Lesson2 Using Numpy Last Update Sept 7, 2020

Numpy lets you do additional operations on arrays. **Numpy arrays** are similar to Python **lists** but are not the same and have more features. Numpy arrays are made from python **lists**. To use numpy arrays you must import the numpy package at the top of your python file. Numpy stands for numeric python.

```
import numpy as np
```

as np means instead of saying **numpy** you can say **np** instead for convenience

recap: numpy arrays are array objects that you create from python lists.

Create numpy arrays from python lists

Create a 1 d numpy array

To create numpy array of 5 elements you would use this statement:

```
a = np.array([1,2,3,4,5])
```

Print out numpy array **a:**

```
[ 1 2 3 4 5]
```

Numpy arrays have a **rank**, **shape** and **type**.

rank	number of dimensions	1
shape	size of the array for each dimension return as a tuple (size1, size2, size3)	(5,) (5 rows no columns)
type	data type like int or float	int, float

a = np.array([1,2,3,4,5])

The above array would have rank of 1 dimensions of shape size (5,) and data type int. (5,) means 1 dimension of 5 elements.

print(np.rank(a)) # 1
print(a.shape) # (5,)
print(a.dtype) # int32

Create a 2 d numpy array

We can make a 2 dimensional numpy array from a 2 dimensional python array

b = np.array([[1,2,3][4,5,6]])

Print out numpy array **b:**

```
[[1 2 3]
 [4 5 6]]
```

b = np.array([[1,2,3][4,5,6]])

The above array would have rank of 2 dimensions of shape size (2,3) and data type int and 2 rows and 3 columns

print(np.rank(b)) # 2
print(b.shape) # (2,3)
print(b.dtype) # int32

You can use numpy arrays just like python lists

a = np.array([1,2,3,4,5] # [1,2,3,4,5]

a[0] = 5; # assign 5 to index 0 of numpy array a

x = a[0] ; #retrieve value from number array a at index 0

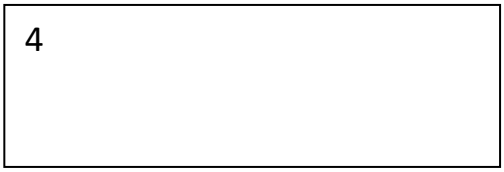
print(x) # print out x

x = a[-1] ; #retrieve value from array a at last index

read rows and columns from a 2 dimensional array

b = np.array[[1,2,3][4,5,6]]

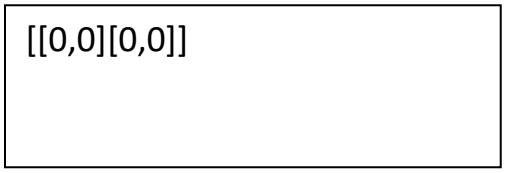
x = b[1][0]; #retrieve value from array row index 1 column index 0



creating Numpy arrays

Numpy has many function's to create arrays as follows:

 # Create a 2x2 array of all zeros
a = np.zeros((2,2))



 # Create a 2x2 array of all ones
a = np.ones((2,2))

```
[[1,1][1,1]]
```

 # Create a 2x2 array of a specified value
a = np.full((2,2), 5)

```
[[5,5][5,5]]
```

 # Create a 2x2 identity matrix
a = np.eye(2)

```
[[1,0][0,1]]
```

 # Create an array with random values 0 to 1
a = np.random.random((2,2))

```
[[0.5484408  0.57400321],[0.19896352 0.65698084]]
```

#create an array with a sequence of 0 to 18 by step of 2
a = np.arange(0, 20, 2)

```
[0, 2, 4, 6, 8,10,12,14,16,18]
```

#create an array of even space between a given range of values
 # (divides 0 to 1 evenly by 5)
a = np.linspace(0, 1, 5)

```
([ 0., 0.25, 0.5 , 0.75, 1.])
```

