Assignment_1

April 12, 2018

1 Team Members

1. Mohammad Wasil
2. Ravikiran Bhat

2 Matplotlib

Documentation: http://matplotlib.org/

Matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

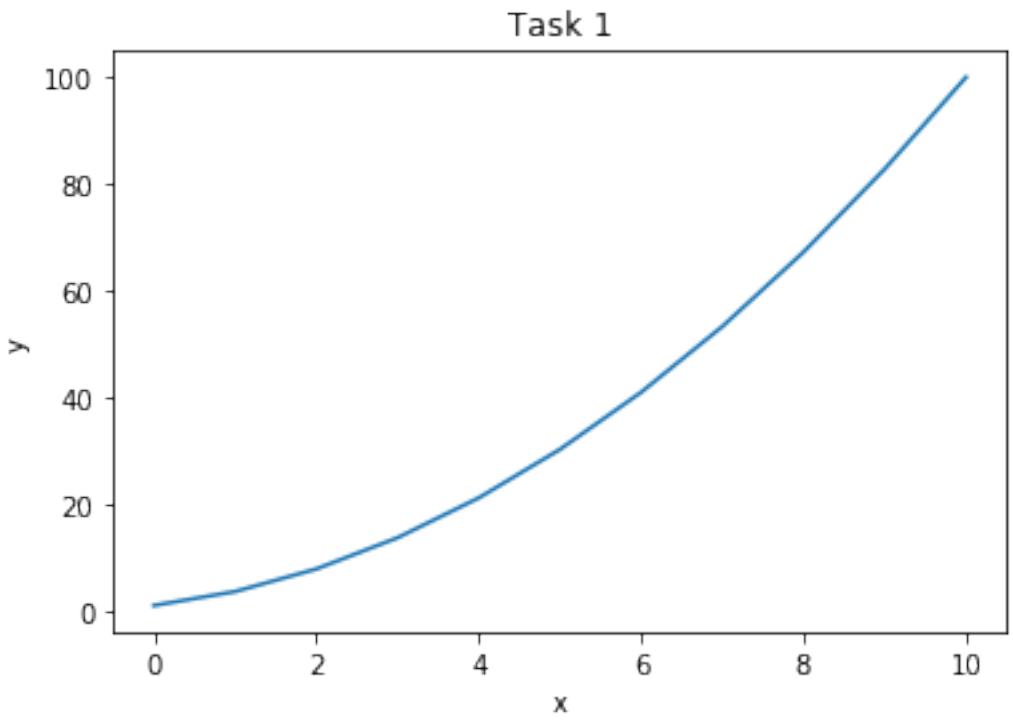
You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc.

```
In [1]: # needed to display the graphs
        %matplotlib inline
        import pylab
        import numpy as np
        import matplotlib.pyplot as plt
```

2.1 Task 1

- Create a plot $y = x^2$ for $x \in [1 : 10]$
- Add Title and Axes (Replicate the plot below)

```
In [2]: x = np.linspace(1,10,11)
        y = x**2
        plt.plot(y)
        plt.title("Task 1")
        plt.xlabel("x")
        plt.ylabel("y")

Out[2]: <matplotlib.text.Text at 0x7fc5dc829f50>
```



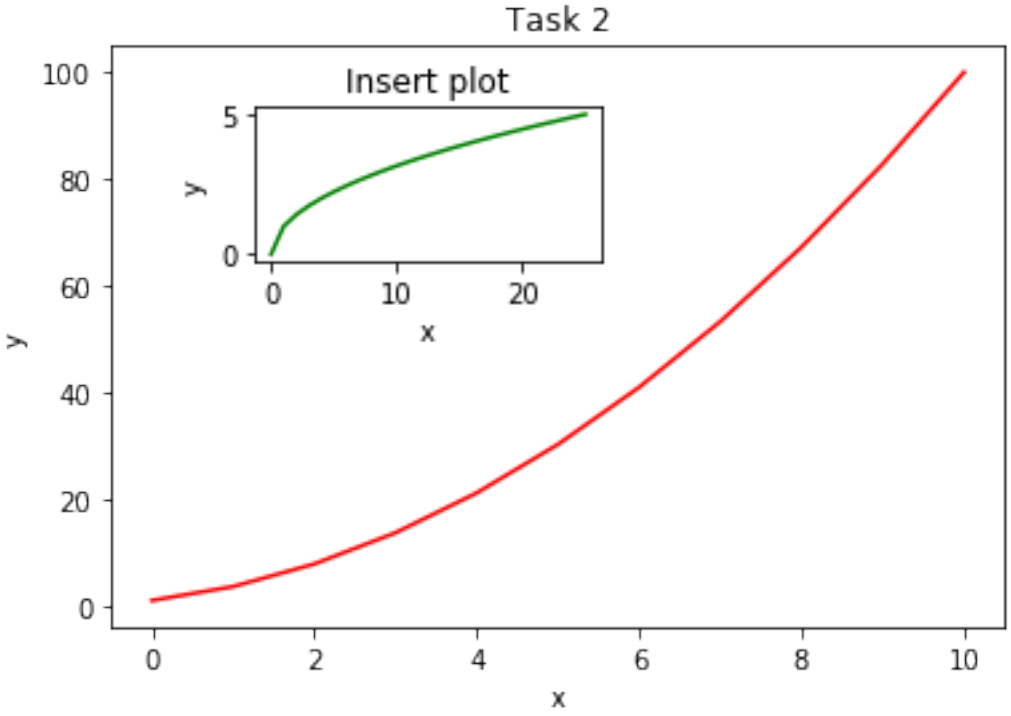
In [ ]:

2.2 Task 2

Create two plots: 'main' and 'insert' and place them such that - The 'insert' plot are included into the 'main' plot - The 'insert' is next to the 'main' plot (Replicate the plots below)

In [3]:
```
#Main plot

x = np.linspace(1,10,11)
y = x**2
#Insert plot
x_2 = np.arange(0,26)
y_2 = np.sqrt(x_2)

fig, ax1 = plt.subplots()
ax1.set_title('Task 2')
ax1.set_xlabel("x")
ax1.set_ylabel("y")

left, bottom, width, height = [0.25, 0.6, 0.3, 0.2]
ax2 = fig.add_axes([left, bottom, width, height])
ax2.set_title('Insert plot')
ax2.set_xlabel("x")
```

