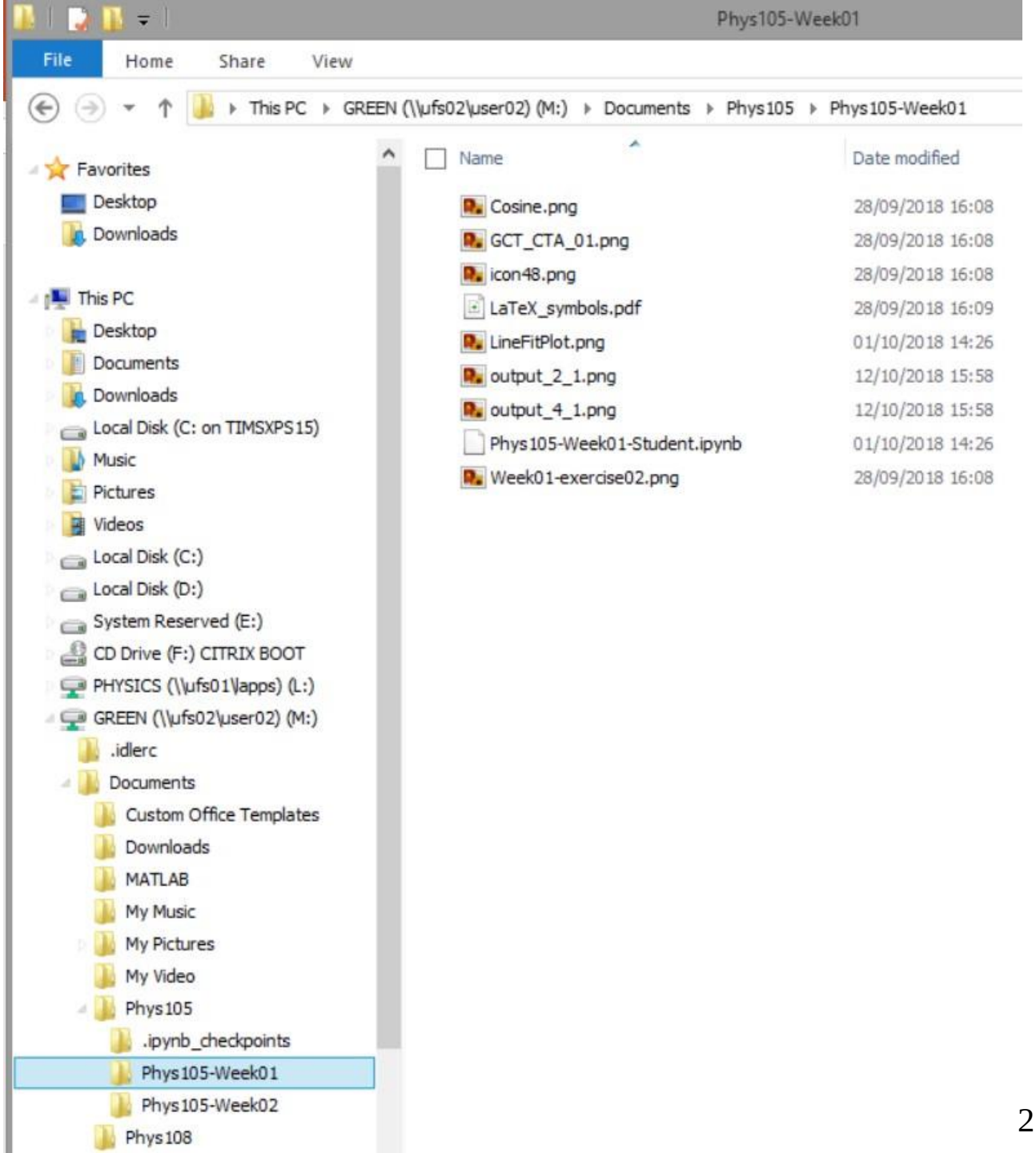
News

- This week:
  - Comments from last week:
    - Uploading files to CoCalc.
    - Images and figures.
    - Switching cell types.
  - Directories/folders:
    - Windows
    - Mac
    - Linux
    - CoCalc
  - Lists, tuples and NumPy arrays.
  - Histograms take one.

