Numpy Tutorial

What is NumPy? Well we have been using it but what is it. Really all it is a package used for Scientific Computing with Python. This will a **BRIEF** overview of Numpy really the best way to learn about these functions is from doing little projects and posting them on your Github. You also have to learn how to look through an API or a package index and be able to see which functions are located within the package.

A code sample for numpy is the following:

Numpy arrays

```
import numpy as np #importing numpy as np
data=[[2,6,1,3], # creating two numpy tuples
[5,10,4,9]] #continuation of tuple
data=np.array(data) # converts the tuple to a numpy array and sets it back equal to data
print(data) # prints out two numpy arrays
#[[2 6 1 3],
#[5 10 4 9]]
print(data.shape)
#(2,4) 2 rows and 4 columns
print(np.zeros((3,3))) # prints an array which is 3 by 3 full of zeros  note it doesn't need to be
#square
#[[0. 0. 0.]
#[0. 0. 0.]
#[0. 0. 0.]]
 print(np.ones((3,6))) # prints a 3 row and 6 column array full of ones
#[[1. 1. 1. 1. 1. 1.]
 #[1. 1. 1. 1. 1. 1.]
 #[1. 1. 1. 1. 1. 1.]]
 array=np.arange(10) # set an array of 10 values 0-9
 print(array) # print the contents of array
#[0 1 2 3 4 5 6 7 8 9]
print(array[3:6])
#slicing array [3 4 5] is result
array[5:8]=0 # sets index from 5 up to not including 8 to 0
print(array) # print array contents
```

Array Operations

```
data=np.array([[5,6,8],
 [8,9,19]]) # setting the data variable equal to 2 numpy array
 print(data+5) # add 5 to each value in the numpy array
#[[10 11 13]
 #[13 14 24]]
print(data-3) # subtract the value 3 from all the data values
```

```
#[[ 2  3  5]
 #[ 5  6 16]]

print(data*data) # multiples the data arrayby itself
#[[ 25  36  64]
 #[ 64  81 361]]
print(data**2)#square of the data array
#[[ 25  36  64]
 #[ 64  81 361]]
print(data**3) # cube of the data array
#[[ 125  216  512]
 #[ 512  729 6859]]
#Transposing the array and swapping axis of the array
data=np.arange(15).reshape((5,3))#the range of 0-14  and reshape of 5 rows and 3 #columns
data
#array([[ 0,  1,  2],
   #       [ 3,  4,  5],
   #       [ 6,  7,  8],
   #       [ 9, 10, 11],
   #       [12, 13, 14]])
data=np.arange(16).reshape((4,4)) # 0-15 4 by 4 array
print(data) # prints out array
#[[ 0  1  2  3]
 #[ 4  5  6  7]
 #[ 8  9 10 11]
# [12 13 14 15]]
>>> print(data.T) # transpose the array all the rows turn into the columns and all the #columns
become the rows
#[[ 0  4  8 12]
 #[ 1  5  9 13]
 #[ 2  6 10 14]
 #[ 3  7 11 15]]
 x=np.random.randn(3,4) # an array of floating integers which is 3 by 4 but can be any value
#specified
print(x) # prints out x
[[-0.01460912  1.84547612 -2.66349986  0.5791522 ]
 [-1.58242171  0.00626904 -0.68614246 -0.60487307]
 [ 0.69393671  0.00560702  0.00586001  0.64677434]]
X #prints out just by typing x
array([[-0.01460912,  1.84547612, -2.66349986,  0.5791522 ],
       [-1.58242171,  0.00626904, -0.68614246, -0.60487307],
       [ 0.69393671,  0.00560702,  0.00586001,  0.64677434]])
```

More on Arrays

```
a=np.array([1,2,3,4])
print(type(a)) # prints the class numpy.ndarray
print(a.shape) # prints (4,) which is the dimension of the 1-Dimensional array
print(a[0],a[1],a[2]) # prints 1 2 3
a[1]=6 # sets the index of the array at position 1 to 6
print(a) # [1 6 3 4]

b=np.array([[1,2,3],[4,5,6]]) # define a 2 dimensional array
print(b.shape) # (2,3)
print(b[0,0],b[0,1],b[1,2]) #prints 1 2 6

a=np.zeros((3,3)) #prints a 3 by 3 array full of 0's
 #[[0. 0. 0.]
# [0. 0. 0.]
 #[0. 0. 0.]]

b=np.ones((3,5)) #3 rows 5 columns
print(b)
#[[1. 1. 1. 1. 1.]
 #[1. 1. 1. 1. 1.]
 #[1. 1. 1. 1. 1.]]

c=np.full((100,100),1) #creates a 100 by 100 matrix full of 1's note can be any number not 1
print(c) # print the array
#[[1 1 1 ... 1 1 1]
 #[1 1 1 ... 1 1 1]
 #[1 1 1 ... 1 1 1]
# ...
 #[1 1 1 ... 1 1 1]
 #[1 1 1 ... 1 1 1]
 #[1 1 1 ... 1 1 1]]

d=np.eye(2) # creates a 2 by 2 identity matrix which means putting 1's on the diagonal
print(d) # print out the array
#[[1. 0.]
 #[0. 1.]]

e=np.random.random((3,3)) # create an array filled with random values
print(e)
#[[0.01977006 0.90894143 0.09457297]
 #[0.46140166 0.89627253 0.03047473]
 #[0.85571882 0.94680604 0.52270272]
```