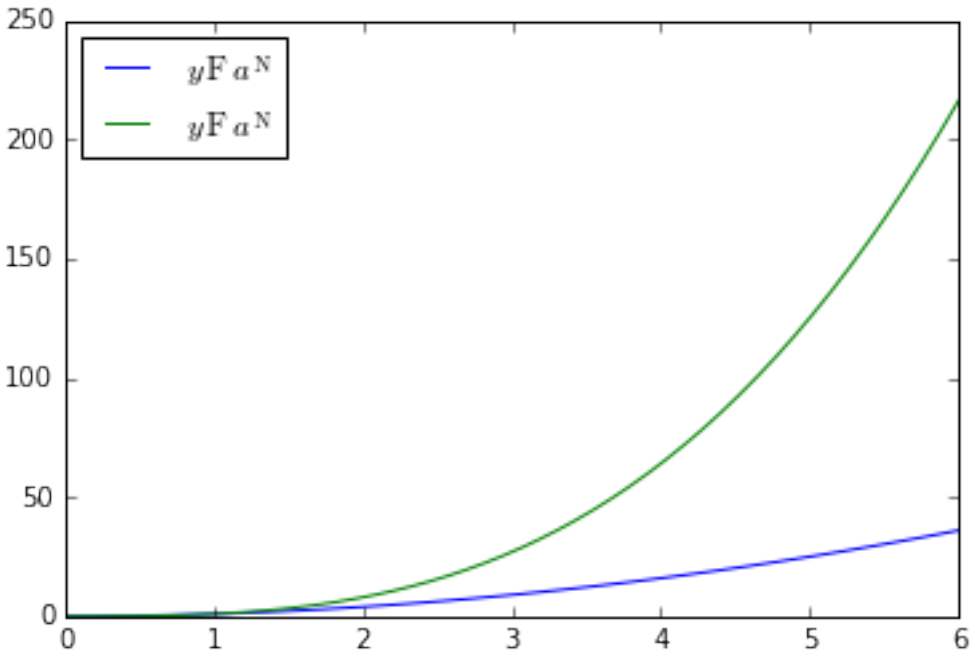
```
ax2.set_ylabel("y")

ax1.plot(y, color='red')
ax2.plot(y_2, color='green')
plt.show()
```



2.3 Task 3

Create a plot with a legend and latex symbols

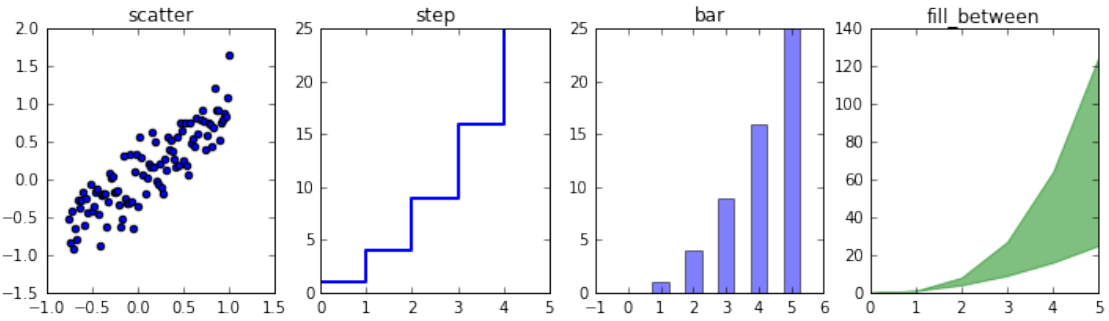
```
In [148]: a = np.linspace(0,6)
          y_1 = a**2
          y_2 = a**3
          plt.plot(a, y_1, color='blue', label="$y=a^{2}$")
          plt.plot(a, y_2, color='green', label='$y=a^2$')
          plt.legend(loc=2)

Out[148]: <matplotlib.legend.Legend at 0x7f00c6009850>
```



2.4 Task 4

Other plot styles. Given:

```
In [171]: xx = np.linspace(-0.75, 1., 100)
          n = array([0,1,2,3,4,5])

          fig, ax = plt.subplots(1, 4, figsize=(12,3))
          ax[0].scatter(xx, xx + 0.25*np.random.randn(len(xx)))
          ax[0].set_title("scatter")
          ax[1].step(n, n**2, lw=2)
          ax[1].set_title("step")
          ax[2].bar(n, n**2, align="center", width=0.5, alpha=0.5)
          ax[2].set_title("bar")
          ax[3].fill_between(n, n**2, n**3, color="green", alpha=0.5);
          ax[3].set_title("fill_between")

Out[171]: <matplotlib.text.Text at 0x7f00c3bc1790>
```

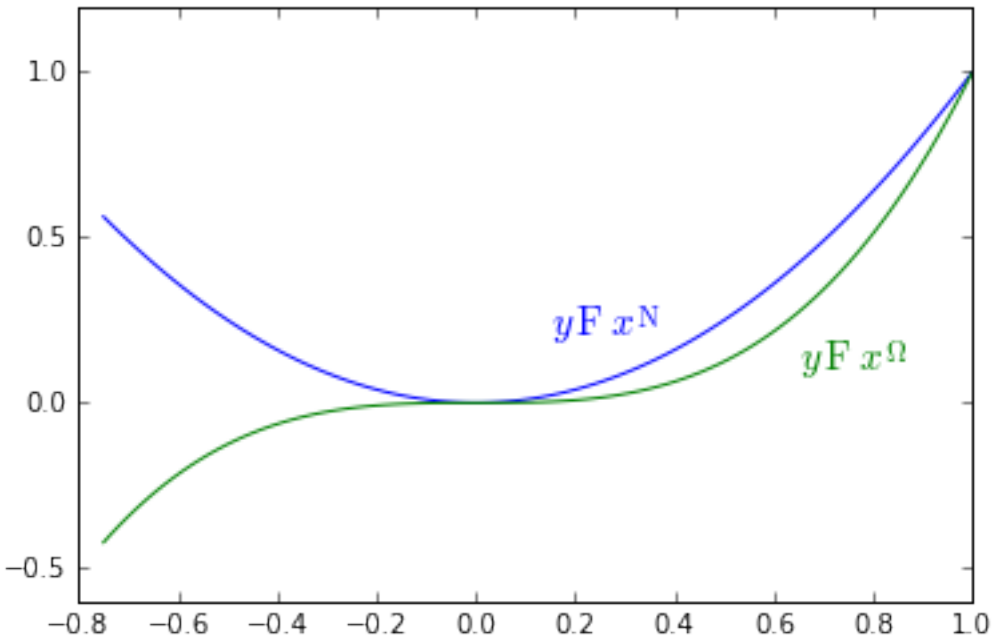


Generate: scatter, step, bar, fill_between

2.5 Task 5

Create a plot with annotations of the curves.

```
In [165]: x = np.linspace(-0.75, 1., 100)
          y_1 = x**2
          y_2 = x**3

          fig = plt.figure()
          ax = fig.add_subplot(111)
          ax.plot(xx, xx**2, xx, xx**3)
          ax.text(0.15, 0.2, "$y=x^2$", fontsize=15, color="blue")
          ax.text(0.65, 0.1, "$y=x^3$", fontsize=15, color="green");
```



