Introduction to numpy:

Package for scientic computing with Python

Numerical Python, or "Numpy" for short, is a foundational package on which many of the most common data science packages are built. Numpy provides us with high performance multi-dimensional arrays which we can use as vectors or matrices. The key features of numpy are:

- **ndarrays:** n-dimensional arrays of the same data type which are fast and space-efficient.There are a number of built-in methods for ndarrays which allow for rapid processing of data without using loops (e.g., compute the mean).
- **Broadcasting:** a useful tool which defines implicit behavior between multi-dimensional arrays of different sizes.
- **Vectorization:** enables numeric operations on ndarrays.
- **Input/Output:** simplifies reading and writing of data from/to file.

Getting started with ndarray

The N-Dimensional Array (ndarray)

An ndarray is a (usually fixed-size) multidimensional container of items of the same type and size. The number of dimensions and items in an array is defined by its shape, which is a tuple of N positive integers that specify the sizes of each dimension. The type of items in the array is specified by a separate data-type object (dtype), one of which is associated with each ndarray. As with other container objects in Python, the contents of an ndarray can be accessed and modified by indexing or slicing the array (using, for example, N integers), and via the methods and attributes of the ndarray.

How to create Rank 1 numpy arrays:

```
import numpy as np
an_array = np.array([3, 33, 333])
type(an_array)
```

```
    numpy.ndarray
```

ndarray.shape

This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

```
# test the shape of the array we just created, it should have just one dimension (Rank 1)
an_array.shape
```

```
(3,)
```

```
# because this is a 1-rank array, we need only one index to accesss each element
an_array[0], an_array[1], an_array[2]
```

```
(3, 33, 333)
```

```
# ndarray are mutable
an_array[0]= 0
an_array
```

```
array([  0,  33, 333])
```

How to create a Rank 2 numpy array:

A rank 2 ndarray is one with two dimensions. Notice the format below of [[row] , [row]]. 2 dimensional arrays are great for representing matrices which are often useful in data science.

```
another = np.array([[11,12,13],[21,22,23]]) # Create a rank 2 array
print("Rank2 ndarray:", another) # print the array
print("The shape is 2 rows, 3 columns: ", another.shape) # rows x columns
print("Accessing elements [0,0], [0,1], and [1,0] of the ndarray: ", another[0, 0], ", ",a
```

```
Rank2 ndarray: [[11 12 13]
 [21 22 23]]
The shape is 2 rows, 3 columns:  (2, 3)
Accessing elements [0,0], [0,1], and [1,0] of the ndarray:  11 ,  12 ,  21
```

There are many way to create numpy arrays:

Here we create a number of different size arrays with different shapes and different pre-filled values. numpy has a number of built in methods which help us quickly and easily create multidimensional arrays.

```
import numpy as np
# create a 2x2 array of zeros
ex1 = np.zeros((2,2))
ex1
```

```
array([[0., 0.],
       [0., 0.]])
```

```
# create a 2x2 array filled with 9.0
ex2 = np.full((2,2), 9.0)
ex2
```

```
array([[9., 9.],
       [9., 9.]])
```

```
c= np.array([1,2,3,4], ndmin = 1)
print(c)
```

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

```
# minimum dimension
c= np.array([1,2,3,4], ndmin = 2)
print(c)
```

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

```
c= np.array([1,2,3,4], ndmin = 3)
print(c)
```

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

```
# create a 4x4 matrix with the diagonal 1s and the others 0
ex3 = np.eye(4,4)
print(ex3)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

```
# create an array of ones
ex4 = np.ones((1,2))
ex4
```

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

```
ex4 = np.ones((1,2))
ex4
print(ex4)
```

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

```
# notice that the above ndarray (ex4) is actually rank 2, it is a 1x2 array
print(ex4.shape)
```

```
(1, 2)
```

```
# which means we need to use two indexes to access an element
print(ex2[0,0])
```

```
9.0
```

```
# create an array of random floats between 0 and 1
ex5 = np.random.random((2,2))
ex5
```

```
array([[0.35314462, 0.69522466],
       [0.74727481, 0.94576608]])
```

```
type(ex5)
```

```
numpy.ndarray
```

```
a = np.random.rand(5)
a, type(a)
```

```
(array([0.51975167, 0.85761965, 0.80083676, 0.96815404, 0.23821035]),
 numpy.ndarray)
```

```
a_int = np.random.randint(0,10,5) # creat total 5 random integer from 0 to 10
a_int, type(a_int)
```

```
(array([5, 8, 4, 3, 7]), numpy.ndarray)
```

Resize the ndarray

```
# this resizes the ndarray
import numpy as np
Re=np.array([[1,2,3],[4,5,6]])
Re.shape , Re
```

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

```
Re.shape=(3,2)
Re
```

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

Slice indexing:

Similar to the use of slice indexing with lists and strings, we can use slice indexing to pull out sub-regions of ndarrays.

```
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
x[1:7:2]

     array([1, 3, 5])
```

```
x[-2:10]

     array([8, 9])
```

```
x[-3:3:-1]

     array([7, 6, 5, 4])
```

```
# Rank 2 array of shape (3, 4)
an_array = np.array([[11,12,13,14], [21,22,23,24], [31,32,33,34]])
an_array

     array([[11, 12, 13, 14],
            [21, 22, 23, 24],
            [31, 32, 33, 34]])
```

Use array slicing to get a subarray consisting of the first 2 rows x 2 columns. [ ]↳ 9 cells hidden

numpy.itemsize

This array attribute returns the length of each element of array in bytes.

