EASC2410 Lecture 7

Python Basics: 2D Numpy Arrays and Files

Dr. Binzheng Zhang
Department of Earth Sciences

Review of Lecture 6

In Lecture 6, we learned:

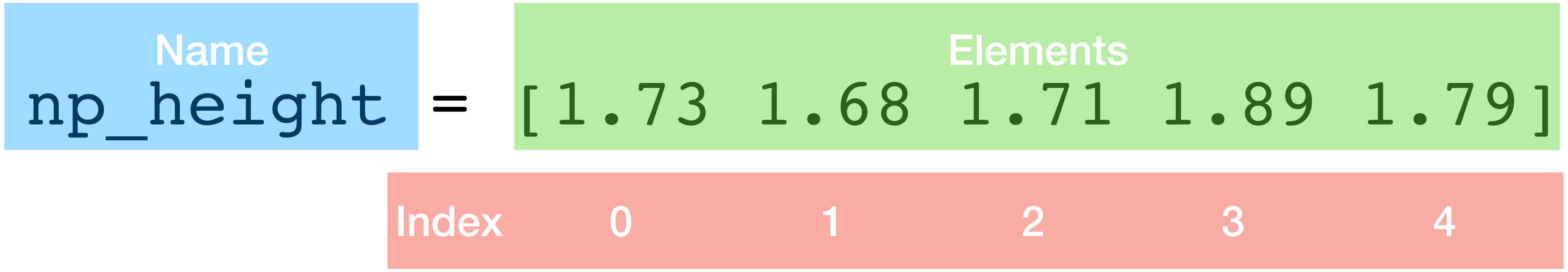
- Visualise 1-D NumPy arrays with Matplotlib.pyplot
- Various 1-D plots using Matplotlib.pyplot
- Style and control on your 1-D data plots
- Beginner's visualisation

In Lecture 7, you will learn:

- 2-D NumPy arrays
- Load data into Python using txt files

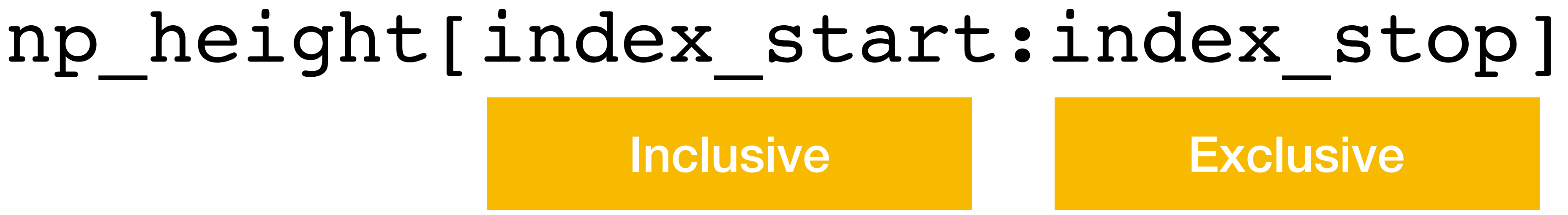
Recall 1-D NumPy Arrays (or Vectors)

1-D Arrays:



1-D NumPy arrays are the basic data type of the NumPy module which contains a series of numerical values with indices starting from zero. Unlike lists, NumPy arrays can have only **one** type of variables

Index Slicing Numpy Arrays:



```
np_height[index_start:index_stop]
```

For example

```
np_height[1:4] =[1.68 1.71 1.89]
```

2-D NumPy Arrays

- We briefly mentioned **arrays** in the last lecture but quickly moved into plotting (because that is more fun).
- But **arrays** are essential to our computational happiness, so we need to bite the bullet and learn about them now.

Recall that **numpy Arrays** are somewhat similar to lists but there are important differences with **advantages** and **disadvantages**. Unlike lists, **arrays** are usually all of the *same data type* (**dtype**), usually numbers (integers or floats) and at times characters. A "feature" of arrays is that the size, shape and type are **fixed** when it's created.

Remember that you can define a list using “[ ]”:

```
mylist = []
```

Then use the append() method to add data to the list:

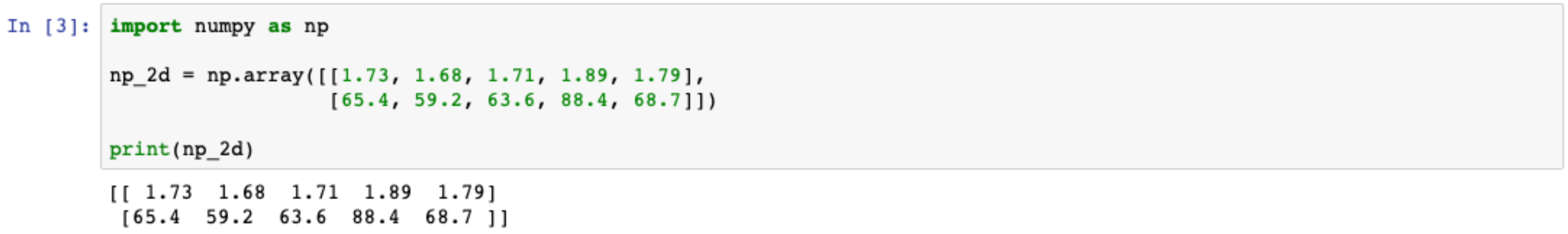
```
mylist.append()
```

But it’s more complicated (but still possible) to **extend** arrays.

Why use arrays when you can use lists? Arrays are far more computationally efficient than lists, particularly for things like matrix math. You can perform calculations on the entire array in one go instead of looping through element by element as for lists.

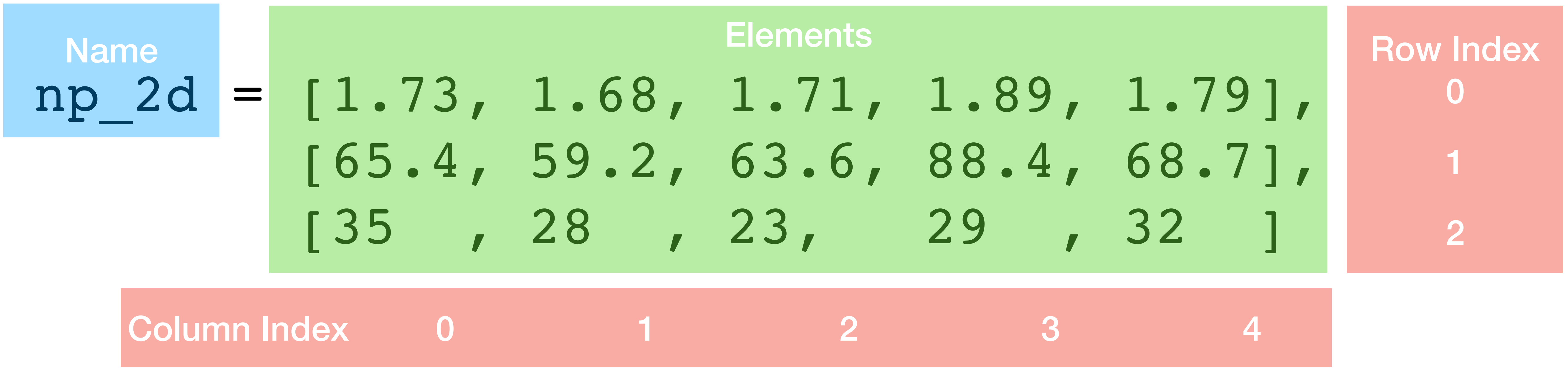
2-D NumPy Arrays

Remember the np.array() function for generating 1-D NumPy arrays, we can use that to generate 2-D arrays as well



```
In [3]: import numpy as np

        np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                          [65.4, 59.2, 63.6, 88.4, 68.7]])

        print(np_2d)

        [[  1.73   1.68   1.71   1.89   1.79]
         [ 65.4   59.2   63.6   88.4   68.7 ]]
```

2-D Arrays:



```
np_2d = [1.73, 1.68, 1.71, 1.89, 1.79],   0
        [65.4, 59.2, 63.6, 88.4, 68.7],   1
        [35  , 28  , 23,   29  , 32  ]    2
          0     1     2     3     4
```

three rows, five columns

2-D NumPy Arrays

Remember that NumPy arrays must be single-typed:

```
In [5]: # try different data types in a 2-D Numpy array:
        np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                          [65.4, 59.2, 63.6, 88.4, 68.7],
                          [35,   28   , 23   , 29   , '32' ]]) # now the last element is a string

        print(np_2d) # print the results - you can see everyone is a string now
```

```
[['1.73' '1.68' '1.71' '1.89' '1.79']
 ['65.4' '59.2' '63.6' '88.4' '68.7']
 ['35' '28' '23' '29' '32']]
```