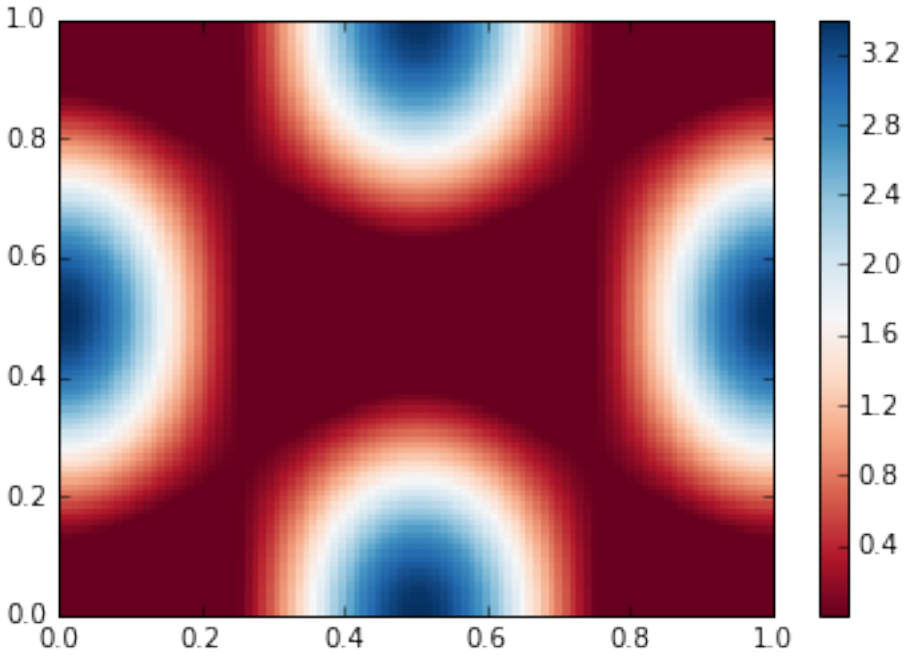
In [ ]:

2.6 Task 6

Create a color map using pcolor and colorbar functions for the following X,Y and Z

In [136]:
```
alpha = 0.7
phi_ext = 2 * pi * 0.5

def flux_qubit_potential(phi_m, phi_p):
    return ( + alpha - 2 * cos(phi_p)*cos(phi_m) -
        alpha * cos(phi_ext - 2*phi_p))

phi_m = linspace(0, 2*pi, 100)
phi_p = linspace(0, 2*pi, 100)
X,Y = meshgrid(phi_p, phi_m)
Z = flux_qubit_potential(X, Y).T
```

In [166]:
```
fig, ax = plt.subplots()
p = ax.pcolor(X/(2*np.pi), Y/(2*np.pi), Z, cmap=plt.cm.RdBu, vmin=abs(Z).
cb = fig.colorbar(p, ax=ax)
```



In [ ]:

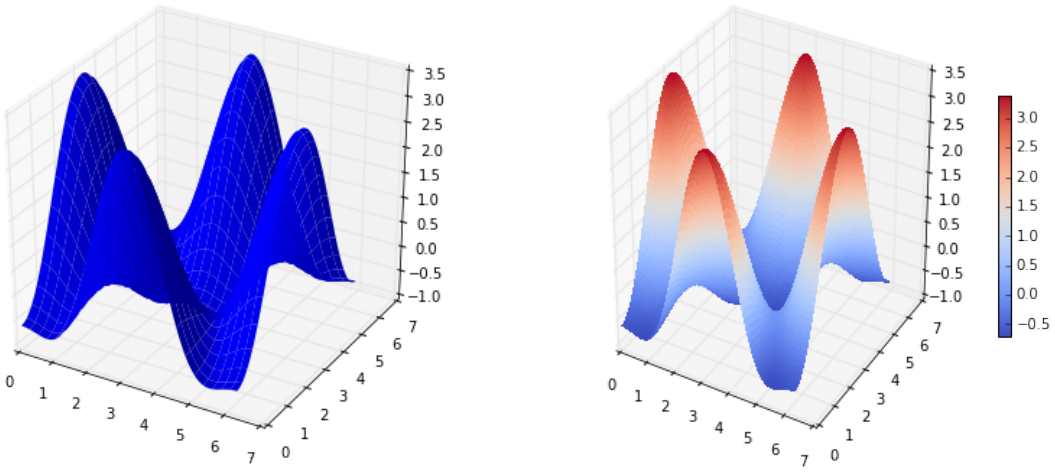
2.7 Task 7

For the same data (i.e. X,Y and Z) create plot_surface, plot_wireframe, contour plot with projections, using

```
In [168]: from mpl_toolkits.mplot3d.axes3d import Axes3D
```

Replicate the plots introduced below (you can use your own data for this)

```
In [170]: fig = plt.figure(figsize=(14,6))

          ax = fig.add_subplot(1, 2, 1, projection='3d')

          p = ax.plot_surface(X, Y, Z, rstride=4, cstride=4, linewidth=0)

          ax = fig.add_subplot(1, 2, 2, projection='3d')
          p = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.coolwarm,
          cb = fig.colorbar(p, shrink=0.5)
```




```
In [ ]:
```

Exercise2 Pandas

April 12, 2018

1 Pandas

Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python.

Library documentation: http://pandas.pydata.org/

1.0.1 General

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
```

1.1 Task 1

Create dataframe (that we will be importing)

```
In [2]:  data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
                 'last_name': ['Miller', 'Jacobson', ".", 'Milner', 'Cooze'],
                 'age': [42, 52, 36, 24, 73],
                 'preTestScore': [4, 24, 31, ".", "."],
                 'postTestScore': ["25,000", "94,000", 57, 62, 70]}

In [3]: df = pd.DataFrame(data, columns = ['first_name', 'last_name', 'age', 'preTe

In [4]: df
```

Out[4]:

	first_name	last_name	age	preTestScore	postTestScore
0	Jason	Miller	42	4	25,000
1	Molly	Jacobson	52	24	94,000
2	Tina	.	36	31	57
3	Jake	Milner	24	.	62
4	Amy	Cooze	73	.	70

1.2 Task 2

- Save dataframe as csv
- Load a csv

- Load a csv with no headers
- Load a csv while specifying column names
- Load a csv while skipping the top 3 rows

