Agenda

1. Installation
2. Basics
3. Iterables
4. Numpy (for math and matrix operations)
5. Matplotlib (for plotting)
6. Q&A

In [1]:
```
# Note: This tutorial is based on Python 3.8
#       but it should apply to all Python 3.X versions
# Please note that this tutorial is NOT exhaustive
# We try to cover everything you need for class assignments
# but you should also navigate external resources
#
# More tutorials:
# NUMPY:
# https://cs231n.github.io/python-numpy-tutorial/#numpy
# https://numpy.org/doc/stable/user/quickstart.html
# MATPLOTLIB:
# https://matplotlib.org/gallery/index.html
# BASICS:
# https://www.w3schools.com/python/
# CONSULT THESE WISELY:
# The official documentation, Google, and Stack-overflow are your friend
s!
```

1. Installation

Anaconda for environment management

https://www.anaconda.com/ (https://www.anaconda.com/)

common commands

conda env list <-- list all environments

conda create -n newenv python=3.8 <-- create new environment

conda enc create -f env.yml <-- create environment from config file

conda activate envname <-- activate a environment

conda deactivate <-- exit environment

pip install packagename <-- install package for current environment

jupyter notebook <-- open jupyter in current environment

Package installation using conda/pip

Live demo

Recommended IDEs

Spyder (in-built in Anaconda)

Pycharm (the most popular choice, compatible with Anaconda)

In [2]:
```
# common anaconda commands
#conda env list
#conda create -n name python=3.8
#conda env create -f env.yml
#conda activate python2.7
#conda deactivate
#install packages
#pip install <package>
```

4. Numpy

Very powerful python tool for handling matrices and higher dimensional arrays

In [8]:
```
import numpy as np
```

In [36]:
```
# create arrays
a = np.array([[1,2],[3,4],[5,6]], dtype='f8')
print(a)
print(a.shape)
# create all-zero/one arrays
b = np.ones((3,4)) # np.zeros((3,4))
print(b)
print(b.shape)
# create identity matrix
c = np.eye(5)
print(c)
print(c.shape)
```

```
[[ 1.  2.]
 [ 3.  4.]
 [ 5.  6.]]
(3, 2)
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
(3, 4)
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]
(5, 5)
```

In [39]:
```
# reshaping arrays
a = np.arange(8)         # [8,] similar range() you use in for-loops
b = a.reshape((4,2))     # shape [4,2]
c = a.reshape((2,2,-1))  # shape [2,2,2] -- -1 for auto-fill
d = c.flatten()          # shape [8,]
e = np.expand_dims(a, 0) # [1,8]
f = np.expand_dims(a, 1) # [8,1]
g = e.squeeze()          # shape[8, ]    -- remove all unnecessary dimens
ions
print(a)
print(b, c, d, e,f)
```

```
[0 1 2 3 4 5 6 7]
(array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]]), array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]]), array([0, 1, 2, 3, 4, 5, 6, 7]), array([[0, 1, 2, 3,
4, 5, 6, 7]]), array([[0],
       [1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7]]))
```

In [35]:
```
# concatenating arrays
a = np.ones((4,3))
b = np.ones((4,3))
c = np.concatenate([a,b], 0)
print(c.shape)
d = np.concatenate([a,b], 1)
print(d.shape)
```

```
(8, 3)
(4, 6)
```

In [36]:
```
# one application is to create a batch for NN
x1 = np.ones((32,32,3))
x2 = np.ones((32,32,3))
x3 = np.ones((32,32,3))
# --> to create a batch of shape (3,32,32,3)
x = [x1, x2, x3]
x = [np.expand_dims(xx, 0) for xx in x] # xx shape becomes (1,32,32,3)
x = np.concatenate(x, 0)
print(x.shape)
```

```
(3, 32, 32, 3)
```

In [41]:
```
# access array slices by index
a = np.zeros([10, 10])
a[:3] = 1
a[:, :3] = 2
a[:3, :3] = 3
rows = [4,6,7]
cols = [9,3,5]
a[rows, cols] = 4
print(a)
```

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

In [38]:
```
# transposition
a = np.arange(24).reshape(2,3,4)
print(a.shape)
print(a)
a = np.transpose(a, (2,1,0)) # swap 0th and 2nd axes
print(a.shape)
print(a)
```

```
(2, 3, 4)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
(4, 3, 2)
[[[ 0 12]
  [ 4 16]
  [ 8 20]]

 [[ 1 13]
  [ 5 17]
  [ 9 21]]

 [[ 2 14]
  [ 6 18]
  [10 22]]

 [[ 3 15]
  [ 7 19]
  [11 23]]]
```

In [39]:
```
c = np.array([[1,2],[3,4]])
# pinv is pseudo inversion for stability
print(np.linalg.pinv(c))
# l2 norm by default, read documentation for more options
print(np.linalg.norm(c))
# summing a matrix
print(np.sum(c))
# the optional axis parameter
print(c)
print(np.sum(c, axis=0)) # sum along axis 0
print(np.sum(c, axis=1)) # sum along axis 1
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
5.477225575051661
10
[[1 2]
 [3 4]]
[4 6]
[3 7]
```