Now everyone is a string! - what happened in the NumPy array is that whenever Python detects different data types for a NumPy array, it converts everything to strings (if there's string in the array)

More ways to generate 2-D arrays

```
In [11]: D=np.zeros((2,3)) # 2-D numpy array with 2x3 zeros.
                           # Notice the size is specified by a TUPLE of numbers of rows and columns.
         print(D)
```

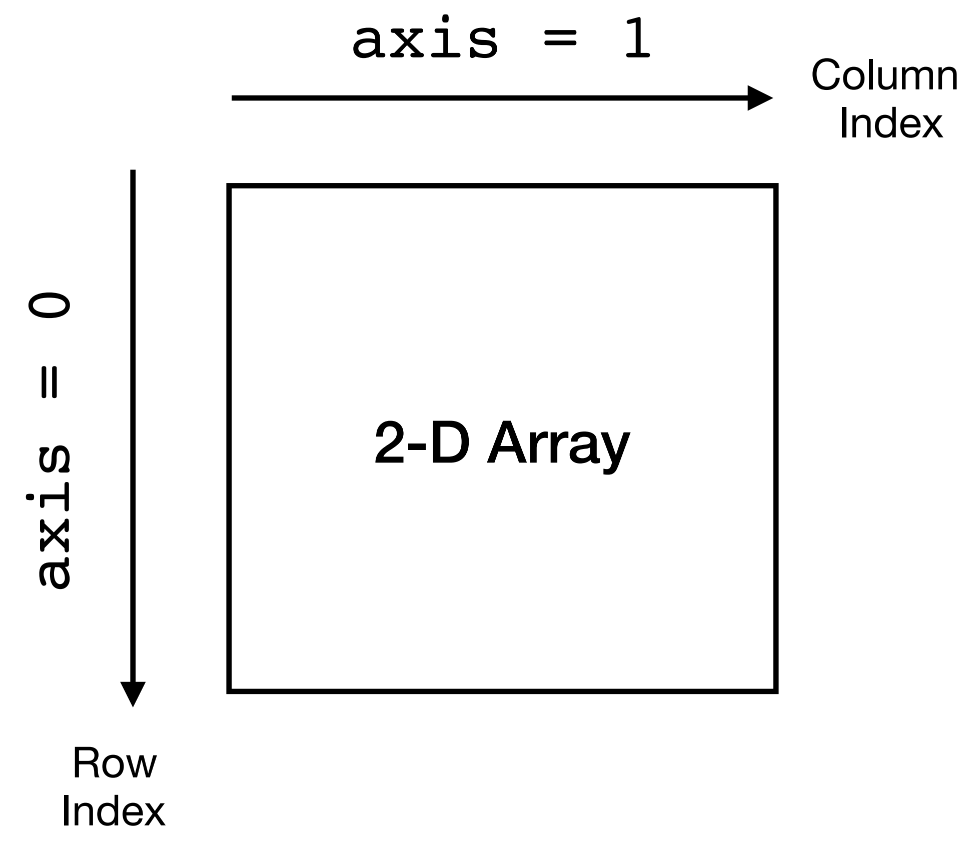
```
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
In [9]: E=np.ones((4,5))
        print(E)
```

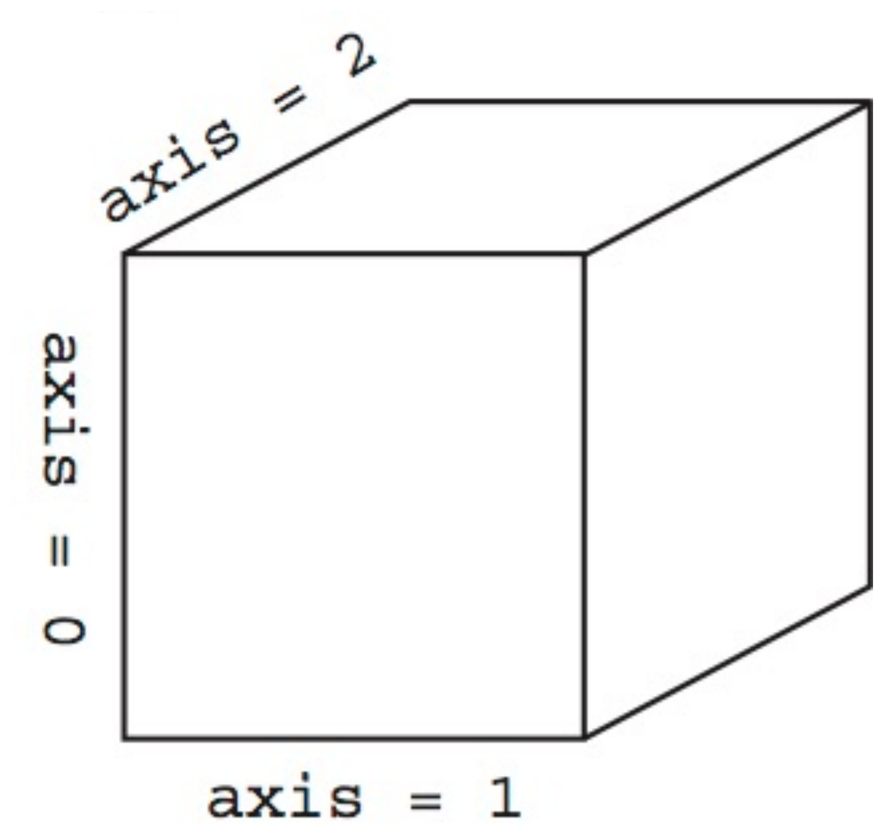
```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

2-D NumPy Arrays - Axis

Axises of arrays:

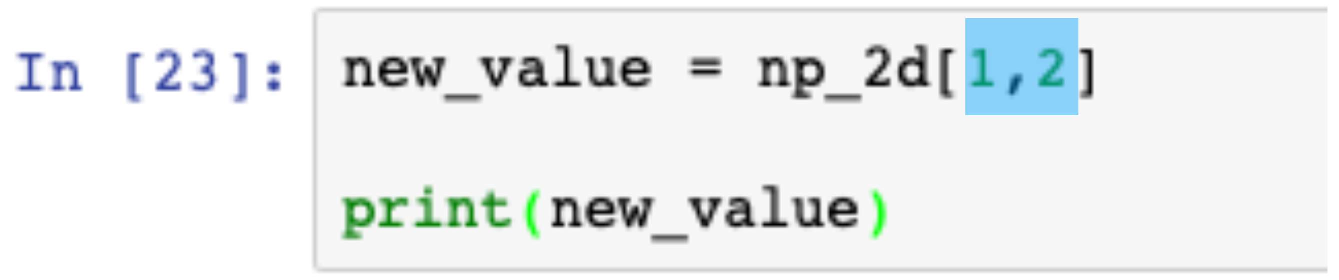


2-D Arrays

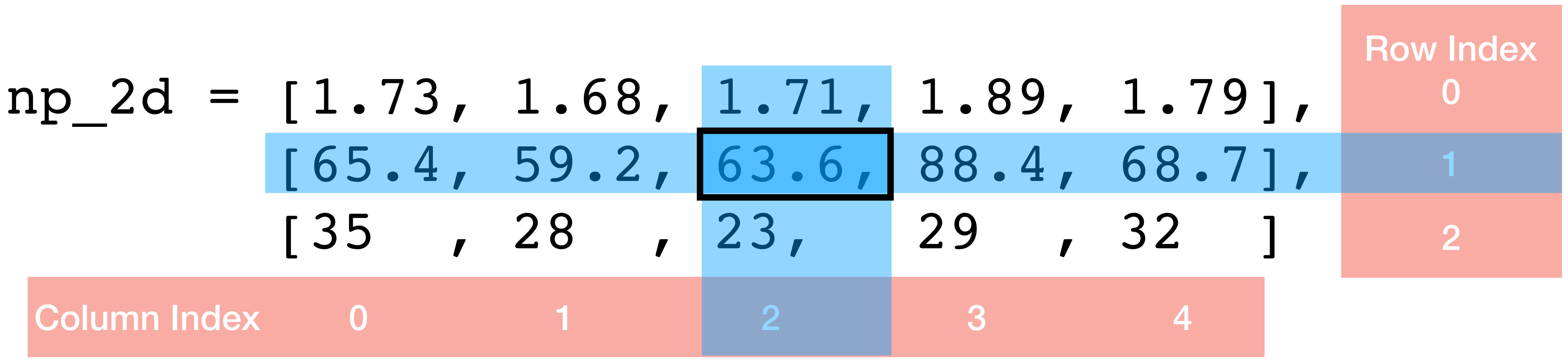


3-D Arrays

2-D NumPy Arrays - Index Slicing

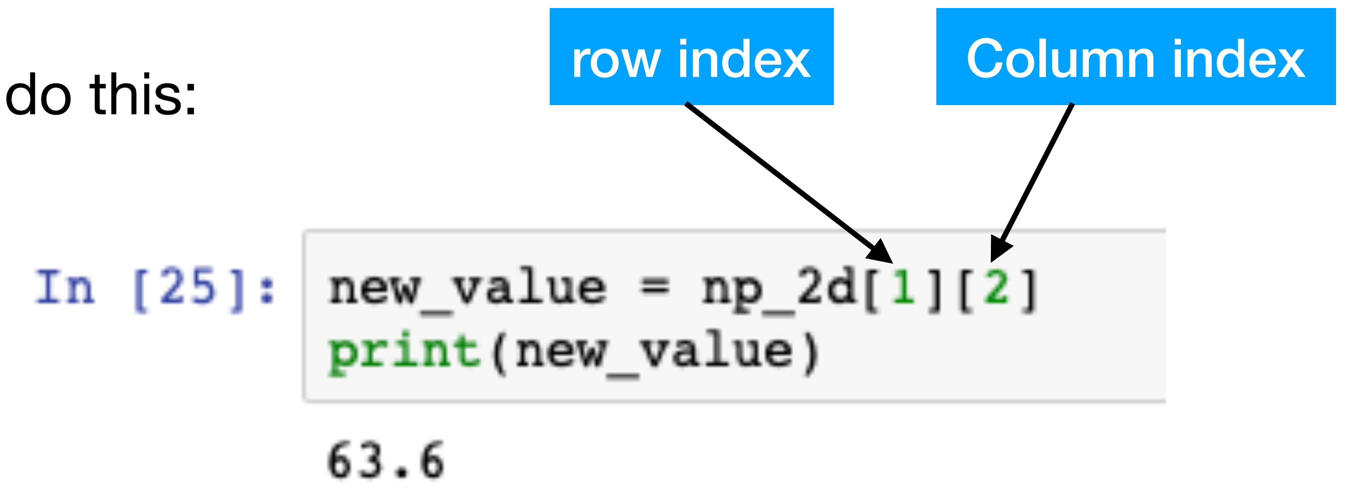