```
In [5]: #save to csv file
        df.to_csv('exported_data.csv',index=False)
```

```
In [6]: #load csv
        df = pd.read_csv('exported_data.csv')
        df
```

Out[6]:

	first_name	last_name	age	preTestScore	postTestScore
0	Jason	Miller	42	4	25,000
1	Molly	Jacobson	52	24	94,000
2	Tina	.	36	31	57
3	Jake	Milner	24	.	62
4	Amy	Cooze	73	.	70

```
In [9]: #load csv without headers
        df = pd.read_csv('exported_data.csv', header=None)
        df
```

Out[9]:

	0	1	2	3	4
0	first_name	last_name	age	preTestScore	postTestScore
1	Jason	Miller	42	4	25,000
2	Molly	Jacobson	52	24	94,000
3	Tina	.	36	31	57
4	Jake	Milner	24	.	62
5	Amy	Cooze	73	.	70

```
In [10]: #Load a csv while specifying column names
         df = pd.read_csv('exported_data.csv', names=['ID', 'First Name', 'Last Nam
         df
```

Out[10]:

	ID	First Name	Last Name	Age	Pre-Test Score	\
0	first_name	last_name	age	preTestScore	postTestScore	
1	Jason	Miller	42	4	25,000	
2	Molly	Jacobson	52	24	94,000	
3	Tina	.	36	31	57	
4	Jake	Milner	24	.	62	
5	Amy	Cooze	73	.	70	

	Post-Test Score
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
5	NaN

```
In [11]: #Load a csv while skipping the top 3 rows
         df = pd.read_csv('exported_data.csv', skiprows=3)
         df
```

```
Out[11]:    Tina       .  36 31  57
         0  Jake  Milner  24  .  62
         1   Amy   Cooze  73  .  70
```

```
In [ ]:
```

2 It is interesting to know and play around

```
In [60]: # create a series
         s = pd.Series([1,3,5,np.nan,6,8])
```

```
In [61]: # create a data frame
         dates = pd.date_range('20130101',periods=6)
         df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('ABCD'))
```

```
In [62]: # another way to create a data frame
         df2 = pd.DataFrame(
             { 'A' : 1.,
               'B' : pd.Timestamp('20130102'),
               'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
               'D' : np.array([3] * 4,dtype='int32'),
               'E' : 'foo' })
         df2
```

```
Out[62]:    A          B  C  D    E
         0  1 2013-01-02  1  3  foo
         1  1 2013-01-02  1  3  foo
         2  1 2013-01-02  1  3  foo
         3  1 2013-01-02  1  3  foo
```

```
In [63]: df2.dtypes
```

```
Out[63]: A           float64
         B    datetime64[ns]
         C           float32
         D             int32
         E            object
         dtype: object
```

```
In [64]: df.head()
```

```
Out[64]:                    A         B         C         D
         2013-01-01  1.264103  0.290035 -1.970288  0.803906
         2013-01-02  1.030550  0.118098 -0.021853  0.046841
         2013-01-03 -1.628753 -0.392361  1.700973  1.061330
         2013-01-04  0.695804 -0.435989 -0.332942  0.602135
         2013-01-05  0.108789  0.036767 -0.538963  0.499178
```

```
In [65]: df.index

Out[65]: DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
                        '2013-01-05', '2013-01-06'],
                       dtype='datetime64[ns]', freq='D')

In [66]: df.columns

Out[66]: Index([u'A', u'B', u'C', u'D'], dtype='object')

In [67]: df.values

Out[67]: array([[ 1.26410337,  0.29003478, -1.9702885 ,  0.80390589],
                [ 1.03055033,  0.11809794, -0.02185333,  0.04684071],
                [-1.62875286, -0.39236059,  1.70097271,  1.06132976],
                [ 0.69580357, -0.43598857, -0.33294162,  0.60213456],
                [ 0.10878896,  0.03676693, -0.53896341,  0.49917789],
                [-0.71195176, -0.23700097,  0.85711924, -1.8823519 ]])

In [68]: # quick data summary
         df.describe()

Out[68]:                A         B         C         D
         count  6.000000  6.000000  6.000000  6.000000
         mean   0.126424 -0.103408 -0.050992  0.188506
         std    1.115328  0.295114  1.256679  1.069268
         min   -1.628753 -0.435989 -1.970288 -1.882352
         25%   -0.506767 -0.353521 -0.487458  0.159925
         50%    0.402296 -0.100117 -0.177397  0.550656
         75%    0.946864  0.097765  0.637376  0.753463
         max    1.264103  0.290035  1.700973  1.061330

In [69]: df.T

Out[69]:    2013-01-01  2013-01-02  2013-01-03  2013-01-04  2013-01-05  2013-01-06
         A    1.264103    1.030550   -1.628753    0.695804    0.108789   -0.711952
         B    0.290035    0.118098   -0.392361   -0.435989    0.036767   -0.237001
         C   -1.970288   -0.021853    1.700973   -0.332942   -0.538963    0.857119
         D    0.803906    0.046841    1.061330    0.602135    0.499178   -1.882352

In [70]: # axis 0 is index, axis 1 is columns
         df.sort_index(axis=1, ascending=False)

Out[70]:                   D         C         B         A
         2013-01-01  0.803906 -1.970288  0.290035  1.264103
         2013-01-02  0.046841 -0.021853  0.118098  1.030550
         2013-01-03  1.061330  1.700973 -0.392361 -1.628753
         2013-01-04  0.602135 -0.332942 -0.435989  0.695804
         2013-01-05  0.499178 -0.538963  0.036767  0.108789
         2013-01-06 -1.882352  0.857119 -0.237001 -0.711952
```

```
In [71]: # can sort by values too
         df.sort(columns='B')

/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:2: FutureWarning: sort
```

```
Out[71]:                    A         B         C         D
         2013-01-04  0.695804 -0.435989 -0.332942  0.602135
         2013-01-03 -1.628753 -0.392361  1.700973  1.061330
         2013-01-06 -0.711952 -0.237001  0.857119 -1.882352
         2013-01-05  0.108789  0.036767 -0.538963  0.499178
         2013-01-02  1.030550  0.118098 -0.021853  0.046841
         2013-01-01  1.264103  0.290035 -1.970288  0.803906
```

2.0.1 Selection

```
In [72]: # select a column (yields a series)
         df['A']

Out[72]: 2013-01-01    1.264103
         2013-01-02    1.030550
         2013-01-03   -1.628753
         2013-01-04    0.695804
         2013-01-05    0.108789
         2013-01-06   -0.711952
         Freq: D, Name: A, dtype: float64

