Usman NumPy

December 26, 2022

[2]:
```
import numpy as np #import NumPy Module
```

1 Introduction: NumPy

- NumPy is a Python module that provides many convenience methods and also better performance. NumPy provides a core library for scientific computing in Python, with performant multidimensional arrays and good vectorized math functions, along with support for linear algebra and random numbers.
- The NumPy package provides the ndarray object that encapsulates multidimensional arrays of homogeneous data types. Many ndarray operations are performed in compiled code in order to improveperformance.
- What are NumPy Arrays?
- An array is a set of consecutive memory locations used to store data. Each item in the array is called an element. The number of elements in an array is called the dimension of the array. A typical array declaration is shown here: arr1 = np.array([1,2,3,4,5])
- Numpy is zero cost ; Open Source.'

2 NumPy is used for..!!

1.Arithmetic Operation

2.Searching,sorting,counting

3.Mathematical operation

4.Linear algebra Operation

5.Statics operation

6.Broadcasting

7.Matrix operations

8.Sacking Operation

9.Copy and viewing arrays

10.Bitwise Operation

11.Statistics operations

3 Python List Vs NumPy Arrays

4 Python List

1. No Support for Vectorized operations.
2. No fixed type elements.
3. For loops not efficient.
4. Lists are built-in data structures.
5. List can hold elemetns of different types. **list = [“ball”,1234,“cat”] –» [str,int,str]
6. List is flexible, we add new element and expands.

5 NumPy Arrays

1. Supports Vectorized operations (Addition, multiplications)
2. Fixed data type
3. More efficient
4. Code is cleaner and has more advanced built-in-functions.
5. Arrays must be imported. **Import numpy as np**
6. Arrays which can hold only data of the same data type for eg. only integers or only strings. **array = np.array([1,2,2,2,3,4]) –» [int,int]
7. Numpy Array consume less memory than List, Fewer loops.
8. Compution is much faster than list & Advance Mathematical functions..
9. Once we define the dimension of an array, can not expand it anymore , limited.
10. Same tasks with in fewer lines.

6 Applications of NumPy

1.Machine Learning and computional packages.

2.Essential library for scientific projects.

7 Common things between Python list and NumPy Array.

1.Python list and NumPy arrays have same Syntax.

2.There elements are ordered, Mutable and are able to store duplicate items.

3.They also allow indexing, slicing and iterating.

8 some well known library use NumPy.

Scipy,Scikit Learn, Matplotlib, Pandas, Statsmodel.

8.1 comparision between Numpy Array and Python for loop (Computation time)

[ ]:
```
import numpy as np
import time
import math
import numpy as np

iter = 1000000

x = np.zeros((iter,1))
v = np.random.randn(iter,1)

before = time.time()

for i in range(iter):
    x[i] = math.exp(v[i])
after = time.time()
print(x)
print("Regular for loop= " + str((after-before)*1000) + "ms")
print('\n')
time1 = (after-before)*1000

before = time.time()
x = np.exp(v)
after = time.time()
print(x)
print("Numpy operation= " + str((after-before)*1000) + "ms")
time2 = (after-before)*1000
print('\n')
print("Numpy is "+ str(round(time1/time2,2)) + " times faster than for loop in
 Python.")
```

```
[[0.12141492]
 [0.67375589]
 [1.91506148]
 …
 [2.54321501]
 [2.42290436]
 [0.39557539]]
Regular for loop= 461.8847370147705ms


[[0.12141492]
 [0.67375589]
 [1.91506148]
```

```
 …
 [2.54321501]
 [2.42290436]
 [0.39557539]]
Numpy operation= 6.728410720825195ms


Numpy is 68.65 times faster than for loop in Python.
```

9 Array Types and Conversions Between Types

[ ]:
```
array = np.array([2,3,4,5,10])
array
```

[ ]:
```
array([ 2,  3,  4,  5, 10])
```

[ ]:
```
array[0]
```

[ ]:
```
2
```

[ ]:
```
#adding new element in index one
array[1] = 100
array
```

[ ]:
```
array([  2, 100,   4,   5,  10])
```

[ ]:
```
#adding nwew float point in index one.
array[1]=100.1
array
```

[ ]:
```
array([  2, 100,   4,   5,  10])
```

10 Numpy Array Type and Creating Numpy Array

**Array types and onversion between types 1. over come unwanted overflow, if don't define dtype. 2. memory consumption and precision which especially irrelent Machine learning algorithm.