Array Indexing

In this section of the tutorial we are going to go over slicing arrays in numpy I know we did some slicing in numpy arrays and lists.

```
import numpy as np
a=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]]) # create 3  by 4 array

#the next line of code will be using slicing to pull out the subarray of the first 2 rows and
#columns 1 and 2. B will yield a 2 by 2 array the first value the 2 controls the number of rows
#which we are slicing and the second value controls the columns check out a couple of the
#code snippets
b=a[:2,1:3]
print(b)
# [[2 3]
#[6 7]]
b=a[:3,0:3]
print(b)
#[[ 1 2 3]
#[5 6 7]
#[9 10 11]
print(a[0,1])
#prints 2 0th  row column 1
b[0,0]=89 # set 0th row and 0th column value to 89
b[1,0]=100 # set row 1 column 0 value to 100
print(a[0,0]) # prints 89
print(a[1,0]) # prints 100
```

We can also mix integer indexing with slicing. It is important to note when we do this the shape of the matrix and dimension can change.

```
a=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
# what we are going to do next is some other techniques with slicing this is kind of a tricky
#topic so don't feel discouraged if you aren't understanding it completely trust me  while

row1=a[1,:] # slice the 2nd row of the array since it all starts at 0
row12=a[1:2,:] # slice the 2nd row of the array but the 1:2 slices the entire contents
print(row1) #[5 6 7 8]
print(row12) #[[ 5 6 7 8]]
print(row1.shape) # prints (4,)
print(row12.shape) # prints (1,4)
# we can do the same thing slicing columns
col1=a[:,1]
col12=a[:,1:2]
print(col1) # [2 6 10]
print(col12)
#[[2]
#[6]
#[10]]
print(col1.shape) #(3,)
print(col12.shape) #(3,1)
```

Integer array indexing

When you index into numpy arrays using slicing the method we just went over the array displayed will be a subarray of the original array. In turn we are taking an integer array and indexing it which allows us to construct arbitrary arrays using the data from another array.

```
a=np.array([[1,2],[3,4],[5,6]])
print(a[[0,1,2],[0,1,0]]) #  [1 4 5] is printed  and this is will become clearer in the next line of
#code. We are distributing the index values 00 11 20
print(np.array([a[0,0],a[1,1],a[2,0]])) # prints [1 4 5 ]
print(a[[0,0],[1,1]]) # distributing the index of 0,1 0,1 so the result is [2,2]
print(np.array([a[0,1],a[0,1]])) # prints 2 2 same as the prior examples distributed index #values
```

Boolean array indexing

The last idea with indexing that I want to cover is Boolean array indexing. Boolean array indexing allows us to pick arbitrary elements of an array frequently this type of indexing is used to select the elements of an array that satisfy a given condition such as less than or greater than.

```
a=np.array([[1,2],[3,4],[5,6]]) # here is the given array A

bool_index=(a>=2) # this is saying each value in the array a which is greater than or equal #to 2
is true

print(bool_index)#print out in the array if it is true or false
 #[[False  True]
 #[ True  True]
 #[ True  True]]

print(a[bool_index])  # [2 3 4 5 6]

# wow that was a lot 3 lines of code to do all that I wish there was an easier way maybe
#something like the next line of code
print(a[a>=2]) # prints [2 3 4 5 6]
# for more about indexing check out the documentation on the website it is below the website
```

Data Types

Every numpy array is a grid of elements of the same type (usually at least). Numpy allows for a large set of numeric data types that you can use to construct array. Numpy you may notice allows you construct an array with specifying the type this is because it tries to figure out the data type when you create an array. But the functions that construct arrays usually also include an optional argument to specify the data type.