In [73]: # column names also attached to the object
         df.A

Out[73]: 2013-01-01    1.264103
         2013-01-02    1.030550
         2013-01-03   -1.628753
         2013-01-04    0.695804
         2013-01-05    0.108789
         2013-01-06   -0.711952
         Freq: D, Name: A, dtype: float64

In [74]: # slicing works
         df[0:3]

Out[74]:                    A         B         C         D
         2013-01-01  1.264103  0.290035 -1.970288  0.803906
         2013-01-02  1.030550  0.118098 -0.021853  0.046841
         2013-01-03 -1.628753 -0.392361  1.700973  1.061330

In [75]: df['20130102':'20130104']
```

```
Out[75]:                    A         B         C         D
         2013-01-02  1.030550  0.118098 -0.021853  0.046841
         2013-01-03 -1.628753 -0.392361  1.700973  1.061330
         2013-01-04  0.695804 -0.435989 -0.332942  0.602135

In [76]: # cross-section using a label
         df.loc[dates[0]]

Out[76]: A    1.264103
         B    0.290035
         C   -1.970288
         D    0.803906
         Name: 2013-01-01 00:00:00, dtype: float64

In [77]: # getting a scalar value
         df.loc[dates[0], 'A']

Out[77]: 1.26410336557194

In [78]: # select via position
         df.iloc[3]

Out[78]: A    0.695804
         B   -0.435989
         C   -0.332942
         D    0.602135
         Name: 2013-01-04 00:00:00, dtype: float64

In [79]: df.iloc[3:5,0:2]

Out[79]:                    A         B
         2013-01-04  0.695804 -0.435989
         2013-01-05  0.108789  0.036767

In [80]: # column slicing
         df.iloc[:,1:3]

Out[80]:                    B         C
         2013-01-01  0.290035 -1.970288
         2013-01-02  0.118098 -0.021853
         2013-01-03 -0.392361  1.700973
         2013-01-04 -0.435989 -0.332942
         2013-01-05  0.036767 -0.538963
         2013-01-06 -0.237001  0.857119

In [81]: # get a value by index
         df.iloc[1,1]

Out[81]: 0.1180979357631569
```

In [82]:
```
# boolean indexing
df[df.A > 0]
```

Out[82]:

	A	B	C	D
2013-01-01	1.264103	0.290035	-1.970288	0.803906
2013-01-02	1.030550	0.118098	-0.021853	0.046841
2013-01-04	0.695804	-0.435989	-0.332942	0.602135
2013-01-05	0.108789	0.036767	-0.538963	0.499178

In [83]:
```
df[df > 0]
```

Out[83]:

	A	B	C	D
2013-01-01	1.264103	0.290035	NaN	0.803906
2013-01-02	1.030550	0.118098	NaN	0.046841
2013-01-03	NaN	NaN	1.700973	1.061330
2013-01-04	0.695804	NaN	NaN	0.602135
2013-01-05	0.108789	0.036767	NaN	0.499178
2013-01-06	NaN	NaN	0.857119	NaN

In [84]:
```
# filtering
df3 = df.copy()
df3['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
df3[df3['E'].isin(['two', 'four'])]
```

Out[84]:

	A	B	C	D	E
2013-01-03	-1.628753	-0.392361	1.700973	1.061330	two
2013-01-05	0.108789	0.036767	-0.538963	0.499178	four

In [85]:
```
# setting examples
df.at[dates[0],'A'] = 0
df.iat[0,1] = 0
df.loc[:, 'D'] = np.array([5] * len(df))
df
```

Out[85]:

	A	B	C	D
2013-01-01	0.000000	0.000000	-1.970288	5
2013-01-02	1.030550	0.118098	-0.021853	5
2013-01-03	-1.628753	-0.392361	1.700973	5
2013-01-04	0.695804	-0.435989	-0.332942	5
2013-01-05	0.108789	0.036767	-0.538963	5
2013-01-06	-0.711952	-0.237001	0.857119	5

In [86]:
```
# dealing with missing data
df4 = df.reindex(index=dates[0:4],columns=list(df.columns) + ['E'])
df4.loc[dates[0]:dates[1],'E'] = 1
df4
```

Out[86]:

	A	B	C	D	E
2013-01-01	0.000000	0.000000	-1.970288	5	1
2013-01-02	1.030550	0.118098	-0.021853	5	1
2013-01-03	-1.628753	-0.392361	1.700973	5	NaN
2013-01-04	0.695804	-0.435989	-0.332942	5	NaN

```
In [87]: # drop rows with missing data
         df4.dropna(how='any')
```

```
Out[87]:                    A         B         C  D  E
         2013-01-01  0.00000  0.000000 -1.970288  5  1
         2013-01-02  1.03055  0.118098 -0.021853  5  1
```

```
In [88]: # fill missing data
         df4.fillna(value=5)
```

```
Out[88]:                    A         B         C  D  E
         2013-01-01  0.000000  0.000000 -1.970288  5  1
         2013-01-02  1.030550  0.118098 -0.021853  5  1
         2013-01-03 -1.628753 -0.392361  1.700973  5  5
         2013-01-04  0.695804 -0.435989 -0.332942  5  5
```

```
In [89]: # boolean mask for nan values
         pd.isnull(df4)
```

```
Out[89]:                 A      B      C      D      E
         2013-01-01  False  False  False  False  False
         2013-01-02  False  False  False  False  False
         2013-01-03  False  False  False  False   True
         2013-01-04  False  False  False  False   True
```

2.0.2 Operations

```
In [90]: df.mean()
```

```
Out[90]: A   -0.084260
         B   -0.151748
         C   -0.050992
         D    5.000000
         dtype: float64
```

```
In [91]: # pivot the mean calculation
         df.mean(1)
```

```
Out[91]: 2013-01-01    0.757428
         2013-01-02    1.531699
         2013-01-03    1.169965
         2013-01-04    1.231718
         2013-01-05    1.151648
         2013-01-06    1.227042
         Freq: D, dtype: float64
```

```
In [92]: # aligning objects with different dimensions
         s = pd.Series([1,3,5,np.nan,6,8],index=dates).shift(2)
         df.sub(s,axis='index')
```

```
Out[92]:                    A         B         C   D
         2013-01-01       NaN       NaN       NaN NaN
         2013-01-02       NaN       NaN       NaN NaN
         2013-01-03 -2.628753 -1.392361  0.700973   4
         2013-01-04 -2.304196 -3.435989 -3.332942   2
         2013-01-05 -4.891211 -4.963233 -5.538963   0
         2013-01-06       NaN       NaN       NaN NaN

In [93]: # applying functions
         df.apply(np.cumsum)