[ ]:
```
array.dtype
```

[ ]:
```
dtype('int32')
```

[ ]:
```
array_dsize = np.array(array, dtype=np.int8)
```

[ ]:
```
array_dsize
```

[ ]:
```
array([ 2, 100,  4,  5, 10], dtype=int8)
```

[ ]:
```
array.nbytes
```

[ ]: 20

[ ]:
```
#this take 4 time less memory than array.nbytes
array_dsize.nbytes
```

[ ]: 5

[ ]:
```
array_dtype = np.array([12,13,14,15,100], dtype = np.int8)
array_dtype
```

[ ]: array([12, 13, 14, 15, 100], dtype=int8)

[ ]:
```
array_float = np.array([1.3,4.5,8.8,10.12,90.99])
array_float
```

[ ]: array([1.3 , 4.5 , 8.8 , 10.12, 90.99])

[ ]:
```
array_float.dtype
```

[ ]: dtype('float64')

11 Multidimensional Arrays

1. The NumPy array called ndarray is the central object of the NumPy package.
2. One-dimensional array can be thought of as a vetor, Two-dimensional array as a matrix and three-dimensional arrays as a tensor, if you more than three-dimensional its simply with NumPy building functions.

[ ]:
```
#two-dimensional array
twd_array=np.array([[1,2,3,4],[5,6,7,6]])
twd_array
```

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

[ ]:
```
twd_array[0,0]
```

[ ]: 1

[ ]:
```
#check dimensional of array
twd_array.ndim
```

[ ]: 2

[ ]:
```
#three diminsional array
thd_array = np.array([[[1,2,3],[2,5,6],[7,8,9]]])
thd_array
```

[ ]:
[ ]: array([[[1, 2,
3], [2, 5,
6],
[7, 8, 9]]])

```
[ ]: thd_array.ndim
```

[ ]: 3

```
[ ]: thd_array[0,0,2]
```

[ ]: 3

12 Creating arrays from lists and other Python Structure

```
[ ]: lst = [1,2,4,5,6,7,8,9]
```

```
[ ]: array_lst = np.array(lst)
array_lst
```

[ ]: array([1, 2, 4, 5, 6, 7, 8, 9])

```
[ ]: nums = [1,4,5.999,-1.23,-4,99.9999]
```

```
[ ]: array_nums = np.array(nums)
array_nums
```

[ ]: array([1. , 4. , 5.999 , -1.23 , -4. , 99.9999])

```
[ ]: array_nums.dtype
```

[ ]: dtype('float64')

```
[ ]: third_list = ['apple',1,2,5,'cat','dog']
```

```
[ ]: array_third = np.array(third_list)
array_third
```

[ ]: array(['apple', '1', '2', '5', 'cat', 'dog'], dtype='<U11')

```
[ ]: multi_dim_list = [[[1,2,3],[3,4,5],[9,8,10]]]
```

```
[ ]: array_multi_dim = np.array(multi_dim_list)
array_multi_dim
```

array([[[1, 2, 3],
[3, 4, 5],

[ ]:
```
           [ 9,  8, 10]]])
```

[ ]:
```
tuple_nums = (1,2,4,5,6,7,100)
```

[ ]:
```
array_tuple = np.array(tuple_nums)
array_tuple
```

[ ]:
```
array([  1,   2,   4,   5,   6,   7, 100])
```

[ ]:
```
array_tuple.dtype
```

[ ]:
```
dtype('int32')
```

13 Intrisic NumPy array Creation

[ ]:
```
array = np.arange(20)
array
```

[ ]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

[ ]:
```
array = np.arange(10,20)
array
```

[ ]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

[ ]:
```
array = np.arange(10,41,2)
array
```

[ ]:
```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38,
40])
```

[ ]:
```
array = np.linspace(10,30)
array
```

[ ]:
```
array([10.  , 10.40816327, 10.81632653, 11.2244898 , 11.63265306,
       12.04081633, 12.44897959, 12.85714286, 13.26530612, 13.67346939,
       14.08163265, 14.48979592, 14.89795918, 15.30612245, 15.71428571,
       16.12244898, 16.53061224, 16.93877551, 17.34693878, 17.75510204,
       18.16326531, 18.57142857, 18.97959184, 19.3877551 , 19.79591837,
       20.20408163, 20.6122449 , 21.02040816, 21.42857143, 21.83673469,
       22.24489796, 22.65306122, 23.06122449, 23.46938776, 23.87755102,
       24.28571429, 24.69387755, 25.10204082, 25.51020408, 25.91836735,
           26.32653061, 26.73469388, 27.14285714, 27.55102041,
           27.95918367,
        28.36734694, 28.7755102 , 29.18367347, 29.59183673, 30.    ])
```

