Introduction to Python with Google Colab. I have found that many students only know how to do calculations and make plots using Excel. I am a reasonable person, so I will not require that you adopt something else. Still, I find Excel can be quite limiting, and many students have wished they knew how to use Matlab, Mathematica, Python, R, or some other solution.

Here is my attempt to provide a (hopefully gentle) on-ramp to using python. So you don't have to install anything, we will run everything remotely using a free tool called Google Colab.

1. Go to the Google Colab file called PlottingDemo.ipynb. The first block of code is reproduced below:

```
# First note that anything preceded by ``#'' is a comment and will be ignored
# by the computer.

# We start by importing libraries
# import libraryname as lb
# would import a library named libraryname. When you use functions that
# library defines, you would need to type libraryname.function(). Because
# it is tedious to type out libraryname over and over again, it is common to
# import the library with a different shortened name, in this case ``lb''

# matplotlib and particularly pyplot are pretty common plotting libraries
import matplotlib as mpl
import matplotlib.pyplot as plt
# numpy is numerical python. It gives you a bunch of mathematical functions,
# including, for example, the ability to work with vectors and matrices
import numpy as np

# Set the default plotting style
plt.style.use('default')

# Let's first just plot ln(x)

# One way to do that is to generate a vector of x values from 0.1 to 10 in
# increments of 0.1

x = np.arange(0.1, 10, 0.1)
y = np.log(x)
```

   (i) **Figure out how to run this block of code and check what the values of x and y are (you may want to use the print() function).**

   (ii) **How long is the vector x?**

   (iii) **The vector y?**

2. The next block of code generates the plot.

```
# Let's make our first plot...

plt.plot(x, y, label=r'$y = \log(x)$')
plt.xlabel(r'$x$', fontsize=18)
plt.ylabel(r'$\log(x)$', fontsize=18)
plt.title(r'$\log(x)$ from $x=$'+str(np.min(x))+' to '+str(np.max(x)), fontsize
    =20)
plt.legend(loc=''lower right'')
```

(i) **Suppose I only wanted to make the plot with no text labels. Identify which part of which line would be necessary to do so (go ahead and test it out!).**

(ii) **Why are there weird dollar signs?**

(iii) **In line 6 of this block, what is the purpose of the single quote symbol '? What is the purpose of str()? What is the purpose of +?**

3. You can also use numpy to generate random numbers. For example, we can draw random numbers from a Gaussian as follows.

```
# Draw random numbers from a Gaussian distribution

mean = 0
spread = 1
num_samples = 20

# I always forget if the function wants a mean, variance, and number of samples
# or if it wants a mean, *standard deviation*, and the number of samples
# We will figure that out...
random_samples = np.random.normal(mean, spread, num_samples)

# Use matplotlib's hist function to plot a histogram with automatic bins
plt.hist(random_samples)
```

(i) **Run the code to see a histogram based on the twenty random samples from the Gaussian distribution. Save the image as both a .png and a .pdf file (to convince yourself that you can. Because you are running python remotely on Google Colab, this part is slightly more awkward than running on your computer. As an example, the following block downloads an .eps file for me.**

```
plt.savefig('HistDemo.eps',format='eps',transparent=True)
from google.colab import files
files.download('HistDemo.eps')
```

[Note: If you do not know the difference between a "rasterized" graphics file like a .jpg or .png and a "vector" graphics file like a .pdf or .eps, you might decide to learn about it. Chosing the right format for the right job can significantly improve your presentation of your work in papers, posters, and talks, so it's worth learning the basics.]

(ii) **Make a new block of code to generate 5000 random numbers from the Gaussian distribution and plot a histogram of those numbers. By adjusting the spread variable, figure out if the np.random.normal() function expects spread to be the variance or the standard deviation.**

(iii) **You could alternatively figure out what np.random.normal() expects by looking it up in the numpy documentation. Figure out how to do so and confirm that you were correct in (ii).**

(iv) **Congratulations, you are hopefully now in a position to understand the reference code that I gave you to demonstrate how to do calculations with a biased coin flip.**