Out[93]:                    A         B         C   D
         2013-01-01  0.000000  0.000000 -1.970288   5
         2013-01-02  1.030550  0.118098 -1.992142  10
         2013-01-03 -0.598203 -0.274263 -0.291169  15
         2013-01-04  0.097601 -0.710251 -0.624111  20
         2013-01-05  0.206390 -0.673484 -1.163074  25
         2013-01-06 -0.505562 -0.910485 -0.305955  30

In [94]: df.apply(lambda x: x.max() - x.min())

Out[94]: A    2.659303
         B    0.554087
         C    3.671261
         D    0.000000
         dtype: float64

In [95]: # simple count aggregation
         s = pd.Series(np.random.randint(0,7,size=10))
         s.value_counts()

Out[95]: 1    3
         6    2
         2    2
         5    1
         3    1
         0    1
         dtype: int64
```

2.0.3 Merging / Grouping / Shaping

```
In [96]: # concatenation
         df = pd.DataFrame(np.random.randn(10, 4))
         pieces = [df[:3], df[3:7], df[7:]]
         pd.concat(pieces)

Out[96]:           0         1         2         3
         0 -0.131960 -1.414354  1.523244 -0.338449
         1  0.705489 -1.522506 -1.440273  0.228157
```

```
2  0.375323 -0.507394  1.995258 -2.653196
3  0.161172  0.498807 -1.410537 -2.405265
4 -0.832132 -0.859253  1.620849 -1.177659
5 -0.809916 -0.297847 -1.505251 -2.491711
6  0.944713 -1.317164  0.098182 -1.202643
7  1.156958  0.994988  0.506475  1.560681
8  1.342500 -0.605280  1.992343 -0.922623
9 -0.429583  0.426440  1.908732  2.042202
```

In [97]:
```
# SQL-style join
left = pd.DataFrame({'key': ['foo', 'foo'], 'lval': [1, 2]})
right = pd.DataFrame({'key': ['foo', 'foo'], 'rval': [4, 5]})
pd.merge(left, right, on='key')
```

Out[97]:

	key	lval	rval
0	foo	1	4
1	foo	1	5
2	foo	2	4
3	foo	2	5

In [98]:
```
# append
df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
s = df.iloc[3]
df.append(s, ignore_index=True)
```

Out[98]:

	A	B	C	D
0	1.279461	1.079281	0.541122	-0.287823
1	-0.111323	-0.627521	0.203337	-1.188971
2	1.616444	-0.316999	2.622289	0.166613
3	1.523248	0.589181	0.085152	-0.578820
4	-0.276869	-0.740993	-1.582538	-1.481723
5	0.692529	-0.156737	0.901637	1.153501
6	0.203035	0.963768	-1.049610	0.838853
7	0.264092	1.313649	-0.978666	-1.481115
8	1.523248	0.589181	0.085152	-0.578820

In [99]:
```
df = pd.DataFrame(
    { 'A' : ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'foo'],
      'B' : ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
      'C' : np.random.randn(8),
      'D' : np.random.randn(8) })
df
```

Out[99]:

	A	B	C	D
0	foo	one	-0.066725	-0.430368
1	bar	one	0.103149	2.274703
2	foo	two	0.434630	-0.601340
3	bar	three	-0.575259	0.305842
4	foo	two	0.570481	0.259178

```
5  bar    two -0.229949 -0.469119
6  foo    one -1.198411 -1.428004
7  foo  three -0.373095 -0.861415
```

```
In [100]: # group by
          df.groupby('A').sum()
```

```
Out[100]:             C         D
          A
          bar -0.702058  2.111426
          foo -0.633119 -3.061949
```

```
In [101]: # group by multiple columns
          df.groupby(['A','B']).sum()
```

```
Out[101]:                   C         D
          A   B
          bar one    0.103149  2.274703
              three -0.575259  0.305842
              two   -0.229949 -0.469119
          foo one   -1.265136 -1.858372
              three -0.373095 -0.861415
              two    1.005111 -0.342162
```

```
In [102]: df = pd.DataFrame(
              { 'A' : ['one', 'one', 'two', 'three'] * 3,
                'B' : ['A', 'B', 'C'] * 4,
                'C' : ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 2,
                'D' : np.random.randn(12),
                'E' : np.random.randn(12)} )
          df
```

```
Out[102]:         A  B    C         D         E
          0     one  A  foo  0.498432  2.197149
          1     one  B  foo  1.109739  1.043534
          2     two  C  foo  0.474046  0.182097
          3   three  A  bar -0.704602  0.658874
          4     one  B  bar -0.873703 -1.392941
          5     one  C  bar -1.419418  0.300034
          6     two  A  foo  0.599422 -0.748327
          7   three  B  foo -1.814723 -1.443186
          8     one  C  foo  1.845116  0.006333
          9     one  A  bar -0.151767 -1.618217
          10    two  B  bar  1.418789 -1.078039
          11  three  C  bar -0.091888  0.898184
```

```
In [107]: # pivot table
          pd.pivot_table(df, values='D', rows=['A', 'B'], columns=['C'])
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-107-6f79515beb31> in <module>()
      1 # pivot table
----> 2 pd.pivot_table(df, values='D', rows=['A', 'B'], columns=['C'])


TypeError: pivot_table() got an unexpected keyword argument 'rows'
```

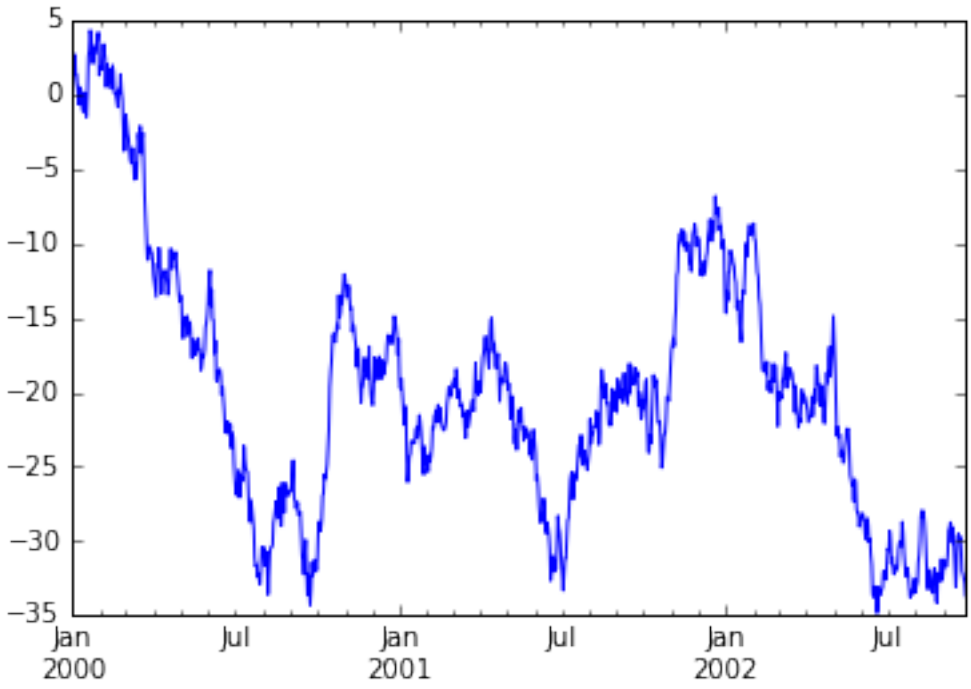
2.0.4 Time Series