```
In [23]: new_value = np_2d[1,2]

         print(new_value)
```



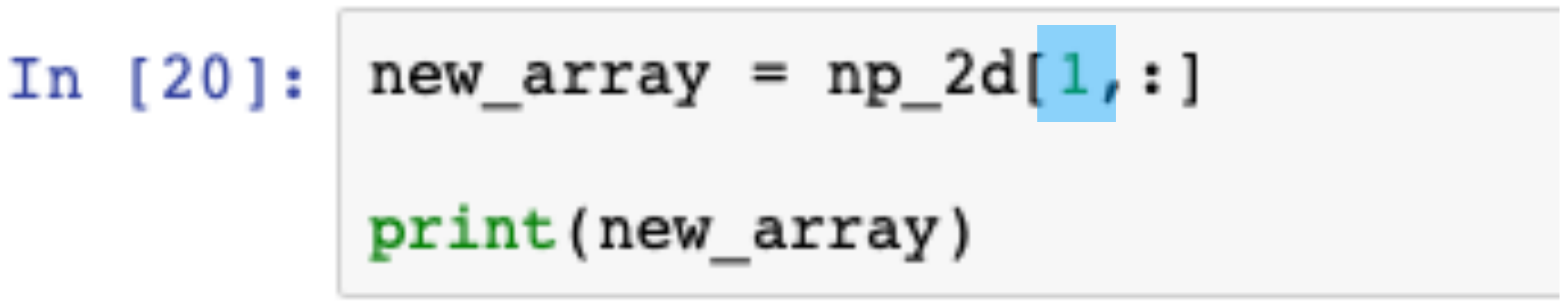
```
np_2d = [1.73, 1.68, 1.71, 1.89, 1.79],
        [65.4, 59.2, 63.6, 88.4, 68.7],
        [35  , 28  , 23,   29  , 32  ]
```

Or you can also do this:



```
In [25]: new_value = np_2d[1][2]
         print(new_value)

         63.6
```

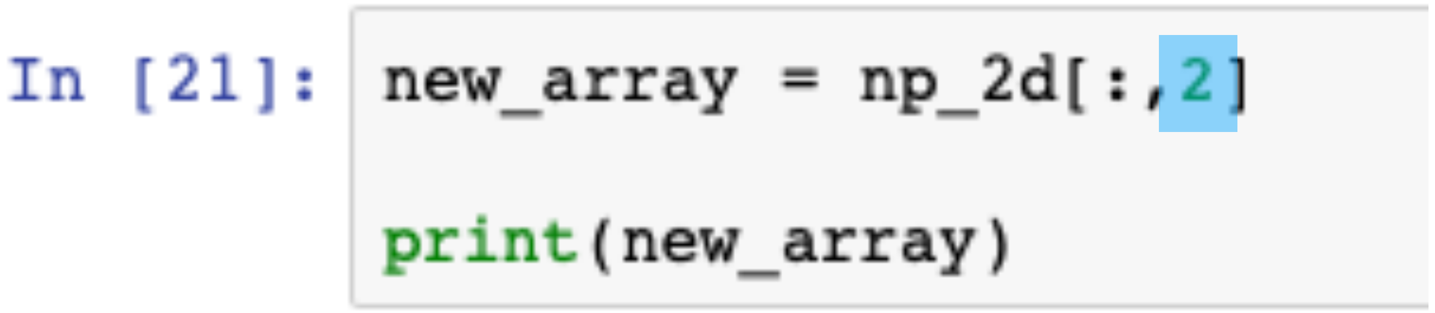
2-D NumPy Arrays - Index Slicing



```
In [20]: new_array = np_2d[1,:]

         print(new_array)
```

```
np_2d = [1.73, 1.68, 1.71, 1.89, 1.79],     Row Index 0
        [65.4, 59.2, 63.6, 88.4, 68.7],               1
        [35  , 28  , 23,   29  , 32  ]                2
Column Index  0     1     2     3     4
```

2-D NumPy Arrays - Index Slicing



```
In [21]: new_array = np_2d[:,2]

         print(new_array)
```

```
np_2d = [1.73, 1.68, 1.71, 1.89, 1.79],
        [65.4, 59.2, 63.6, 88.4, 68.7],
        [35  , 28  , 23,   29  , 32  ]
```

Row Index
0
1
2

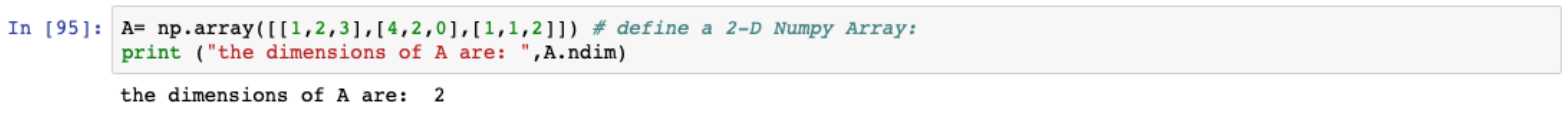
Column Index	0	1	2	3	4

Think: what is `np_2d[2,1:4]`**?**

2-D NumPy Arrays - useful attributes and methods

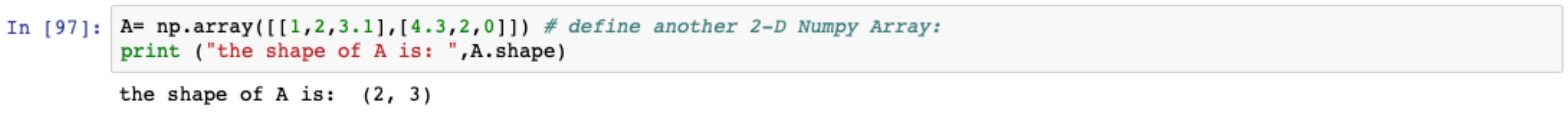
Array Attributes

The ndim: tells you the dimension of your array



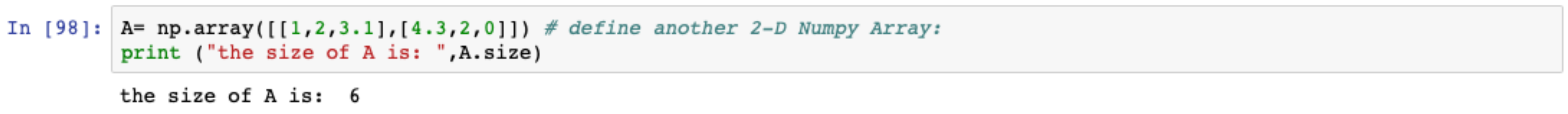
```
In [95]: A= np.array([[1,2,3],[4,2,0],[1,1,2]]) # define a 2-D Numpy Array:
         print ("the dimensions of A are: ",A.ndim)

         the dimensions of A are:  2
```

The shape: tells you how many elements are in each dimension