[ ]:
```
array = np.linspace(10,30,5)
array
```

[ ]:
```
array([10., 15., 20., 25., 30.])
```

[ ]:
```
array_rand = np.random.rand(5,5)
array_rand
```

[ ]:
```
array([[0.9542999 , 0.71239093, 0.88888455, 0.90488098,
0.74325562], [0.33182353, 0.94763146, 0.86001134, 0.70082372,
                                                  0.30725493],
       [0.0575304 , 0.0550632 , 0.30162191, 0.6966861 ,
       0.28773108],
       [0.70940368, 0.86810313, 0.04617945, 0.79321346,
       0.47748578],
       [0.24548673, 0.57844721, 0.32001166, 0.72051759,
       0.73719348]])
```

[ ]:
```
array_randint = np.random.randint(0,100,20)
array_randint
```

[ ]:
```
array([12, 16, 51, 24, 40, 36, 83, 0, 91, 60, 14, 59, 18, 85, 31,
       52, 59, 69, 16, 47])
```

14 Creating array filled with constant values

[ ]:
```
array = np.zeros(4)
array
```

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

[ ]:
```
array = np.zeros((4,5))
array
```

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

[ ]:
```
array = np.ones(5)
array
```

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

[ ]:
```
array = np.ones((5,6))
array
```

```
[ ]:
```

```
[ ]: 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.]])
```

[ ]:
```
array = np.ones((5,6), dtype=int)
array
```

[ ]:
```
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]])
```

[ ]:
```
array = np.empty(10,dtype=int)
array.fill(10)
array
```

[ ]:
```
array([10, 10, 10, 10, 10, 10, 10, 10, 10, 10])
```

[ ]:
```
array = np.full(5,10)
array
```

[ ]:
```
array([10, 10, 10, 10, 10])
```

[ ]:
```
array = np.full((4,5),8)
array
```

[ ]:
```
array([[8, 8, 8, 8,
        8], [8, 8, 8, 8,
        8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8]])
```

15 Finding the Size and shape of an array

[ ]:
```
array_first = np.arange(20)
array_first
```

[ ]:
```
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19])
```

[ ]:
```
array_second = np.linspace((1,2),(10,20),10)
array_second
```

[ ]:
```
array([[ 1.,
        2.], [ 2.,
        4.],
       [ 3., 6.],
       [ 4., 8.],
       [ 5., 10.],
       [ 6., 12.],
```

```
       [ 7., 14.],
       [ 8., 16.],
       [ 9., 18.],
       [10., 20.]])
```

```
[ ]: array_third = np.full((2,2,2),10)
     array_third
```

```
[ ]: array([[[10, 10],
             [10, 10]],

            [[10, 10],
             [10, 10]]])
```

```
[ ]: array_first.shape
```

```
[ ]: (20,)
```

```
[ ]: np.shape(array_second)
```

```
[ ]: (10, 2)
```

```
[ ]: np.shape(array_third)
```

```
[ ]: (2, 2, 2)
```

```
[ ]: np.size(array_first)
```

```
[ ]: 20
```

16 Manipulate NumPy Arrays

- Adding, Removing and Sorting elements

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

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

```
[ ]: new_array_insert = np.insert(array,1,8)
     new_array_insert
```

```
[ ]: array([1, 8, 2, 3, 4, 5, 6])
```

```
[ ]: new_array_append = np.append(array,10)
     new_array_append
```

```
[ ]: array([ 1, 2, 3, 4, 5, 6, 10])
```

```
[ ]: new_array_delete = np.delete(array,2)
     new_array_delete
```