```
In [108]: # time period resampling
          rng = pd.date_range('1/1/2012', periods=100, freq='S')
          ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)
          ts.resample('5Min', how='sum')

Out[108]: 2012-01-01    25140
          Freq: 5T, dtype: int64

In [109]: rng = pd.date_range('1/1/2012', periods=5, freq='M')
          ts = pd.Series(np.random.randn(len(rng)), index=rng)
          ts

Out[109]: 2012-01-31   -1.514774
          2012-02-29   -1.639687
          2012-03-31    0.576491
          2012-04-30    0.572730
          2012-05-31    1.348056
          Freq: M, dtype: float64

In [110]: ps = ts.to_period()
          ps.to_timestamp()

Out[110]: 2012-01-01   -1.514774
          2012-02-01   -1.639687
          2012-03-01    0.576491
          2012-04-01    0.572730
          2012-05-01    1.348056
          Freq: MS, dtype: float64
```

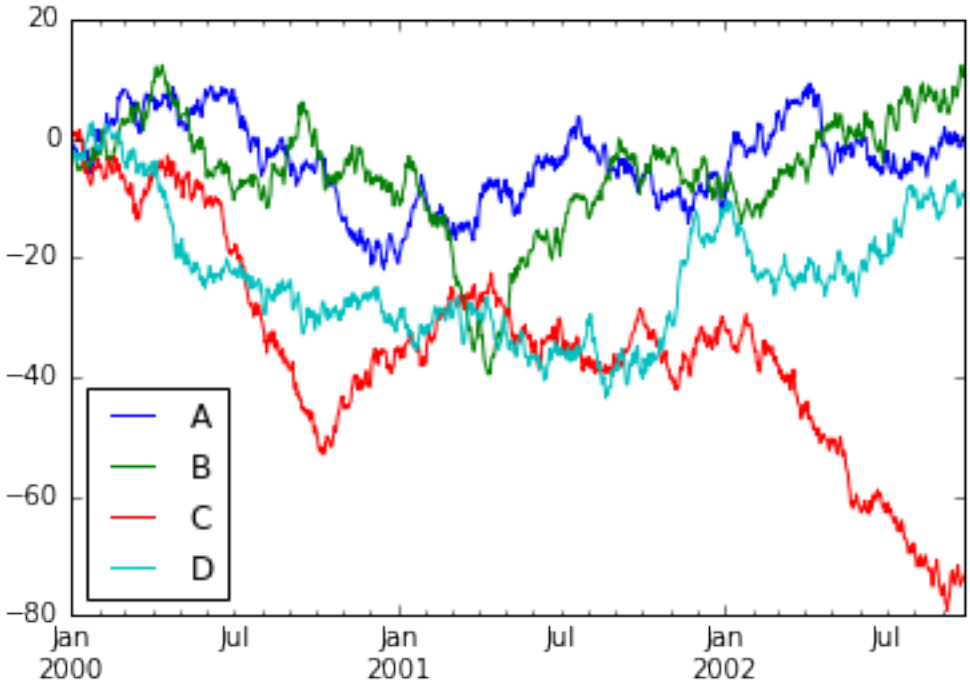
2.0.5 Plotting

```
In [111]: # time series plot
          ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', per
          ts = ts.cumsum()
          ts.plot()
```

```
Out[111]: <matplotlib.axes._subplots.AxesSubplot at 0x7efe0b50f610>
```




```
In [112]: # plot with a data frame
          df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=['A',
          df = df.cumsum()
          plt.figure(); df.plot(); plt.legend(loc='best')

Out[112]: <matplotlib.legend.Legend at 0x7efe0b3d79d0>

<matplotlib.figure.Figure at 0x7efe0b3e0c50>
```



2.0.6 Input / Output

```
In [113]: # write to a csv file
          df.to_csv('foo.csv', index=False)

In [116]: # read file back in
          path = r'exported_data.csv'
          newDf = pd.read_csv(path)
          newDf.head()
```

Out[116]:

	Unnamed: 0	first_name	last_name	age	preTestScore	postTestScore
0	0	Jason	Miller	42	4	25,000
1	1	Molly	Jacobson	52	24	94,000
2	2	Tina	.	36	31	57
3	3	Jake	Milner	24	.	62
4	4	Amy	Cooze	73	.	70

```
In [117]: # remove the file
          import os
          os.remove(path)

In [118]: # can also do Excel
          df.to_excel('foo.xlsx', sheet_name='Sheet1')

In [119]: newDf2 = pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['
          newDf2.head()
```

Out[119]:

	A	B	C	D
2000-01-01	0.498236	-0.566043	0.336430	-1.880709
2000-01-02	-0.138182	-2.221249	-0.008688	-1.734768
2000-01-03	-1.019832	-1.904256	1.114548	-2.660315
2000-01-04	-1.208775	-2.894402	0.377247	-3.262846
2000-01-05	-1.798420	-2.822972	0.855575	-2.200027

In [120]:
```
os.remove('foo.xlsx')
```

In [ ]:

Exercise3 NumPy

April 12, 2018

1 NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Library documentation: http://www.numpy.org/

```
In [1]: import numpy as np
```

2 Task 1: declare a vector using a list as the argument

```
In [4]: three_d_vector = np.array([1,2,3])
        three_d_vector

Out[4]: array([1, 2, 3])
```

3 Task 2: declare a matrix using a nested list as the argument

```
In [6]: matrix_example = np.array([[1,2,3],[4,5,6]])
        matrix_example

Out[6]: array([[1, 2, 3],
               [4, 5, 6]])
```

4 Task 3: initialize x or x and y using the following functions: arange, linspace, logspace, mgrid