```
In [97]: A= np.array([[1,2,3.1],[4.3,2,0]]) # define another 2-D Numpy Array:
         print ("the shape of A is: ",A.shape)

         the shape of A is:  (2, 3)
```

The size: tells you how many elements in total



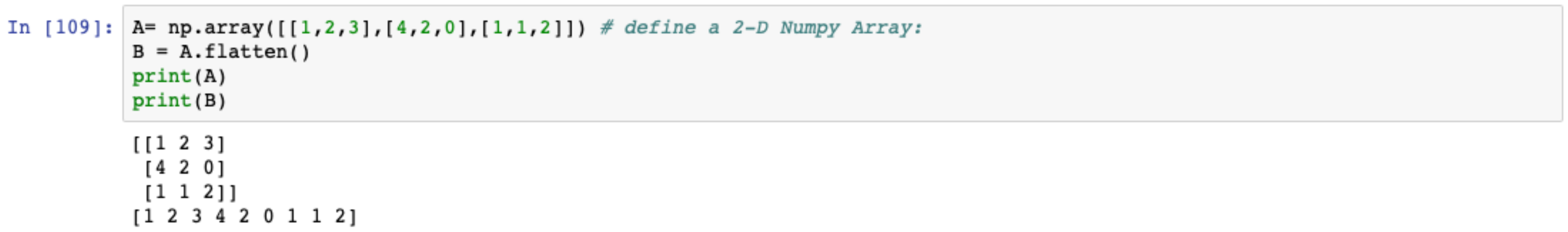
```
In [98]: A= np.array([[1,2,3.1],[4.3,2,0]]) # define another 2-D Numpy Array:
         print ("the size of A is: ",A.size)

         the size of A is:  6
```

Note: no parentheses “( )” are needed when accessing numpy array attributes

2-D NumPy Arrays - useful attributes and methods

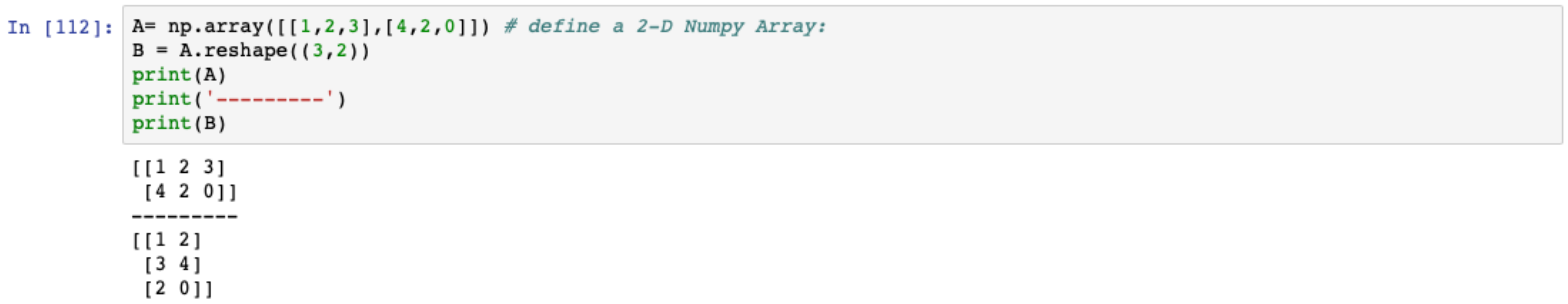
Array Methods

The flatten method: claps multi-dimensional arrays to 1-D arrays



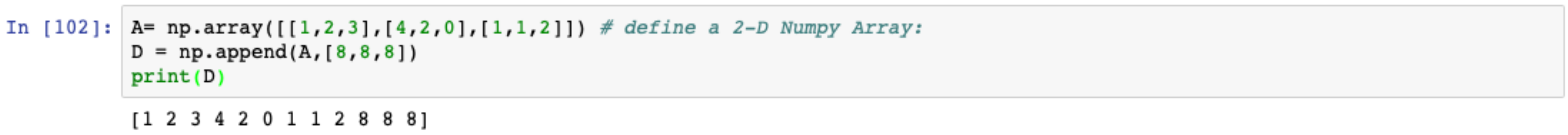
```
In [109]: A= np.array([[1,2,3],[4,2,0],[1,1,2]]) # define a 2-D Numpy Array:
          B = A.flatten()
          print(A)
          print(B)

          [[1 2 3]
           [4 2 0]
           [1 1 2]]
          [1 2 3 4 2 0 1 1 2]
```

The reshape method: changing the shape (row and column) of your array



```
In [112]: A= np.array([[1,2,3],[4,2,0]]) # define a 2-D Numpy Array:
          B = A.reshape((3,2))
          print(A)
          print('---------')
          print(B)

          [[1 2 3]
           [4 2 0]]
          ---------
          [[1 2]
           [3 4]
           [2 0]]
```

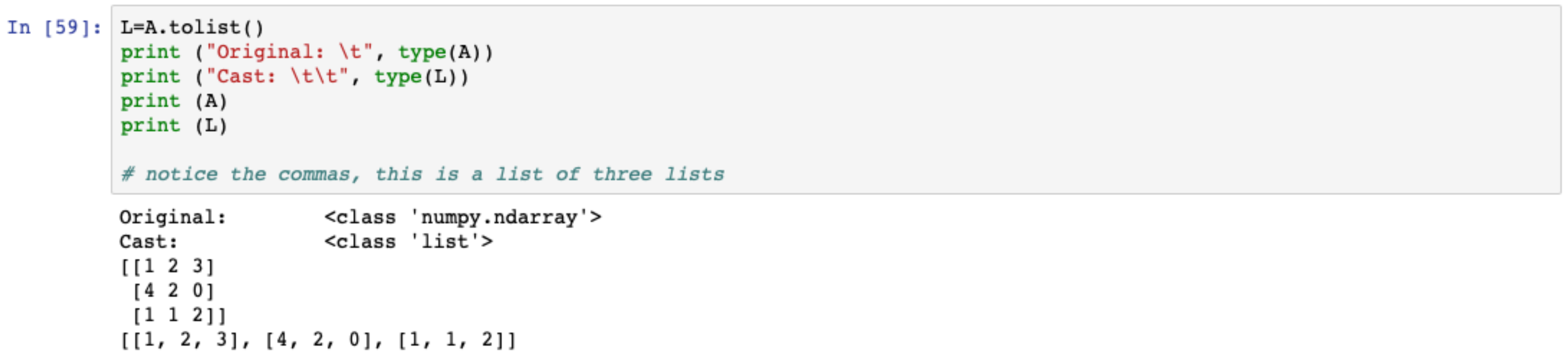
The append method: adding new elements to arrays (in a weird way):



```
In [102]: A= np.array([[1,2,3],[4,2,0],[1,1,2]]) # define a 2-D Numpy Array:
          D = np.append(A,[8,8,8])
          print(D)

          [1 2 3 4 2 0 1 1 2 8 8 8]
```

2-D NumPy Arrays - Converting Arrays to Lists

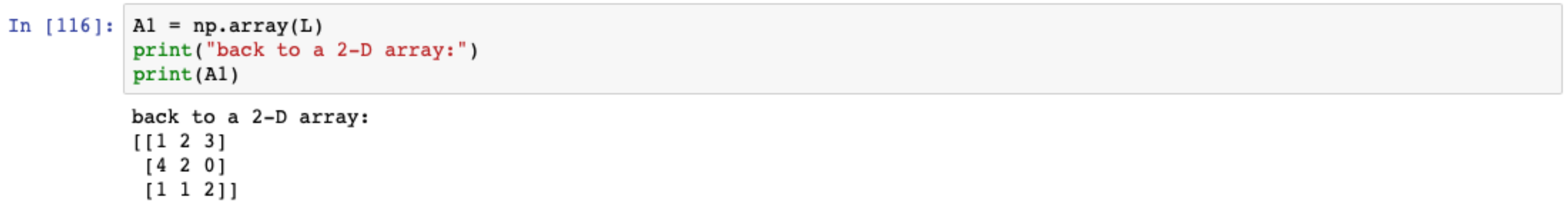
sometimes you may want to convert your numpy arrays to a list, here's how to do it:



```
In [59]: L=A.tolist()
         print ("Original: \t", type(A))
         print ("Cast: \t\t", type(L))
         print (A)
         print (L)

