In-class exercise for tutorial 009

For the first part of this exercise, you can stay in this notebook and work through it.

Working with Python lists

Make a new Python list. You can put whatever you want in it. Make it at least 5 items long.

In [ ]:
```
my_list = [1,2,3,4,5,6,7,8,9]
```

Get the first element of your new list.

In [ ]:
```
my_list[0]
```

Get the last element of your new list in a way that wouldn't depend on list length.

In [ ]:
```
my_list[-1]
```

Get all but the last two elements in a way that wouldn't depend on list length.

In [ ]:
```
my_list[:-2]
```

Get every other element of your list.

In [ ]:
```
my_list[0:len(my_list):2]
```

Working with numpy arrays

In [2]:
```
import numpy as np
```

Read in our first data file.

Make sure you have downloaded the two exercise files into your datasets folder

Numpy has its own file format for saving numpy arrays. Here, we'll load an existing array. This call should fail for you as is – see if you can fix it!

In [3]:
```
data_1 = np.load('/Users/paulwen/OneDrive - The University of Texas at Aust
```

Find out the shape of your new array.

In [4]:
```
data_1.shape
```

Out[4]:
```
(10, 10)
```

If your new array is not ***too*** huge, take a peek at it so you know what you're dealing with.

In [5]:
```
data_1[0:]
```

Out[5]:
```
array([[  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.],
       [  2.,   4.,   6.,   8.,  10.,  12.,  14.,  16.,  18.,  20.],
       [  3.,   6.,   9.,  12.,  15.,  18.,  21.,  24.,  27.,  30.],
       [  4.,   8.,  12.,  16.,  20.,  24.,  28.,  32.,  36.,  40.],
       [  5.,  10.,  15.,  20.,  25.,  30.,  35.,  40.,  45.,  50.],
       [  6.,  12.,  18.,  24.,  30.,  36.,  42.,  48.,  54.,  60.],
       [  7.,  14.,  21.,  28.,  35.,  42.,  49.,  56.,  63.,  70.],
       [  8.,  16.,  24.,  32.,  40.,  48.,  56.,  64.,  72.,  80.],
       [  9.,  18.,  27.,  36.,  45.,  54.,  63.,  72.,  81.,  90.],
       [ 10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.]])
```

Now let's practice grabbing specific bits out of our array.

Get the last row.

In [6]:
```
data_1[-1]
```

Out[6]:
```
array([ 10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.])
```

Get the second column (remember that indexes start at 0!).

In [11]:
```
data_1[:,1]
```

Out[11]:
```
array([ 2.,  4.,  6.,  8., 10., 12., 14., 16., 18., 20.])
```

Get every other row of the first two columns.

In [13]:
```
data_1[::2,:2]
```

Out[13]:
```
array([[ 1.,  2.],
       [ 3.,  6.],
       [ 5., 10.],
       [ 7., 14.],
       [ 9., 18.]])
```

Store the the second column in its own varable.

In [14]:
```
new_variable = data_1[:,1]
```

Look at the variable you just created.

In [15]:
```
new_variable
```

Out[15]:
```
array([ 2.,  4.,  6.,  8., 10., 12., 14., 16., 18., 20.])
```

Determine its type (using `type()`).

In [16]:
```
type(new_variable)
```

Out[16]:
```
numpy.ndarray
```

Okay, that's nice – when you grab a subset of a numpy array, the result is also a numpy array with superpowers included!

Multiply every element in this array by 2.

In [17]:
```
new_variable * 2
```

Out[17]:
```
array([ 4.,  8., 12., 16., 20., 24., 28., 32., 36., 40.])
```

Store the last column in another variable.

In [19]:
```
last_col = data_1[:,-1]
last_col
```

Out[19]:
```
array([ 10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.])
```

Multiply your two new numpy arrays, assigning the result to a new variable.

In [20]:
```
new_col = new_variable * last_col
new_col
```

Out[20]:
```
array([  20.,   80.,  180.,  320.,  500.,  720.,  980., 1280., 1620.,
       2000.])
```

Check the shape and type of your new variable.

In [23]:
```
print(new_col.shape)
print(type(new_col))
```

```
(10,)
<class 'numpy.ndarray'>
```

Exploring a larger array

For this part of the exercise, start a new notebook. That is the notebook you'll turn in. At the top of your new notebook, make sure to:

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

Then you can read in our second data file:

In [2]:
```
data_2 = np.load('/Users/paulwen/OneDrive - The University of Texas at Aust
```

Your mission!