```
In [24]: x_arange = np.arange(1,10,2)
         print "Numpy arange (1,10) with step size 2:"
         print x_arange
         x_linspace = np.linspace(1,10,2)
         print "Numpy linspace (uniform) with step size 2:"
         print x_linspace
         x_logspace = np.logspace(1,10,20, endpoint=True)
         print "Numpy logspace with endpoint True:"
         print x_logspace
         x_mgrid = np.mgrid[0:5, 0:5]
         print "Numpy mgrid (return mesh grid)"
         print x_mgrid
```

```
Numpy arange (1,10) with step size 2:
[1 3 5 7 9]
Numpy linspace (uniform) with step size 2:
[  1.  10.]
Numpy logspace with endpoint True:
[  1.00000000e+01   2.97635144e+01   8.85866790e+01   2.63665090e+02
   7.84759970e+02   2.33572147e+03   6.95192796e+03   2.06913808e+04
   6.15848211e+04   1.83298071e+05   5.45559478e+05   1.62377674e+06
   4.83293024e+06   1.43844989e+07   4.28133240e+07   1.27427499e+08
   3.79269019e+08   1.12883789e+09   3.35981829e+09   1.00000000e+10]
Numpy mgrid (return mesh grid)
[[[0 0 0 0 0]
  [1 1 1 1 1]
  [2 2 2 2 2]
  [3 3 3 3 3]
  [4 4 4 4 4]]

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

```
In [25]: from numpy import random
```

5 Task 4: what is difference between random.rand and random.randn

- random.rand returns random values given specific shape from uniform distribution
- random.randn returns a sample from a standard normal distribution

6 Task 5: what are the funciotns diag, itemsize, nbytes and ndim about?

- diag returns a diagonal array from a matrix
- itemsize returns the item size of ndarray
- nbytes returns total bytes consumed by a particular numpy array
- ndim returns the number of array dimensions

```
In [30]: # assign new value
         M = np.array([[1,2,3],[4,5,6]])
         M[0,0] = 7

In [31]: M[0,:] = 0

In [33]: # slicing works just like with lists
         A = np.array([1,2,3,4,5])
         A[1:3]

Out[33]: array([2, 3])
```

7 Task 6: Using list comprehensions create the following matrix

array([[0, 1, 2, 3, 4], [10, 11, 12, 13, 14], [20, 21, 22, 23, 24], [30, 31, 32, 33, 34], [40, 41, 42, 43, 44]])

```
In [35]: A = np.array([[ 0,  1,  2,  3,  4],
                       [10, 11, 12, 13, 14],
                       [20, 21, 22, 23, 24],
                       [30, 31, 32, 33, 34],
                       [40, 41, 42, 43, 44]])

In [36]: row_indices = [1, 2, 3]
         A[row_indices]

Out[36]: array([[10, 11, 12, 13, 14],
                [20, 21, 22, 23, 24],
                [30, 31, 32, 33, 34]])

In [39]: # index masking
         B = np.array([n for n in range(5)])
         row_mask = np.array([True, False, True, False, False])
         B[row_mask]

Out[39]: array([0, 2])
```

7.0.1 Linear Algebra

```
In [42]: v1 = np.arange(0, 5)

In [43]: v1 + 2
```

```
Out[43]: array([2, 3, 4, 5, 6])

In [44]: v1 * 2

Out[44]: array([0, 2, 4, 6, 8])

In [45]: v1 * v1

Out[45]: array([ 0,  1,  4,  9, 16])

In [46]: np.dot(v1, v1)

Out[46]: 30

In [47]: np.dot(A, v1)

Out[47]: array([ 30, 130, 230, 330, 430])

In [48]: # cast changes behavior of + - * etc. to use matrix algebra
         M = np.matrix(A)
         M * M

Out[48]: matrix([[ 300,  310,  320,  330,  340],
                 [1300, 1360, 1420, 1480, 1540],
                 [2300, 2410, 2520, 2630, 2740],
                 [3300, 3460, 3620, 3780, 3940],
                 [4300, 4510, 4720, 4930, 5140]])

In [51]: # inner product
         v1.T * v1

Out[51]: array([ 0,  1,  4,  9, 16])

In [55]: C = np.matrix([[1j, 2j], [3j, 4j]])
         C

Out[55]: matrix([[ 0.+1.j,  0.+2.j],
                 [ 0.+3.j,  0.+4.j]])

In [56]: np.conjugate(C)

Out[56]: matrix([[ 0.-1.j,  0.-2.j],
                 [ 0.-3.j,  0.-4.j]])

In [57]: # inverse
         C.I

Out[57]: matrix([[ 0.+2.j ,  0.-1.j ],
                 [ 0.-1.5j,  0.+0.5j]])
```

7.0.2 Statistics

```
In [58]: np.mean(A[:,3])

Out[58]: 23.0

In [60]: np.std(A[:,3]), np.var(A[:,3])

Out[60]: (14.142135623730951, 200.0)

In [61]: A[:,3].min(), A[:,3].max()

Out[61]: (3, 43)

In [63]: d = np.arange(1, 10)
         np.sum(d), np.prod(d)

Out[63]: (45, 362880)

In [64]: np.cumsum(d)

Out[64]: array([ 1,  3,  6, 10, 15, 21, 28, 36, 45])

In [65]: np.cumprod(d)

Out[65]: array([     1,      2,      6,     24,    120,    720,   5040,  40320,
                362880])

In [66]: # sum of diagonal
         np.trace(A)

Out[66]: 110

In [67]: m = np.random.rand(3, 3)

In [68]: # use axis parameter to specify how function behaves
         m.max(), m.max(axis=0)

Out[68]: (0.87468241317501216, array([ 0.6890535 ,  0.87468241,  0.81414742]))

In [69]: # reshape without copying underlying data
         n, m = A.shape
         B = A.reshape((1,n*m))

In [70]: # modify the array
         B[0,0:5] = 5

In [71]: # also changed
         A
```

```
Out[71]: array([[ 5,  5,  5,  5,  5],
                [10, 11, 12, 13, 14],
                [20, 21, 22, 23, 24],
                [30, 31, 32, 33, 34],
                [40, 41, 42, 43, 44]])

In [72]: # creates a copy
         B = A.flatten()

In [87]: # can insert a dimension in an array
         newaxis = 0
         v = np.array([1,2,3])
         #v[:, newaxis], v[:,newaxis].shape, v[newaxis,:].shape

In [80]: np.repeat(v1, 3)

Out[80]: array([1, 1, 1, 2, 2, 2, 3, 3, 3])

In [81]: np.tile(v1, 3)

Out[81]: array([1, 2, 3, 1, 2, 3, 1, 2, 3])

In [82]: w = np.array([5, 6])

In [83]: np.concatenate((v, w), axis=0)

Out[83]: array([1, 2, 3, 5, 6])

In [84]: # deep copy
         B = np.copy(A)

In [ ]:
```