In [40]:
```
# dot product
c = np.array([1,2])
d = np.array([3,4])
print(np.dot(c,d))
```

```
11
```

In [41]:
```
# matrix multiplication
a = np.ones((4,3)) # 4,3
b = np.ones((3,2)) # 3,2 --> 4,2
print(a @ b)       # same as a.dot(b)
c = a @ b          # (4,2)

# automatic repetition along axis
d = np.array([1,2,3,4]).reshape(4,1)
print(c + d)
# handy for batch operation
batch = np.ones((3,32))
weight = np.ones((32,10))
bias = np.ones((1,10))
print((batch @ weight + bias).shape)
```

```
[[3. 3.]
 [3. 3.]
 [3. 3.]
 [3. 3.]]
[[4. 4.]
 [5. 5.]
 [6. 6.]
 [7. 7.]]
(3, 10)
```

In [42]:
```
# speed test: numpy vs list
a = np.ones((100,100))
b = np.ones((100,100))

def matrix_multiplication(X, Y):
    result = [[0]*len(Y[0]) for _ in range(len(X))]
    for i in range(len(X)):
        for j in range(len(Y[0])):
            for k in range(len(Y)):
                result[i][j] += X[i][k] * Y[k][j]
    return result

import time

# run numpy matrix multiplication for 10 times
start = time.time()
for _ in range(10):
    a @ b
end = time.time()
print("numpy spends {} seconds".format(end-start))

# run list matrix multiplication for 10 times
start = time.time()
for _ in range(10):
    matrix_multiplication(a,b)
end = time.time()
print("list operation spends {} seconds".format(end-start))

# the difference gets more significant as matrices grow in size!
```

```
numpy spends 0.001990079879760742 seconds
list operation spends 8.681961059570312 seconds
```

In [43]:
```
# element-wise operations, for examples
np.log(a)
np.exp(a)
np.sin(a)
# operation with scalar is interpreted as element-wise
a * 3
```

Out[43]:
```
array([[3., 3., 3., ..., 3., 3., 3.],
       [3., 3., 3., ..., 3., 3., 3.],
       [3., 3., 3., ..., 3., 3., 3.],
       ...,
       [3., 3., 3., ..., 3., 3., 3.],
       [3., 3., 3., ..., 3., 3., 3.],
       [3., 3., 3., ..., 3., 3., 3.]])
```

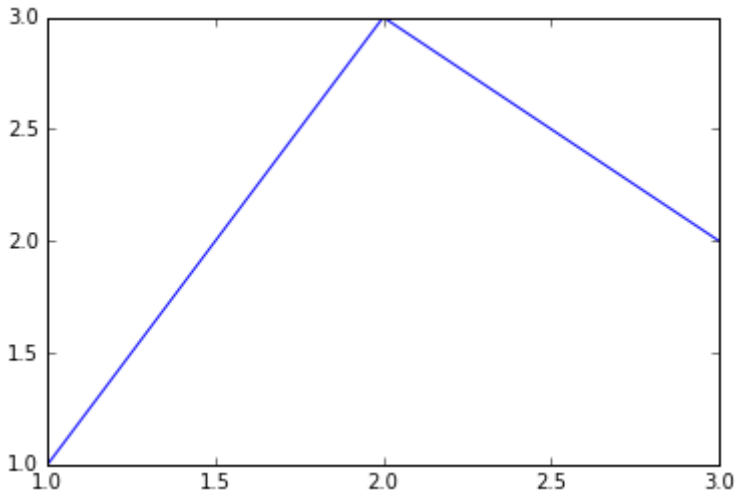
5. Matplotlib

Powerful tool for visualization
Many tutorials online. We only go over the basics here

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

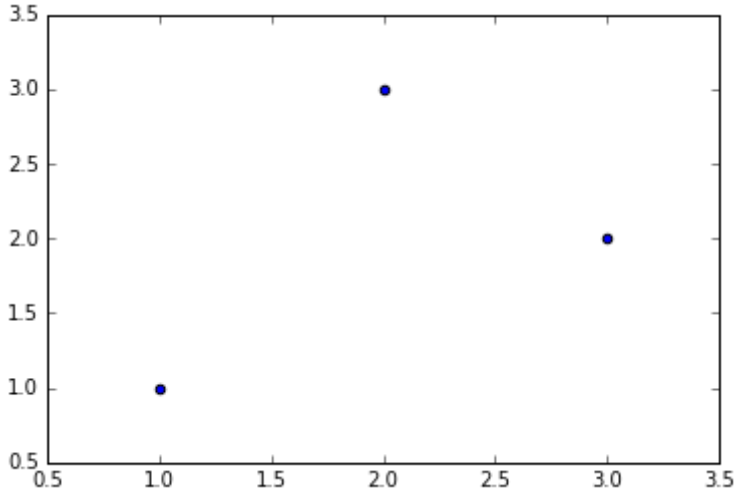
In [4]:
```
# line plot
x = [1,2,3]
y = [1,3,2]
plt.plot(x,y)
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x7f1235fa9d50>]
```