         # notice the commas, this is a list of three lists
```

```
Original:        <class 'numpy.ndarray'>
Cast:            <class 'list'>
[[1 2 3]
 [4 2 0]
 [1 1 2]]
[[1, 2, 3], [4, 2, 0], [1, 1, 2]]
```

Recall we use the np.array() function to convert Python lists to NumPy arrays:

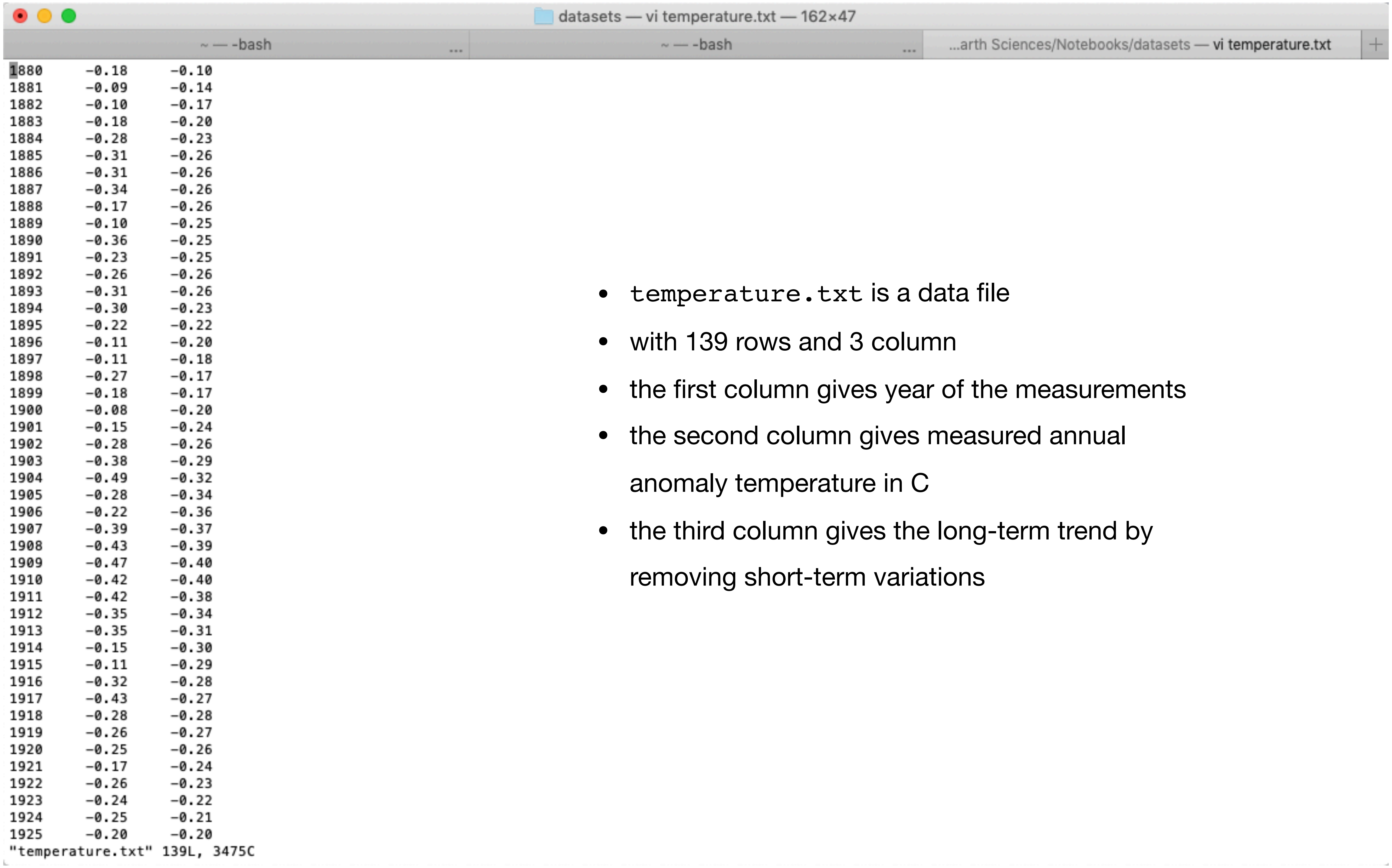


```
In [116]: A1 = np.array(L)
          print("back to a 2-D array:")
          print(A1)
```

```
back to a 2-D array:
[[1 2 3]
 [4 2 0]
 [1 1 2]]
```

Load 2-D NumPy Arrays from files

Most of real-life datasets are stored in computer files such as .txt and .xls files. Numpy provides handy functions to read numeric data from txt files. Here's our data file named **temperature.txt**


datasets — vi temperature.txt — 162×47
~ — -bash
~ — -bash
...arth Sciences/Notebooks/datasets — vi temperature.txt

```
1880     -0.18    -0.10
1881     -0.09    -0.14
1882     -0.10    -0.17
1883     -0.18    -0.20
1884     -0.28    -0.23
1885     -0.31    -0.26
1886     -0.31    -0.26
1887     -0.34    -0.26
1888     -0.17    -0.26
1889     -0.10    -0.25
1890     -0.36    -0.25
1891     -0.23    -0.25
1892     -0.26    -0.26
1893     -0.31    -0.26
1894     -0.30    -0.23
1895     -0.22    -0.22
1896     -0.11    -0.20
1897     -0.11    -0.18
1898     -0.27    -0.17
1899     -0.18    -0.17
1900     -0.08    -0.20
1901     -0.15    -0.24
1902     -0.28    -0.26
1903     -0.38    -0.29
1904     -0.49    -0.32
1905     -0.28    -0.34
1906     -0.22    -0.36
1907     -0.39    -0.37
1908     -0.43    -0.39
1909     -0.47    -0.40
1910     -0.42    -0.40
1911     -0.42    -0.38
1912     -0.35    -0.34
1913     -0.35    -0.31
1914     -0.15    -0.30
1915     -0.11    -0.29
1916     -0.32    -0.28
1917     -0.43    -0.27
1918     -0.28    -0.28
1919     -0.26    -0.27
1920     -0.25    -0.26
1921     -0.17    -0.24
1922     -0.26    -0.23
1923     -0.24    -0.22
1924     -0.25    -0.21
1925     -0.20    -0.20
"temperature.txt" 139L, 3475C
```

- `temperature.txt` is a data file
- with 139 rows and 3 column
- the first column gives year of the measurements
- the second column gives measured annual anomaly temperature in C
- the third column gives the long-term trend by removing short-term variations

Load 2-D NumPy Arrays from files

Most of real-life datasets are stored in computer files such as .txt and .xls files. Numpy provides handy functions to read numeric data from txt files. For example:

In [84]:
```python
import numpy as np
import matplotlib.pyplot as plt

temp_anom = np.loadtxt('datasets/temperature.txt')