- Joe Price's Office Hours:
  - Wednesdays, 10:00...11:00.
- Email to book a slot!
- (And if you can't get to our Office Hours, email and we'll find an alternative!)

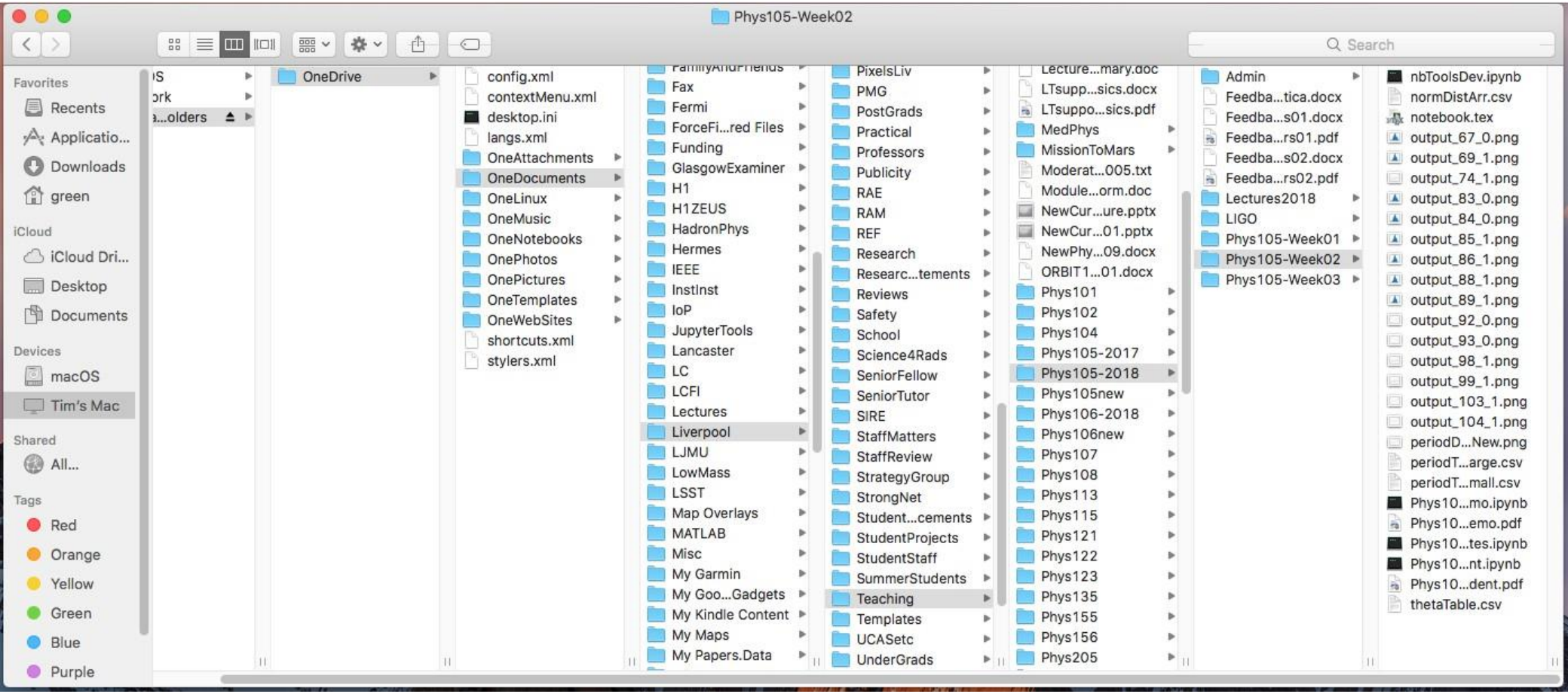
Directories on Windows

- Use *File Explorer*.
- Access via *Start*, *Windows System*, *File Explorer.*
- In *File Explorer*, click *View* and select *Navigation Pane*.
- Shows you all folders in “tree” on left and selected folder’s contents on right.
- Can move (or copy) files on the right into folders on the left…