Lurking in this numpy array of data is a pattern. It might be in the rows (that is, the data in rows all have something in common), or it might be in the columns. **You job is to find the pattern by plotting rows and columns and summaries of the data set!**

Happily, `plt.plot()` is very friendly. So if you want to plot, say, the fifth row, you just do

```
plt.plot(dataFromFile[4, :])
```

Pro Tip! When you plot, `plt.plot()` prints out a listing of the objects it drew above the plot itself. This gets super annoying if you're plotting a lot of lines, e.g. like this:

```
plt.plot(dataFromFile[0:20, :])
```

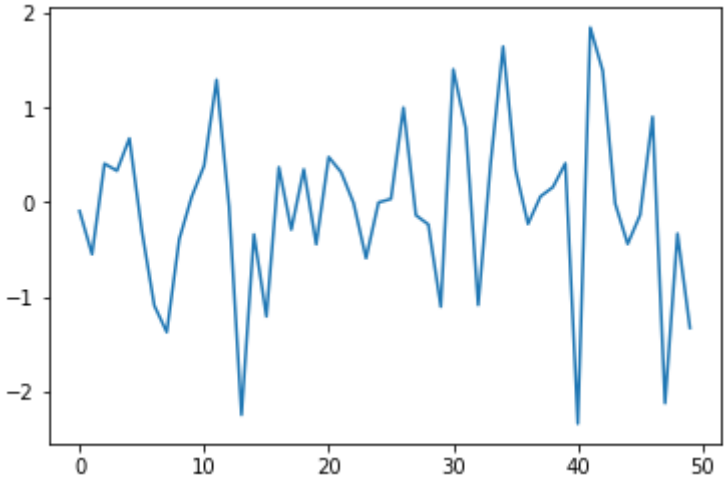
You can suppress this listing by simply adding a semicolon at the end of the line:

```
plt.plot(dataFromFile[4, :]);
```

After you have played around with that for bit, remember that numpy arrays know how to summarize themselves in various ways. As we saw in the tutorial, for example, you can sum down the rows columns!

In [4]:
```
plt.plot(data_2[4, :])
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x7fd1c2b1f8e0>]
```