print(type(temp_anom))
print(temp_anom)
```

- `temp_anom` is a 2-D NumPy array
- with 139 rows and 3 column
- the first column `temp_anom[:, 0]` gives year
- the second column `temp_anom[:, 1]` gives annual anomaly temperature in C
- the third column `temp_anom[:, 2]` gives the long-term trend

In [94]:
```python
print('The data type is',type(temp_anom))
print('The shape of the array is', temp_anom.shape)
print('The size of the array is', temp_anom.size)
print('The dimension of the array is', temp_anom.ndim)
```

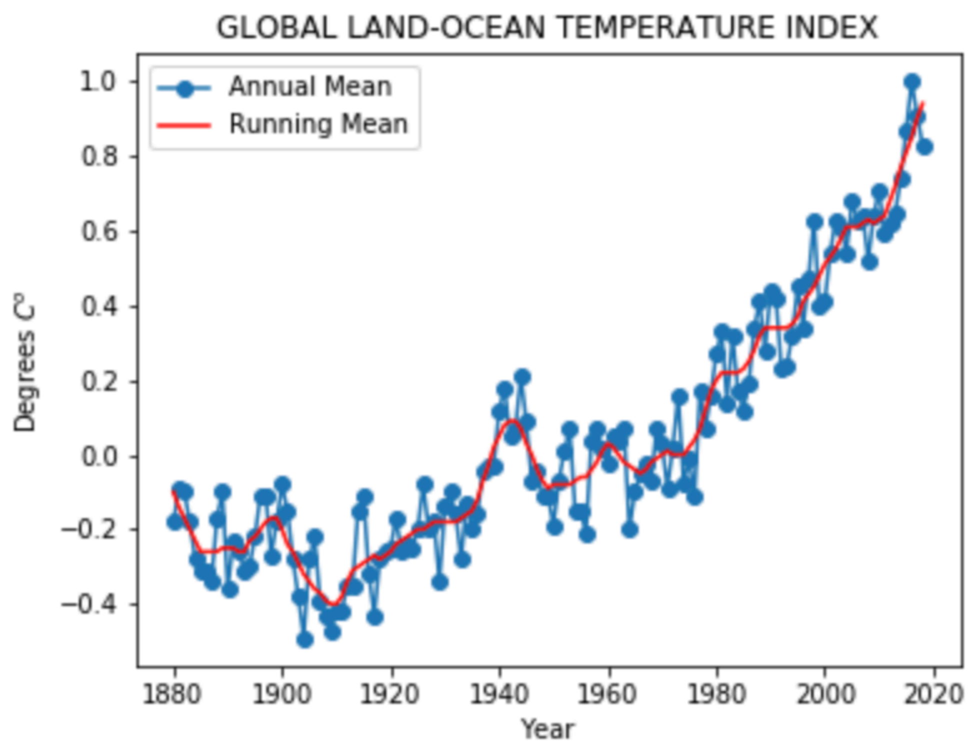
```
The data type is <class 'numpy.ndarray'>
The shape of the array is (139, 3)
The size of the array is 417
The dimension of the array is 2
```

Load 2-D NumPy Arrays from files

Now we can make plots using the data loaded into Python:

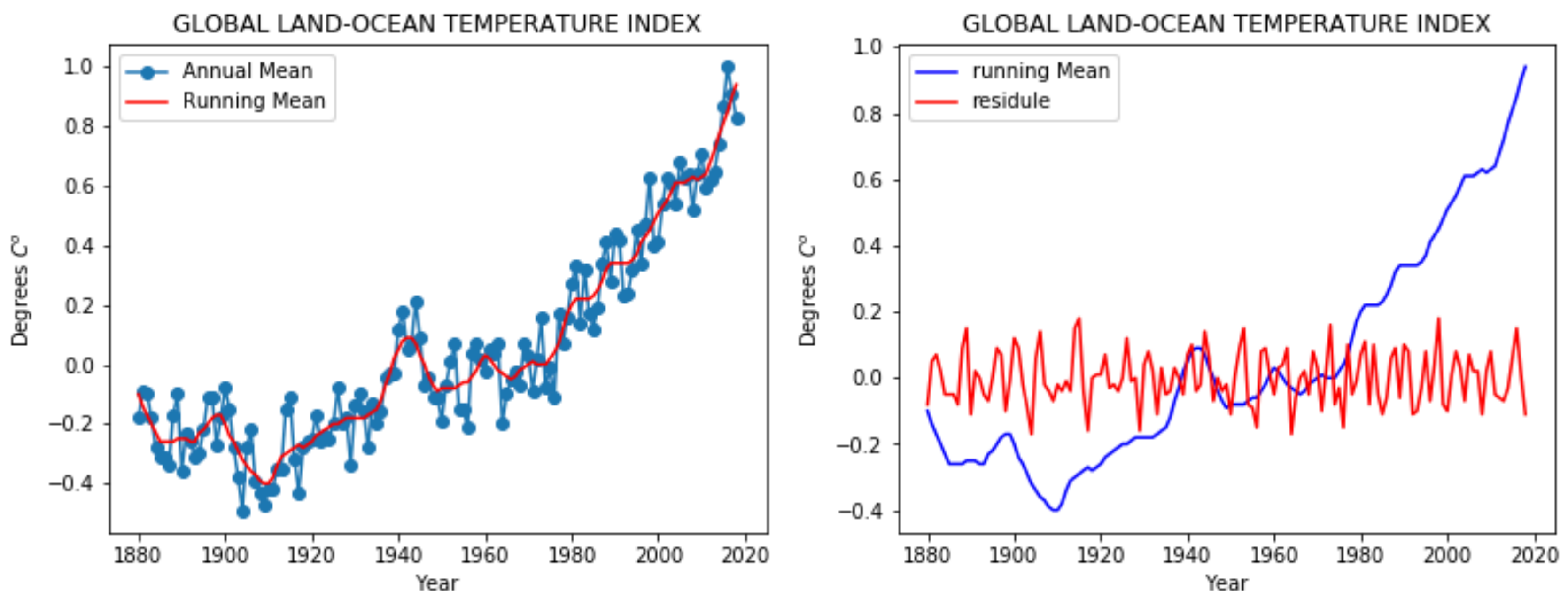
temp_anom is a 2-D NumPy array with 139 rows and 3 column

```
import numpy as np
import matplotlib.pyplot as plt

temp_anom = np.loadtxt('datasets/temperature.txt')

year = temp_anom[:,0]         # column 0: year
anomaly = temp_anom[:,1]      # column 1: annual temperature anomaly
lowess = temp_anom[:,2]       # column 2: the long-term trend of the temperature anomaly
residul = anomaly - lowess    # year-to-year variations

plt.figure(figsize=(13,4.5)) # start a figure, set the size of the figure

plt.subplot(1,2,1) # the left panel
plt.plot(year, anomaly, marker = 'o',label='Annual Mean')
plt.plot(year, lowess, 'r-',label='Running Mean')
plt.xlabel('Year')
plt.ylabel('Degrees $C^o$')
plt.title('GLOBAL LAND-OCEAN TEMPERATURE INDEX')
plt.legend()
```



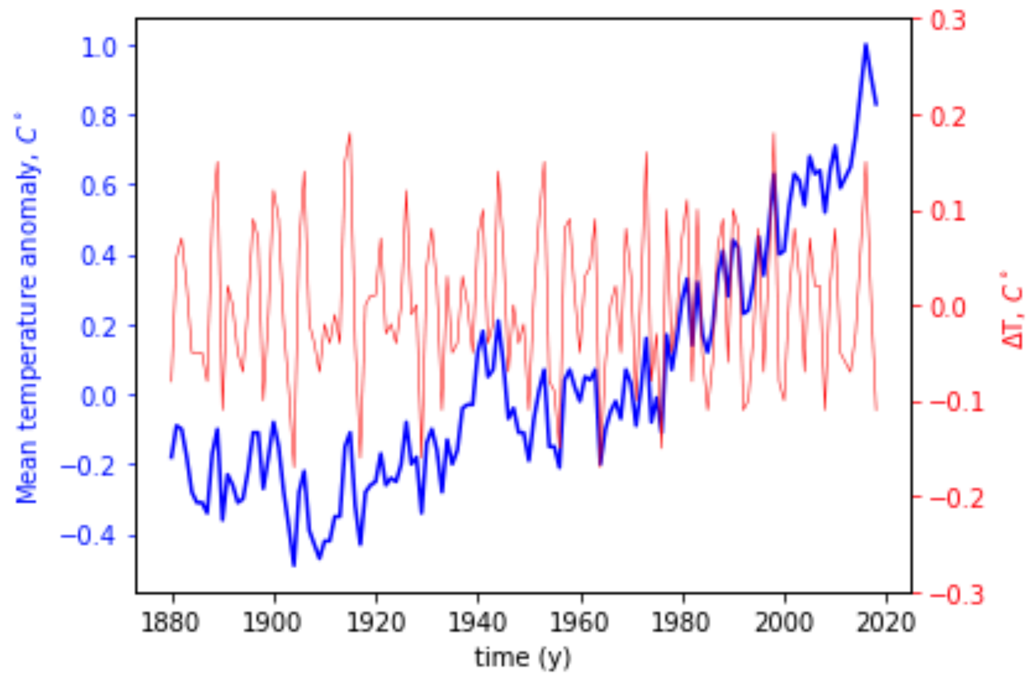
Plot datasets using two panels