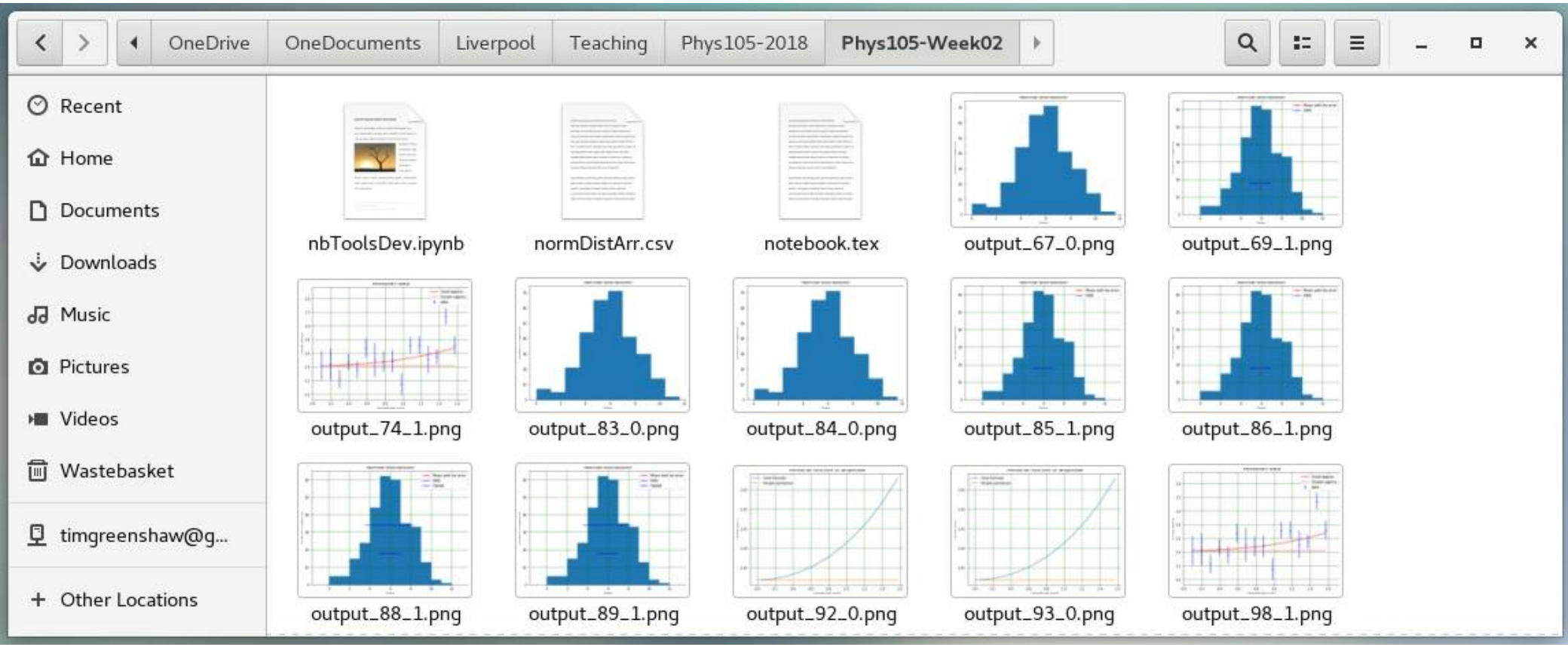
Directories on a Mac

- Use *Finder*, can start from *Dock*.
- Choose columns view 



Directories on Linux

- Many different flavours of Linux!
- Example here is using CentOS 7.
- Select *Applications*, *Accessories*, *Files*.