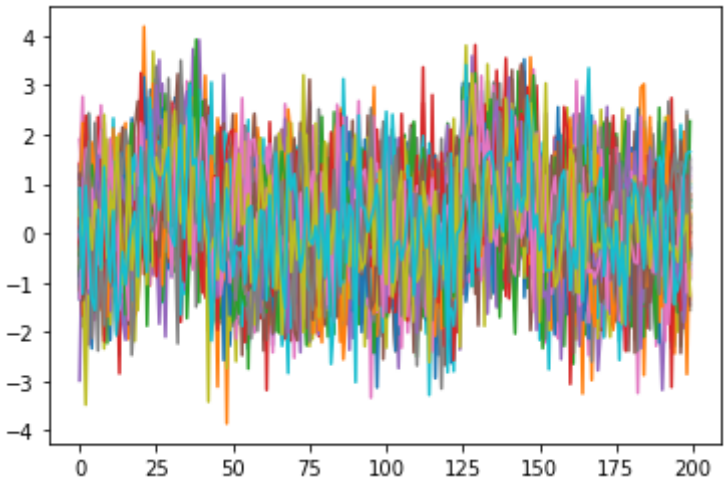


In [6]:
```
data_2.shape
```

Out[6]:
```
(200, 50)
```

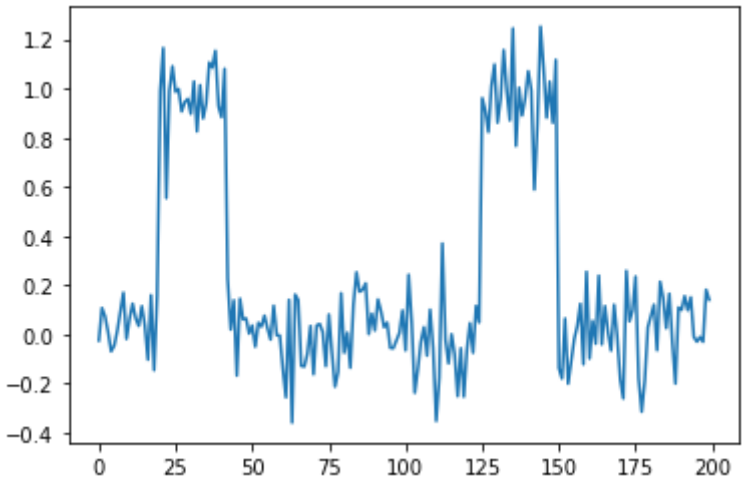
In [7]:
```
plt.plot(data_2[:,0:])
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x7fd1a006cfd0>,
 <matplotlib.lines.Line2D at 0x7fd1a0077070>,
 <matplotlib.lines.Line2D at 0x7fd1a0077190>,
 <matplotlib.lines.Line2D at 0x7fd1a00772b0>,
 <matplotlib.lines.Line2D at 0x7fd1a00773d0>,
 <matplotlib.lines.Line2D at 0x7fd1a00774f0>,
 <matplotlib.lines.Line2D at 0x7fd1a0077610>,
 <matplotlib.lines.Line2D at 0x7fd1a0077730>,
 <matplotlib.lines.Line2D at 0x7fd1a0077850>,
 <matplotlib.lines.Line2D at 0x7fd1a0077970>,
 <matplotlib.lines.Line2D at 0x7fd1a0077a90>,
 <matplotlib.lines.Line2D at 0x7fd1a0077040>,
 <matplotlib.lines.Line2D at 0x7fd1a0077ca0>,
 <matplotlib.lines.Line2D at 0x7fd1a0077dc0>,
 <matplotlib.lines.Line2D at 0x7fd1a0077ee0>,
 <matplotlib.lines.Line2D at 0x7fd1a007a040>,
 <matplotlib.lines.Line2D at 0x7fd1a007a160>,
 <matplotlib.lines.Line2D at 0x7fd1a007a280>,
 <matplotlib.lines.Line2D at 0x7fd1a007a3a0>,
 <matplotlib.lines.Line2D at 0x7fd1a007a4c0>,
 <matplotlib.lines.Line2D at 0x7fd1a007a5e0>,
 <matplotlib.lines.Line2D at 0x7fd1a007a700>,
 <matplotlib.lines.Line2D at 0x7fd1a007a820>,
 <matplotlib.lines.Line2D at 0x7fd1a007a940>,
 <matplotlib.lines.Line2D at 0x7fd1a007aa60>,
 <matplotlib.lines.Line2D at 0x7fd1a007ab80>,
 <matplotlib.lines.Line2D at 0x7fd1a007aca0>,
 <matplotlib.lines.Line2D at 0x7fd1a007adc0>,
 <matplotlib.lines.Line2D at 0x7fd1a007aee0>,
 <matplotlib.lines.Line2D at 0x7fd1a007e040>,
 <matplotlib.lines.Line2D at 0x7fd1a007e160>,
 <matplotlib.lines.Line2D at 0x7fd1a007e280>,
 <matplotlib.lines.Line2D at 0x7fd1a007e3a0>,
 <matplotlib.lines.Line2D at 0x7fd1a007e4c0>,
 <matplotlib.lines.Line2D at 0x7fd1a007e5e0>,
 <matplotlib.lines.Line2D at 0x7fd1a007e700>,
 <matplotlib.lines.Line2D at 0x7fd1a007e820>,
 <matplotlib.lines.Line2D at 0x7fd1a007e940>,
 <matplotlib.lines.Line2D at 0x7fd1a007ea60>,
 <matplotlib.lines.Line2D at 0x7fd1a007eb80>,
 <matplotlib.lines.Line2D at 0x7fd1a007eca0>,
 <matplotlib.lines.Line2D at 0x7fd1a007edc0>,
 <matplotlib.lines.Line2D at 0x7fd1a007eee0>,
 <matplotlib.lines.Line2D at 0x7fd1a0086040>,
 <matplotlib.lines.Line2D at 0x7fd1a0086160>,
 <matplotlib.lines.Line2D at 0x7fd1a0086280>,
 <matplotlib.lines.Line2D at 0x7fd1a00863a0>,
 <matplotlib.lines.Line2D at 0x7fd1a00864c0>,
 <matplotlib.lines.Line2D at 0x7fd1a00865e0>,
 <matplotlib.lines.Line2D at 0x7fd1a0086700>]
```



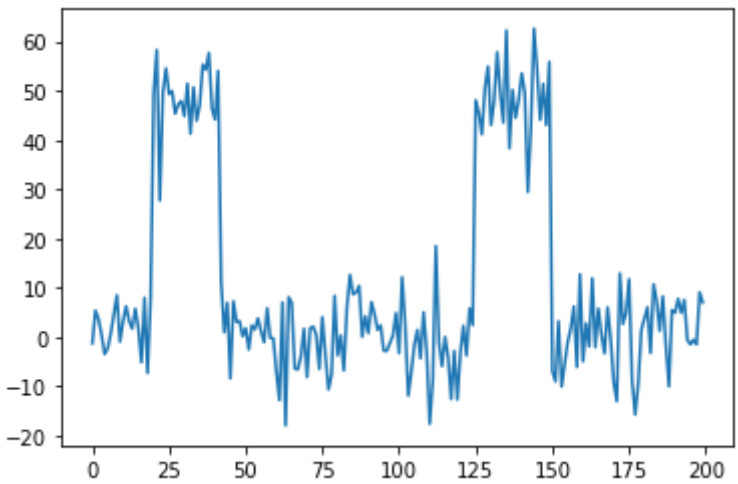
In [10]:
```
plt.plot(data_2.mean(1))
```

Out[10]: [<matplotlib.lines.Line2D at 0x7fd1d15665b0>]



In [11]:
```
plt.plot(data_2.sum(1))
```

Out[11]: [<matplotlib.lines.Line2D at 0x7fd1c330d940>]



In [ ]:

For the assignment, you don't need to show all your figures, just show that you found the pattern!