Here I will show you how numpy guesses datatypes or how we can specify them:

```
x=np.array([1,2]) #allows numpy to choose the data type
print(x.dtype) # prints out int64

x=np.array([1.0,2.0])
print(x.dtype) # prints out float 64

x=np.array([1,2], dtype=np.int64) # specifying the data type
print(x.dtype) # pints out int64
```

Array Math

Mathematical functions operate element-wise on arrays and are available both as operators and function within the numpy module. We are going to look at a couple code samples of the following:

```
x=np.array([[1,2],[3,4]],dtype=np.float64)
y=np.array([[6,7],[8,9]],dtype=np.float64)
print(x+y)
[[ 7.  9.]
 [11. 13.]]
print(np.add(x,y))
[[ 7.  9.]
 [11. 13.]]
print(np.subtract(x,y))
[[-5. -5.]
 [-5. -5.]]
print(x-y)
[[-5. -5.]
 [-5. -5.]]
print(np.multiply(x,y))
[[ 6. 14.]
 [24. 36.]]
print(x*y)
[[ 6. 14.]
 [24. 36.]]
print(np.divide(x,y))
[[0.16666667 0.28571429]
```

```
 [0.375       0.44444444]]
print(x/y)
[[0.16666667 0.28571429]
 [0.375       0.44444444]]
print(np.sqrt(x))
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

<u>Dot Product</u>

```
 import numpy as np

x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
... print(v.dot(w))
219
print(np.dot(v, w))
219
print(np.dot(x, y))
[[19 22]
 [43 50]]
```

A very important topic which I think is important is about dot product such as inner products of vectors. We can use this to multiply a vector by a matrix or a matrix by another matrix.
If you don't know how to calculate inner products. Check below:

The Dot product between two vectors or values is based on the projection one vector onto another. In order to get the dot product between two vectors we typically perform a certain computation. (this is a very vague definition as this is not a calculus class where we could talk about the definition for hours)

Dot product is used widely in the field of mechanical and electrical engineering and now Artificial intelligence. Think of a dot product of something which will tell you how similar in direction a vector such as vector A is to vector B through the measure of an angle between them(note for my examples I will not be using examples).

An example of this for how to deal with dot product is below:

Dot product 1: A=(3,5,7)
Dot Product 2 B=(2,4,6)

We can use something called the component formula for the dot product of the three dimensional vectors:

A * B=$A_1 * B_1 + A_2 + B_2 + A_3 + B_3 \ldots$= $(3 * 2) + (5 * 4) + (7 * 6) = 68$

Since A*B is positive a rule to the vector dot product is that the vectors form an acute angle(I will not be going into what vectors are in here unless asked)

For 2 by 2 matrix values when we calculate the dot product there is a predefined formula.

If we have 2: 2x2 matrices the dot product computation is the following:

$$\begin{matrix}[A & B] \\ [C & D]\end{matrix} \quad * \quad \begin{matrix}[E & F] \\ [G & H]\end{matrix} \quad = \quad \begin{matrix}[AE + BG & AF + BH] \\ [CE + DG & CF + DH]\end{matrix}$$

$$\begin{matrix}[1 & 2] \\ [3 & 4]\end{matrix} \quad * \quad \begin{matrix}[5 & 6] \\ [7 & 8]\end{matrix} \quad = \quad \begin{matrix}[((1 * 5) + (2 * 7) & ((1 * 6) + (2 * 8))] = [19 \quad 22] \\ [((3 * 5) + (4 * 7)) & ((3 * 6) + (4 * 8))] = [43 \quad 50]\end{matrix}$$

Statistical Methods

```
 data=np.random.randn(3,2) # make a random float array which is 3 rows and 2 columns
 print(data) # prints the data array
#[[ 1.72444937  0.47942391]
 #[ 1.30403964 -0.72850969]
 #[-0.70395462 -0.37877009]]
 print(data.mean)
#see how it throws an error <built-in method mean of numpy.ndarray object at
#0x0000000009D0E7B0> this error means the function we are calling is not being called #we
need the "()"
 print(data.mean()) # takes the mean of the array
0.28277975457268556 # mean of array
 print(data.min()) # looking for the minimum of the array
-0.7285096854724838 # min value of array
 print(data.max()) # maximum value of the array
1.724449373153531 # max value of array
```

Matrix Classes

```
X=np.random.randn(2,2) # create 2 by 2 array
X=np.matrix(X) # change it to a matrix now
Y=np.random.randn(2,2) # create a 2 by 2 array
Y=np.matrix(Y) #change it to a matrix
z=X*Y # multiply the 2 matrix values together
print(z) # display the result z
```

I can't give you everything about numpy but where's the fun in that Check out the NumPy Documentation for more information about it::D.

Numpy Documentation:

https://docs.scipy.org/doc/numpy-1.12.0/reference/index.html

When I first started learning Numpy I used this website also it's a great link:
http://cs231n.github.io/python-numpy-tutorial/