More data types – lists

- In addition to the data types *float* and *int*, Python has a type called *list*.
- You can make a list and access its elements as follows:

```
[81]:
1 thisList = [1, 2.278, -8, "cheese", 'blue ', np.pi]
2 print("thisList =",thisList)
3 print("thisList[0] =",thisList[0])
4 print("thisList[0] + thisList[1] =",thisList[0] + thisList[1])
5 print("thisList[4] + thisList[3] =",thisList[4] + thisList[3])
```

```
thisList = [1, 2.278, -8, 'cheese', 'blue ', 3.141592653589793]
thisList[0] = 1
thisList[0] + thisList[1] = 3.278
thisList[4] + thisList[3] = blue cheese
```

- Effect of "+" depends on types of list elements, get error if + doesn't make sense!

```
[17]:
1 print("thislist[0] + thisList[3] =",thisList[0] + thisList[3])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-baff7e321135> in <module>
----> 1 print("thislist[0] + thisList[3] =",thisList[0] + thisList[3])

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Lists

- Lists can be modified, remember we had:

```
thisList = [1, 2.278, -8, 'cheese', 'blue ', 3.141592653589793]
```

- Change element 4 (the fifth in the list!):

```
[28]: 1 thisList[4] = "cream"
      2 print("thislist[4] + thisList[3] =",thisList[4] + thisList[3])

thislist[4] + thisList[3] = creamcheese
```

- If we want a space between “cream” and “cheese”, we have to either modify one of the strings (to “cream ” or “ cheese”) or put in a space explicitly:

```
[29]: 1 thisList[4] = "cream"
      2 print("thislist[4] + thisList[3] =",thisList[4] + " " + thisList[3])

thislist[4] + thisList[3] = cream cheese
```

- Note that *print* adds spaces…

```
[33]: 1 print("thislist[4] =",thisList[5])

thislist[4] = 3.141592653589793
```

Tuples

- Tuples are similar to lists.
- One difference is that they are written using round brackets, not square.
- The second is that while lists can be modified (are *mutable*), tuples cannot be changed once they have been defined (they are *immutable*).

[83]:
```
1 thisTuple = (3.14, -9, "orange")
2 #
3 print("thisTuple[2] =",thisTuple[2])
```

```
thisTuple[2] = orange
```

[84]:
```
1 #
2 thisTuple[2] = 27
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-84-f51f3769bdbf> in <module>()
      1 # <!-- Cell 36 -->
      2 #
----> 3 thisTuple[2] = 27

TypeError: 'tuple' object does not support item assignment
```

NumPy arrays

- One of most useful data types for scientific computing is the numpy array.
- These can be created and filled in many ways, for example:

```
[3]: length = 3
     arrayOfZeros = np.zeros((3))
     print("arrayOfZeros =",arrayOfZeros)

     arrayOfZeros = [0. 0. 0.]
```

or

```
[3]: length = 3
     arrayOfZeros = np.zeros(3)
     print("arrayOfZeros =",arrayOfZeros)

     arrayOfZeros = [0. 0. 0.]
```

```
[24]: 1 length = 3
      2 start = 10
      3 stop = 30
      4 arrayOfNumbers = np.linspace(start, stop, length)
      5 print("arrayOfNumbers",arrayOfNumbers)

      arrayOfNumbers [10. 20. 30.]
```

- All elements must have same type.
- Access elements in same way as for list:

```
[26]: 1 print("arrayOfNumbers[1]",arrayOfNumbers[1])

      arrayOfNumbers[1] 20.0
```

NumPy arrays

- Arrays can be multi-dimensional:

```
[7]: nRows = 3
     nCols = 2
     matrix = np.ones((nRows, nCols))
     print("matrix = \n",matrix)
```

```
matrix =
 [[1. 1.]
 [1. 1.]
 [1. 1.]]
```

- More than one assignment per line is possible (but only do this when it results in understandable code!):

```
[9]: matrix[0, 0], matrix[0, 1] = 11, 12
     matrix[1, 0], matrix[1, 1] = 21, 22
     matrix[2, 0], matrix[2, 1] = 31, 32
     print("matrix = \n",matrix)
```

```
matrix =
 [[11. 12.]
 [21. 22.]
 [31. 32.]]
```

- Check the dimensions of an array using *shape*:

```
[10]: matrix.shape
```

```
[10]: (3, 2)
```

- Notice that shape returns a tuple, so we can look at its elements:

```
[13]: nRows, nCols = matrix.shape
      print("nRows =",nRows,"nCols =",nCols)
```

```
nRows = 3 nCols = 2
```

- See that sometimes tuples come without the () brackets.

Slicing arrays

- Can access range of elements using *slices:*

[97]:
```
1  countArr = np.linspace(0, 10, 11)
2  print("countArr =",countArr)
3  print("countArr[3:6] =",countArr[3:6])
```

```
countArr = [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
countArr[3:6] = [3. 4. 5.]
```

- Also works in many dimensions:

[98]:
```
1  print("matrix \n",matrix)
2  print("matrix[1:3, 0:2] \n",matrix[1:3, 0:2])
```

```
matrix
 [[11. 12. 13. 14.]
 [21. 22. 23. 24.]
 [31. 32. 33. 34.]]
matrix[1:3, 0:2]
 [[21. 22.]
 [31. 32.]]
```

Writing out and reading in arrays

- If have an array…

```
normDistArr
 [3.61731068 4.93440785 6.37500914 7.66854973 4.46305934 5.67756143
 6.61728604 5.10273029 5.4611186  6.36901216 2.87407419 3.09351455
 4.02478352 9.55890952 7.37325827 2.82026807 4.64787364 0.70393879
 8.11556904 5.70015363 7.19019305 3.7100883  9.19654448 5.75011898
 6.9846188  5.31088185 6.59710147]
```

- …can write it to a file in many ways, for example:

```
[34]:  1  np.savetxt('normDistArr.csv', normDistArr, delimiter = ',')
```

- Can then open in Excel (see right!), Notepad etc.
- Can read data from file into numpy array as follows:

```
[114]:  1  gaussArr = np.loadtxt("normDistArr.csv")
```

	A
1	3.62E+00
2	4.93E+00
3	6.38E+00
4	7.67E+00
5	4.46E+00
6	5.68E+00
7	6.62E+00
8	5.10E+00
9	5.46E+00
10	6.37E+00
11	2.87E+00
12	3.09E+00
13	4.02E+00
14	9.56E+00
15	7.37E+00
16	2.82E+00
17	4.65E+00
18	7.04E-01
19	8.12E+00
20	5.70E+00
21	7.19E+00
22	3.71E+00
23	9.20E+00
24	5.75E+00
25	6.98E+00
26	5.31E+00
27	6.60E+00

Histograms

- Look at distribution of data using histogram.
- Displays frequency of occurrence of values in data.
- Simplest version chooses bin widths, format of plot etc. automatically…
- …but you can also set your preferred bin widths and alter many aspects of the display of your histograms.

```
binBot = -1.0
binTop = 13.0
binNumber = 14
binEdges, binWidth = np.linspace(binBot, binTop, binNumber + 1, retstep = True)
print("Histogram bins start at",binBot,"finish at",binTop)
print("Number of bins is",binNumber,"and width of bins is",binWidth)
#
nEvents = len(gaussArr)
mu = np.mean(gaussArr) # calculate arithmetic mean of numbers in array
sigma = np.std(gaussArr) # calculate standard deviation (error on single value)
muError = sigma/np.sqrt(nEvents) # calculate error of mean
yMu = nEvents/20
ySigma = 1.2*nEvents/20
#
plt.figure(figsize = (7, 5))
plt.title('Normal distribution', fontsize = 14)
plt.xlabel('Data')
plt.ylabel('Relative frequency')
plt.hist(gaussArr, bins = binEdges, color = 'b')
plt.errorbar(mu, yMu, xerr = muError, marker = '+', color = 'r', label = 'Mean with its error')
plt.errorbar(mu, ySigma, xerr = sigma/2, marker = '', color = 'y', label = 'RMS')
plt.grid(color = 'g')
plt.legend()
plt.show()
```

```
Histogram bins start at -1.0 finish at 13.0
Number of bins is 14 and width of bins is 1.0
```