```python
import numpy as np
import matplotlib.pyplot as plt

temp_anom = np.loadtxt('datasets/temperature.txt')

year = temp_anom[:,0]        # column 0: year
anomaly = temp_anom[:,1]     # column 1: annual temperature anomaly
lowess = temp_anom[:,2]      # column 2: the long-term trend of the temperature anomaly
residul = anomaly - lowess   # year-to-year variations

plt.figure(figsize=(13,4.5)) # start a figure, set the size of the figure

plt.subplot(1,2,1) # the left panel
plt.plot(year, anomaly, marker = 'o',label='Annual Mean')
plt.plot(year, lowess, 'r-',label='Running Mean')
plt.xlabel('Year')
plt.ylabel('Degrees $C^o$')
plt.title('GLOBAL LAND-OCEAN TEMPERATURE INDEX')
plt.legend()

plt.subplot(1,2,2) # the right panel
plt.plot(year, lowess,'b-',label='running Mean')
plt.plot(year, residul, 'r-',label='residule')
plt.xlabel('Year')
plt.ylabel('Degrees $C^o$')
plt.title('GLOBAL LAND-OCEAN TEMPERATURE INDEX')
plt.legend()

plt.show() # show your plot!
```



Plot datasets with different scales

```
In [132]: fig, ax1 = plt.subplots()
          #t = np.arange(0.01, 10.0, 0.01)
          #s1 = np.exp(t)
          ax1.plot(year, results, 'b-')
          ax1.set_xlabel('time (y)')
          # Make the y-axis label, ticks and tick labels match the line color.
          ax1.set_ylabel('Mean temperature anomaly, $C^\circ$', color='b')
          ax1.tick_params('y', colors='b')

          ax2 = ax1.twinx()
          ax2.plot(year, results-lowess, 'r-',linewidth=0.5)
          ax2.set_ylabel('$\Delta$T, $C^\circ$', color='r')
          ax2.tick_params('y', colors='r')
          ax2.set_ylim([-0.3,0.3])

          fig.tight_layout()
          plt.show()
```



Demonstrate how to do two plots on the same axes with different left and right scales.

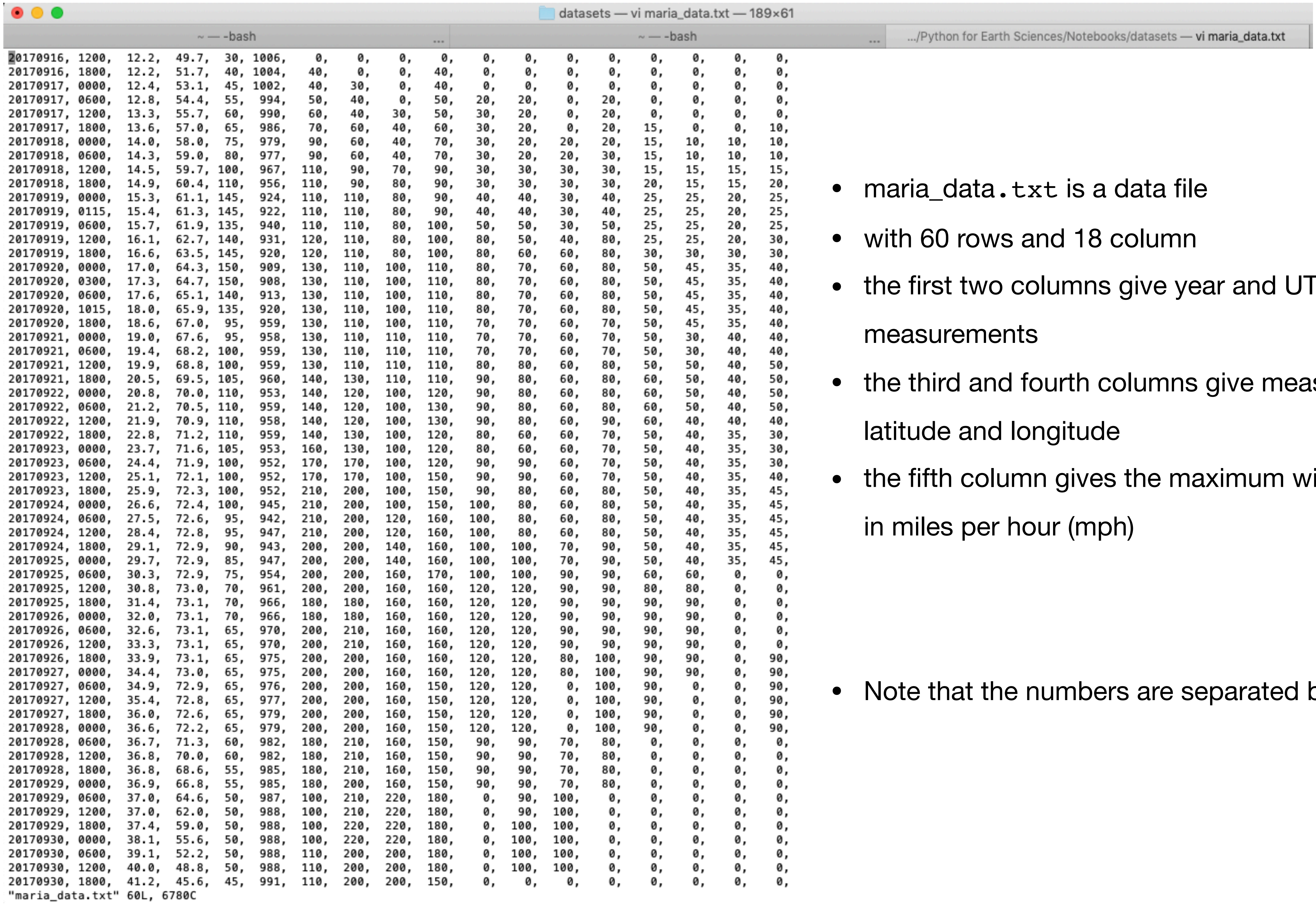
The trick is to use *two different axes* that share the same *x* axis. You can use separate `matplotlib.ticker` formatters and locators as desired since the two axes are independent.

Such axes are generated by calling the `Axes.twinx` method. Likewise, `Axes.twiny` is available to generate axes that share a *y* axis but have different top and bottom scales.

The twinx and twiny methods are also exposed as pyplot functions.

Load 2-D NumPy Arrays from files

There’s another handy numpy function for loading data from txt files, genfromtxt(). For example we have the data file for the Hurricane Maria with track data. The data file looks like this:


datasets — vi maria_data.txt — 189×61
~ — -bash
~ — -bash
.../Python for Earth Sciences/Notebooks/datasets — vi maria_data.txt