#create a 3x3 array with mean 0 and standard deviation 1
 # for a specified array dimension
a = np.random.normal(0, 1, (3,3))

```
[[ 0.72432142, -0.90024075, 0.27363808],
 [ 0.88426129, 1.45096856, -1.03547109],
[-0.42930994, -1.02284441, -1.59753603]]
```

Concatenating Numpy Arrays together

 # Concatenate 1 Dimensional arrays together

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

```
[1 2 3 4 5 6]
```

 # Concatenate 2 Dimensional arrays together (default: row-wise axis=0)

```
x = np.array([[1, 2, 3],[4,5,6]])
y = np.array([[1,2,3],[4,5,6]])
b = np.concatenate([x, y])  # default axis = 0 (row axis)
```

```
[[1 2 3]
 [4 5 6]
 [1 2 3]
[4 5 6]]
```

 # Concatenate 2 Dimensional arrays together (column-wise)

```
x = np.array([[1, 2, 3],[4,5,6]])
y = np.array([[1,2,3],[4,5,6]])
b = np.concatenate([x, y],axis=1)     # 1 = column axis
```

```
[[1 2 3 1 2 3]
[4 5 6 4 5 6]]
```

vstack (vertical stack)

vstack stacks a 1 d array oh the top of a 2 d array **vertically**.

before:

```
[[4 5 6]
 [7 8 9]]
```

after:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
# stack 1d array on top of a 2d array
a = np.array([1,2,3])
b = np.array([[4,5,6],[7,8,9]])
x = np.vstack([a,b])
```

```
[[1 2 3]
[4 5 6]
 [7 8 9]]
```

hstack (horizontal stack)

hstack stacks a 1 d array on the top of a 2 d array **horizontally**.

Before:

```
[[4 5 6]
 [7 8 9]]
```

After:

```
 [[1 4 5 6]
  [2 7 8 9]]
```

```
# stack 1d array on left side of a 2d array
a = np.array([[1],[2])  # note 1 d array of 1 d array's
b = np.array([[4,5,6],[7,8,9]])
x = np.hstack([a,b])
```

```
[[1 4 5 6]
 [2 7 8 9]]
```

Split 1d array into a 2d array

We can split the above 1d array into a 2 d array using the numpy **split** function

a = np.array([1, 2]) # [1,2]
a = np.split(a,2) # [[1],[2]]

The 2 specifies the index slice to split on [0:2]

We get a 2 dimensional array with values [1] and [2]

```
[array([1]), array([2])]
```

We can now use hstack more conveniently where a is **[[1],[2]]** using the split function on [1,2]

b = np.array([[4,5,6],[7,8,9]])
x = np.hstack([a,b]) # a = [[1],[2]]

```
[[1 4 5 6]
[2 7 8 9]]
```

Numpy mathematical operations:

We will use the following 2 dimensional arrays:

a = np.array([[1,2],[3,4]])

b = np.array([[5,6],[7,8]])

You may use the mathematical operators **+, -, *, /** or the mathematical names add, subtract, multiply or divide.

add two numpy arrays

c = a + b
c = np.add(a,b)

```
[[6,8][10,12]]
```

Note: If you add a list to a numpy array you get a numpy array.

c = a + [[5,6][7,8]])

```
[[6,8][10,12]]
```

subtract 2 numpy arrays

c = a - b
c = np.subtract(a,b)

```
[[-4,-4][-4,-4]]
```

multiply 2 numpy arrays (just multiplies each element not dot product)

c = a * b
c = np.multiply(a,b)

```
[[ 5 12] [21 32]]
```

divide 2 numpy arrays

c = a / b
c = np.divide(a,b)

```
[[0.2       0.33333333]
[0.42857143      0.5 ]]
```

square root a numpy array

c = np.sqrt(a)

```
[[1.        1.41421356]
[1.73205081        2. ]]
```

matrix multiply two arrays (dot product)

product = a[0] * b[0] + a[1] * b [1]

c = np.matmul(a,b)

c = a @ b