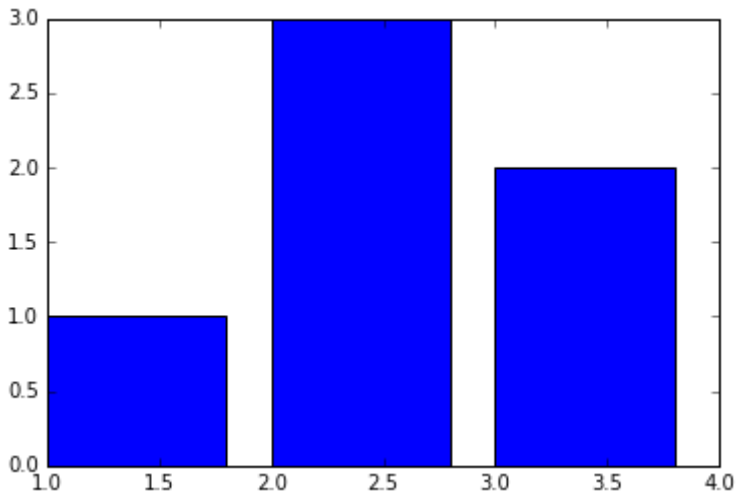
In [6]:
```
# scatter plot
plt.scatter(x,y)
```

Out[6]: <matplotlib.collections.PathCollection at 0x7f1235e3ee10>



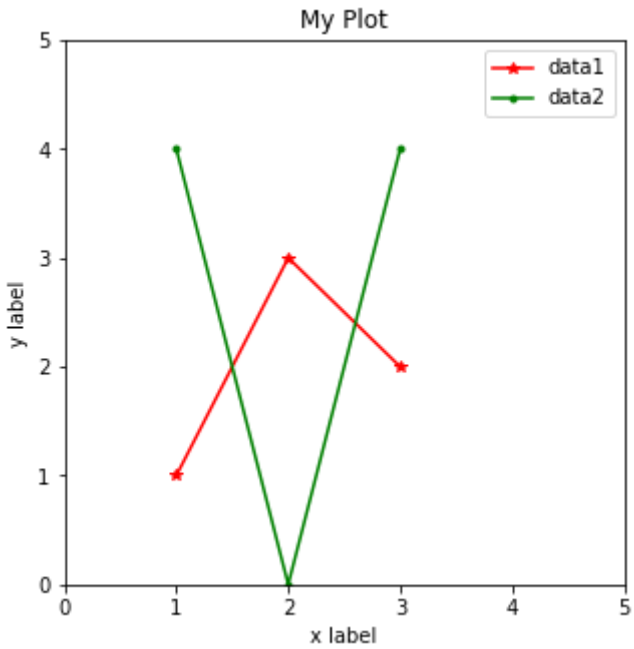
In [5]:
```
# bar plots
plt.bar(x,y)
```

Out[5]: <Container object of 3 artists>



In [48]:
```
# plot configurations
x = [1,2,3]
y1 = [1,3,2]
y2 = [4,0,4]

# set figure size
plt.figure(figsize=(5,5))

# set axes
plt.xlim(0,5)
plt.ylim(0,5)
plt.xlabel("x label")
plt.ylabel("y label")

# add title
plt.title("My Plot")

plt.plot(x,y1, label="data1", color="red", marker="*")
plt.plot(x,y2, label="data2", color="green", marker=".")
plt.legend()
```

Out[48]: <matplotlib.legend.Legend at 0x17b1b669d00>



Q&A

In [29]:
```
mu, sigma = 40, 5 # mean and standard deviation
s = np.random.normal(mu, sigma, 1000)
si=np.round(s)
```

In [30]:
```
hist = np.histogram(si, 30)
print(hist)
plt.bar(hist[1][1:], hist[0])
```

```
(array([ 3,  3,  7,  8, 14, 20, 20, 26, 45, 46, 60, 69, 72, 75, 76, 71, 6
1,
       73, 68, 49, 42, 30, 22, 13, 10,  8,  1,  2,  5,  1]), array([ 26.
,  26.96666667,  27.93333333,  28.9       ,
        29.86666667,  30.83333333,  31.8       ,  32.76666667,
        33.73333333,  34.7       ,  35.66666667,  36.63333333,
        37.6       ,  38.56666667,  39.53333333,  40.5       ,
        41.46666667,  42.43333333,  43.4       ,  44.36666667,
        45.33333333,  46.3       ,  47.26666667,  48.23333333,
        49.2       ,  50.16666667,  51.13333333,  52.1       ,
        53.06666667,  54.03333333,  55.        ]))
```

Out[30]: <Container object of 30 artists>