```
20170916, 1200, 12.2, 49.7,  30, 1006,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
20170916, 1800, 12.2, 51.7,  40, 1004,   40,    0,    0,   40,    0,    0,    0,    0,    0,    0,    0,    0,
20170917, 0000, 12.4, 53.1,  45, 1002,   40,   30,    0,   40,    0,    0,    0,    0,    0,    0,    0,    0,
20170917, 0600, 12.8, 54.4,  55,  994,   50,   40,    0,   50,   20,   20,    0,   20,    0,    0,    0,    0,
20170917, 1200, 13.3, 55.7,  60,  990,   60,   40,   30,   50,   30,   20,    0,   20,    0,    0,    0,    0,
20170917, 1800, 13.6, 57.0,  65,  986,   70,   60,   40,   60,   30,   20,    0,   20,   15,    0,    0,   10,
20170918, 0000, 14.0, 58.0,  75,  979,   90,   60,   40,   70,   30,   20,   20,   20,   15,   10,   10,   10,
20170918, 0600, 14.3, 59.0,  80,  977,   90,   60,   40,   70,   30,   20,   20,   30,   15,   10,   10,   10,
20170918, 1200, 14.5, 59.7, 100,  967,  110,   90,   70,   90,   30,   30,   30,   30,   15,   15,   15,   15,
20170918, 1800, 14.9, 60.4, 110,  956,  110,   90,   80,   90,   30,   30,   30,   30,   20,   15,   15,   20,
20170919, 0000, 15.3, 61.1, 145,  924,  110,  110,   80,   90,   40,   40,   30,   40,   25,   25,   20,   25,
20170919, 0115, 15.4, 61.3, 145,  922,  110,  110,   80,   90,   40,   40,   30,   40,   25,   25,   20,   25,
20170919, 0600, 15.7, 61.9, 135,  940,  110,  110,   80,  100,   50,   50,   30,   50,   25,   25,   20,   25,
20170919, 1200, 16.1, 62.7, 140,  931,  120,  110,   80,  100,   80,   50,   40,   80,   25,   25,   20,   30,
20170919, 1800, 16.6, 63.5, 145,  920,  120,  110,   80,  100,   80,   60,   60,   80,   30,   30,   30,   30,
20170920, 0000, 17.0, 64.3, 150,  909,  130,  110,  100,  110,   80,   70,   60,   80,   50,   45,   35,   40,
20170920, 0300, 17.3, 64.7, 150,  908,  130,  110,  100,  110,   80,   70,   60,   80,   50,   45,   35,   40,
20170920, 0600, 17.6, 65.1, 140,  913,  130,  110,  100,  110,   80,   70,   60,   80,   50,   45,   35,   40,
20170920, 1015, 18.0, 65.9, 135,  920,  130,  110,  100,  110,   80,   70,   60,   80,   50,   45,   35,   40,
20170920, 1800, 18.6, 67.0,  95,  959,  130,  110,  100,  110,   70,   70,   60,   70,   50,   45,   35,   40,
20170921, 0000, 19.0, 67.6,  95,  958,  130,  110,  110,  110,   70,   70,   60,   70,   50,   30,   40,   40,
20170921, 0600, 19.4, 68.2, 100,  959,  130,  110,  110,  110,   70,   70,   60,   70,   50,   30,   40,   40,
20170921, 1200, 19.9, 68.8, 100,  959,  130,  110,  110,  110,   80,   80,   60,   80,   50,   50,   40,   50,
20170921, 1800, 20.5, 69.5, 105,  960,  140,  130,  110,  110,   90,   80,   60,   80,   60,   50,   40,   50,
20170922, 0000, 20.8, 70.0, 110,  953,  140,  120,  100,  120,   90,   80,   60,   80,   60,   50,   40,   50,
20170922, 0600, 21.2, 70.5, 110,  959,  140,  120,  100,  130,   90,   80,   60,   80,   60,   50,   40,   50,
20170922, 1200, 21.9, 70.9, 110,  958,  140,  120,  100,  130,   90,   80,   60,   90,   60,   40,   40,   40,
20170922, 1800, 22.8, 71.2, 110,  959,  140,  130,  100,  120,   80,   60,   60,   70,   50,   40,   35,   30,
20170923, 0000, 23.7, 71.6, 105,  953,  160,  130,  100,  120,   80,   60,   60,   70,   50,   40,   35,   30,
20170923, 0600, 24.4, 71.9, 100,  952,  170,  170,  100,  120,   90,   90,   60,   70,   50,   40,   35,   30,
20170923, 1200, 25.1, 72.1, 100,  952,  170,  170,  100,  150,   90,   90,   60,   70,   50,   40,   35,   40,
20170923, 1800, 25.9, 72.3, 100,  952,  210,  200,  100,  150,   90,   80,   60,   80,   50,   40,   35,   45,
20170924, 0000, 26.6, 72.4, 100,  945,  210,  200,  100,  150,  100,   80,   60,   80,   50,   40,   35,   45,
20170924, 0600, 27.5, 72.6,  95,  942,  210,  200,  120,  160,  100,   80,   60,   80,   50,   40,   35,   45,
20170924, 1200, 28.4, 72.8,  95,  947,  210,  200,  120,  160,  100,   80,   60,   80,   50,   40,   35,   45,
20170924, 1800, 29.1, 72.9,  90,  943,  200,  200,  140,  160,  100,  100,   70,   90,   50,   40,   35,   45,
20170925, 0000, 29.7, 72.9,  85,  947,  200,  200,  140,  160,  100,  100,   70,   90,   50,   40,   35,   45,
20170925, 0600, 30.3, 72.9,  75,  954,  200,  200,  160,  170,  100,  100,   90,   90,   60,   60,    0,    0,
20170925, 1200, 30.8, 73.0,  70,  961,  200,  200,  160,  160,  120,  120,   90,   90,   80,   80,    0,    0,
20170925, 1800, 31.4, 73.1,  70,  966,  180,  180,  160,  160,  120,  120,   90,   90,   90,   90,    0,    0,
20170926, 0000, 32.0, 73.1,  70,  966,  180,  180,  160,  160,  120,  120,   90,   90,   90,   90,    0,    0,
20170926, 0600, 32.6, 73.1,  65,  970,  200,  210,  160,  160,  120,  120,   90,   90,   90,   90,    0,    0,
20170926, 1200, 33.3, 73.1,  65,  970,  200,  210,  160,  160,  120,  120,   90,   90,   90,   90,    0,    0,
20170926, 1800, 33.9, 73.1,  65,  975,  200,  200,  160,  160,  120,  120,   80,  100,   90,   90,    0,   90,
20170927, 0000, 34.4, 73.0,  65,  975,  200,  200,  160,  160,  120,  120,   80,  100,   90,   90,    0,   90,
20170927, 0600, 34.9, 72.9,  65,  976,  200,  200,  160,  150,  120,  120,    0,  100,   90,    0,    0,   90,
20170927, 1200, 35.4, 72.8,  65,  977,  200,  200,  160,  150,  120,  120,    0,  100,   90,    0,    0,   90,
20170927, 1800, 36.0, 72.6,  65,  979,  200,  200,  160,  150,  120,  120,    0,  100,   90,    0,    0,   90,
20170928, 0000, 36.6, 72.2,  65,  979,  200,  200,  160,  150,  120,  120,    0,  100,   90,    0,    0,   90,
20170928, 0600, 36.7, 71.3,  60,  982,  180,  210,  160,  150,   90,   90,   70,   80,    0,    0,    0,    0,
20170928, 1200, 36.8, 70.0,  60,  982,  180,  210,  160,  150,   90,   90,   70,   80,    0,    0,    0,    0,
20170928, 1800, 36.8, 68.6,  55,  985,  180,  210,  160,  150,   90,   90,   70,   80,    0,    0,    0,    0,
20170929, 0000, 36.9, 66.8,  55,  985,  180,  200,  160,  150,   90,   90,   70,   80,    0,    0,    0,    0,
20170929, 0600, 37.0, 64.6,  50,  987,  100,  210,  220,  180,    0,   90,  100,    0,    0,    0,    0,    0,
20170929, 1200, 37.0, 62.0,  50,  988,  100,  210,  220,  180,    0,   90,  100,    0,    0,    0,    0,    0,
20170929, 1800, 37.4, 59.0,  50,  988,  100,  220,  220,  180,    0,  100,  100,    0,    0,    0,    0,    0,
20170930, 0000, 38.1, 55.6,  50,  988,  100,  220,  220,  180,    0,  100,  100,    0,    0,    0,    0,    0,
20170930, 0600, 39.1, 52.2,  50,  988,  110,  200,  200,  180,    0,  100,  100,    0,    0,    0,    0,    0,
20170930, 1200, 40.0, 48.8,  50,  988,  110,  200,  200,  180,    0,  100,  100,    0,    0,    0,    0,    0,
20170930, 1800, 41.2, 45.6,  45,  991,  110,  200,  200,  150,    0,    0,    0,    0,    0,    0,    0,    0,
"maria_data.txt" 60L, 6780C
```

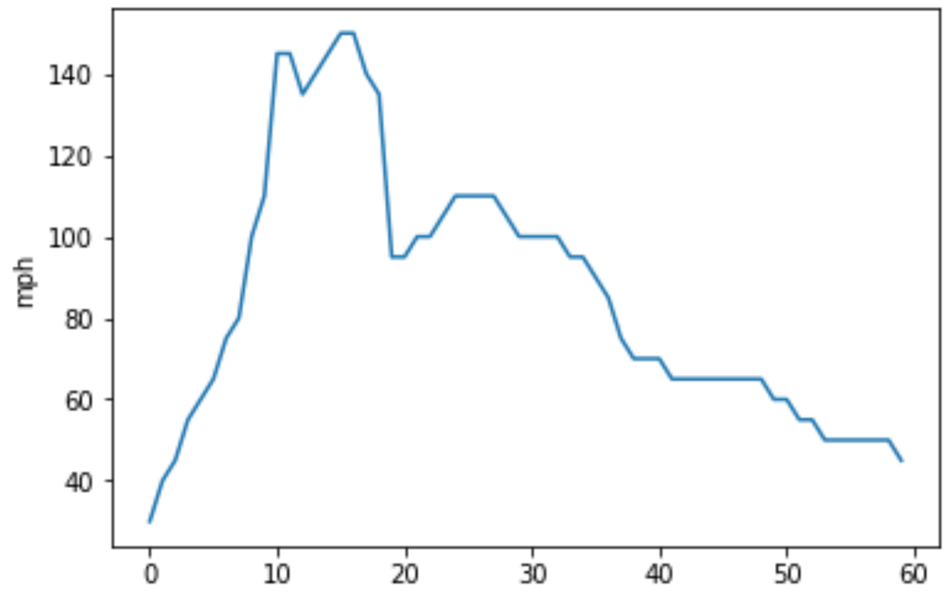
- maria_data.txt is a data file
- with 60 rows and 18 column
- the first two columns give year and UT of the measurements
- the third and fourth columns give measured latitude and longitude
- the fifth column gives the maximum wind speed in miles per hour (mph)

- Note that the numbers are separated by “,”

Load 2-D NumPy Arrays from files

There’s another handy numpy function for loading data from txt files, genfromtxt(). For example we have the data file for the Hurricane Maria with track data. The data file looks like this:

In [123]:
```
# load the data file with delimiter ','
maria = np.genfromtxt('datasets/maria_data.txt', delimiter=',')

# extract the fifth column, which is the wind speed
speed = maria[:,4]

# make a quick plot
plt.plot(speed)
plt.ylabel('mph')
plt.show()
```



- maria_data`.txt` is a data file
- with 60 rows and 18 column
- the first two columns give year and UT of the measurements
- the third and fourth columns give measured latitude and longitude
- the fifth column gives the maximum wind speed in miles per hour (mph)

- Note that the numbers are separated by “,”

Now let’s make some plots!