```
# dtype of array is int8 (1 byte)
s = np.array([1,2,3,4,5], dtype=np.int8)
s.itemsize

     1
```

```
# dtype of array is float32 (4 byte)
s1 = np.array (([1,2,3,4],[11,12,13,14]), dtype =np.float32)
s1.itemsize

     4
```

```
x = np.array([1,2,3,4,5], dtype = np.int64)
x.itemsize

     8
```

Arithmetic Array Operations:

```
x = np.array([[111,112],[121,122]], dtype=np.int)
y = np.array([[211.1,212.1],[221.1,222.1]], dtype=np.float64)
x+y
```

```
     array([[322.1, 324.1],
            [342.1, 344.1]])
```

```
# numpy add function
np.add(x, y)
```

```
     array([[322.1, 324.1],
            [342.1, 344.1]])
```

```
x-y
```

```
     array([[-100.1, -100.1],
            [-100.1, -100.1]])
```

```
# numpy subtract function
np.subtract(x, y)
```

```
     array([[-100.1, -100.1],
            [-100.1, -100.1]])
```

```
x*y
```

```
     array([[23432.1, 23755.2],
            [26753.1, 27096.2]])
```

```
np.multiply(x, y)
```

```
     array([[23432.1, 23755.2],
            [26753.1, 27096.2]])
```

```
x/y
```

```
     array([[0.52581715, 0.52805281],
            [0.54726368, 0.54930212]])
```

```
np.divide(x, y)
```

```
     array([[0.52581715, 0.52805281],
            [0.54726368, 0.54930212]])
```

```
# exponent (e ** x)
np.exp(x)
```

```
     array([[1.60948707e+48, 4.37503945e+48],
            [3.54513118e+52, 9.63666567e+52]])
```

```
# square root
np.sqrt(x)

     array([[10.53565375, 10.58300524],
            [11.        , 11.04536102]])
```

Basic Statistical Operations:

[ ]↳ 7 cells hidden

Broadcasting:

```
import numpy as np
start = np.zeros((4,3))
print(start)

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

```
# create a rank 1 ndarray with 3 values
add_rows = np.array([1, 0, 2])
print(add_rows)

     [1 0 2]
```

```
y = start + add_rows # add to each row of 'start' using broadcasting
print(y)

     [[1. 0. 2.]
      [1. 0. 2.]
      [1. 0. 2.]
      [1. 0. 2.]]
```

```
# create an ndarray which is 4 x 1 to broadcast across columns
add_cols = np.array([[0,1,2,4]])
add_cols_T = add_cols.T
print(add_cols_T)

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

```
add_cols.shape , add_cols_T.shape
```

```
((1, 4), (4, 1))
```

```
# add to each column of 'start' using broadcasting
y = start + add_cols_T
print(y)
```

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

```
# this will just broadcast in both dimensions
add_scalar = np.array([1])
add_scalar
```

```
array([1])
```

```
print(start+add_scalar)
```

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

```
# create our 3x4 matrix
arrA = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print(arrA)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
# create our 4x1 array
arrB = [1,1,1,1]
print(arrB)
```

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

```
# add the two together using broadcasting
print(arrA + arrB)
```

```
[[ 2  3  4  5]
 [ 6  7  8  9]
 [10 11 12 13]]
```

```
arrA
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

```
# Sum by column
arrA.sum(axis=0)

     array([15, 18, 21, 24])
```

```
# Sum by row
arrA.sum(axis=1)

     array([10, 26, 42])
```

Dot Product and Cross Product :

```
# determine the dot product of two matrices
A = np.array([1,2,3])
B = np.array([6,2,3])
print(A.dot(B))

     19
```

```
print(np.dot(A,B))

     19
```

```
#  product of two matrices
A = np.array([21,28,3])
B = np.array([16,2,3])
print(A*(B))

     [336  56   9]
```

```
a1d = np.array([16,2,3])
b1d = np.array([5,2,3])
print(np.cross(a1d,b1d))

     [  0 -33  22]
```

```
print(np.dot(a1d, b1d))

     93
```

```
# determine the cross product of two vectors
a1d = np.array([9 , 9, 1 ])
b1d = np.array([4, 3, 3])
print(np.cross(a1d, b1d))

     [ 24 -23  -9]
```

```
print(np.dot(A, a1d))
```

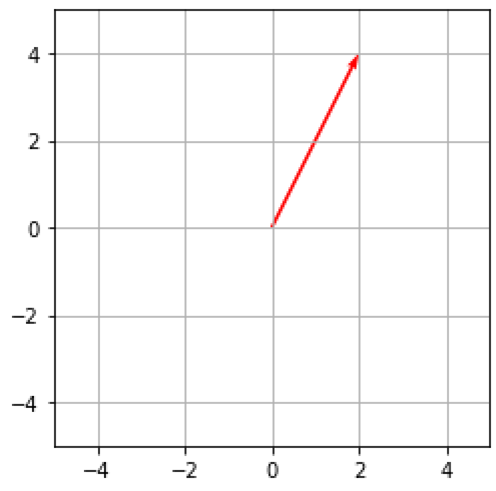
```
444
```

```
Mag_A = np.linalg.norm(A)
Mag_B = np.linalg.norm(B)
print ("Magnitude of vector A is : " , Mag_A)
print ("Magnitude of vector B is : " , Mag_B)
```

```
Magnitude of vector A is :  35.12833614050059
Magnitude of vector B is :  16.401219466856727
```

Draw a vector using Matplotlib :

```
import matplotlib.pyplot as plt
import numpy as np
X = np.array((0))
Y= np.array((0))
U = np.array((2))
V = np.array((4))
fig, ax = plt.subplots()
q = ax.quiver(X, Y, U, V,color = 'r', units='xy' ,scale=1)
plt.grid()
ax.set_aspect('equal')
plt.xlim(-5,5)
plt.ylim(-5,5)
plt.show()
```


4
2
0
−2
−4
−4
−2
0
2
4