```
[[19 22][43 50]]
```

Take the **dot product** of two arrays (similar to matrix multiply)

dot product = a[0] * b[0] + a[1] * b [1]

c = a.dot(b)

c = np.dot(a,b)

```
[[19 22][43 50]]
```

matmul and **dot** are similar but differ in operation for matrices greater than 2 dimensions

sum up all elements in a number 1 dimensional array

x = np.array([1,2,3,4,5])
s = np.sum(x)

```
15
```

sum up all the elements of a 2 dimensional numpy array

a = np.array[[1,2][3,4]]

s = np.sum(a)

```
10
```

Compute **sum** of each **column**

```
a = np.array([[1,2][3,4]])
s = np.sum(a, axis=0))
```

```
[4 6]
```

axis = 0 means sum of each column per row along **x axis** (horizontal axis)

```
  1       2
+ 3     +4
-----    ----
  4       6
```

Compute sum of each **row**

```
a = np.array([[1,2][3,4]])
s = np.sum(a, axis=1))
```

```
[3 7]
```

axis = 1 means sum all columns in a row along **y axis** (vertical axes)

1 + 2 = 3   3 + 4 = 7

transpose a 2 dimensional array [[1,2],[3,4]]

```
a = np.array([[1,2][3,4]])
c = a.T
```

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

inverse of a matrix [[1,2],[3,4]]

```
a = np.array([[1,2][3,4]])
c = np.linalg.inv(a)
```

```
[[-2.   1. ][ 1.5 -0.5]]
```

Note: **linalg** is a sub module of numpy that provides additional mathematical algorithms

add a column of 1's to an array [1,2,3,4,5]

```
x = [1,2,3,4,5]
y = np.ones(5)
c = np.c_[ x,y ]
```

```
[[1. 1.] [2. 1.][3. 1.] [4. 1.][5. 1.]]
```

Numpy statistic operations

a = np.array([1, 3, 2]) # [1 3 2]

function	Description	Example using	result
Min	Minimum value of array	**a.min()**	1
Max	Maximum value of array	**a.max()**	3
argmin	Index of minimum value	**a.argmin()**	0
argmax	Index of maximum value	**a.argmax()**	1
Mean	mean value of array	**a.mean()**	2
median	median value of array	**np.median(a)**	2
Std	Standard deviation	**a.std()**	.82915619758885995

Median of 2 dimensional array
(median of all elements in array)

```
b = np.array([[1, 2, 3], [5, 6, 1]])
np.median(b)
```

```
2.5
```

Median of 2 dimensional array along x axis

b = np.array([[1, 2, 3], [5, 6, 1]])
np.median(b, axis=0)

```
[3.  4. 2.]
```

median (1 + 5)/2 = 3
median (2 + 6) / 2 = 4
median (3 + 1) / 2 = 2

Median of 2 dimensional array along y axis

b = np.array([[1, 2, 3], [5, 6, 1]])
np.median(b, axis=1)

```
[2.  5.]
```

median (1 + 2 + 3) = 2
median (5 + 6 + 1) = 5

Homework to do

1. Add a 1 dimensional numpy array to a 2 dimensional numpy array. Use **arrange** to make the 1 dimensional array and **linspace** to make the 2 dimensional array.
2. Concatenate two 2 dimensional numpy array together and then take the square root of each element in the result
3. **vstack** a 3 element numpy 1 dimensional array with a 2 by 3 element numpy array then **hstack** the 1 by 3 element 1 dimensional array to the result. Use **split** on the 1 dimensional array.
4. Matrix multiply the inverse of a 3 by 3 element 2numbybarray with the transpose of a 3 x 3 element numpy array
5. Create a 3x3 numpy array with mean 0 and standard deviation 5, then calculate the median using the x axis then the y axis.

Put all your answers in a file called numpyhomework.py

END