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

```
[ ]: array = np.random.randint(0,10,20)
     array
```

```
[ ]: array([9, 8, 0, 6, 7, 4, 7, 3, 8, 0, 2, 2, 7, 6, 5, 5, 9, 4, 6,
9])
```

```
[ ]: print(np.sort(array))
```

```
[0 0 2 2 3 4 4 5 5 6 6 6 7 7 7 8 8 9 9 9]
```

```
[ ]: #sort 2-dimensional array
     array = np.array([[2,9,4,3,0],[9,4,10,11,16]])
     print(np.sort(array))
```

```
[[ 0  2  3  4  9]
 [ 4  9 10 11 16]]
```

```
[ ]: array = np.array(['red','blues','green','orange','white','pink'])
```

```
[ ]: print(np.sort(array))
```

```
['blues' 'green' 'orange' 'pink' 'red' 'white']
```

**Copies and views of array

```
[ ]: emp_id = np.array([111,112,113,222,333])
     emp_id
```

```
[ ]: array([111, 112, 113, 222, 333])
```

```
[ ]: emp_id_reg = emp_id
     print("id of employee", id(emp_id))
     print("id of emp_id_reg", id(emp_id_reg))
```

```
id of employee
140299740095632 id of
emp_id_reg 140299740095632
```

```
[ ]: emp_id_reg[1]=333
     print(emp_id )
     print(emp_id_reg)
```

```
[111 333 113 222 333]
[111 333 113 222 333]
```

```
[ ]: emp_id_cp = emp_id.copy()
```

```
[ ]: print(emp_id_cp)
```

```
[111 333 113 222 333]
```

```
[ ]: emp_id[0]=2112
     print('original:', emp_id)
     print('copy:', emp_id_cp)
```

```
original: [2112 333 113 222 333]
copy: [111 333 113 222 333]
```

17 Reshaping Arrays

[ ]:
```
array = np.arange(1,13)
array
```

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

[ ]:
```
reshape_array = np.reshape(array,(3,4))
reshape_array
```

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

[ ]:
```
reshape_array = np.reshape(array,(6,2))
reshape_array
```

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

[ ]:
```
reshape_array = np.reshape(array,(4,5)) reshape_array
#cannot reshape array of size 12 into shape (4,5)
```

[ ]:
```
reshape_array = np.reshape(array,(3,2,2))
print(reshape_array)
print("Dimensions of array:", reshape_array.ndim)
```

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

 [[ 5  6]
  [ 7  8]]

 [[ 9 10]
  [11 12]]]
Dimensions of array: 3
```

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

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

```
[ ]: reshape_array = np.reshape(array,-1)
     reshape_array
```

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

**Array Flat and rav fucntion.(see NumPy doc)

**Indexing and Slicing

```
[ ]: array = np.reshape(np.arange(12),(3,4))
     array
```

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

```
[ ]: array[1][1]
```

```
[ ]: 5
```

```
[ ]: array[1]
```

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

```
[ ]: array = np.reshape(np.arange(3*4*5),(3,4,5))
     array
```

```
[ ]: array([[[ 0, 1, 2, 3, 4],
             [ 5, 6, 7, 8, 9],
             [10, 11, 12, 13, 14],
             [15, 16, 17, 18, 19]],

            [[20, 21, 22, 23, 24],
             [25, 26, 27, 28, 29],
             [30, 31, 32, 33, 34],
             [35, 36, 37, 38, 39]],

            [[40, 41, 42, 43, 44],
             [45, 46, 47, 48, 49],
             [50, 51, 52, 53, 54],
             [55, 56, 57, 58, 59]]])
```

```
[ ]: array[0,1,2]
```

```
[ ]: 7
```

```
[ ]: array[2,-1,-1]
```

[ ]: 59

[ ]:
```
array = np.arange(10)
array
```

[ ]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

[ ]:
```
array[3:6]
```

[ ]: array([3, 4, 5])

[ ]:
```
array[:5]
```

[ ]: array([0, 1, 2, 3, 4])

[ ]:
```
array[-3:]
```

[ ]: array([7, 8, 9])

[ ]:
```
array[::2]
```

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

18 Joining and Splitting Arrays

1.concatenate

2.stack

3.hstack

4.vstack _____ # Splitting Array

one major draw back , It will split arrays only when the number of elements is divisible by the number of splits. so the resulting arrays needs to have the same shape.

1.split

2.array_split

3.hsplit

4vsplit

[ ]:
```
array = np.arange(1,10)
array1 = np.arange(11,20)
print('Array:',array)
print('Array1:',array1)
```

```
A r r a y : [ 1 2 3 4 5 6 7 8 9 ]
Array1: [11 12 13 14 15 16 17 18 19]
```

[ ]:
```
con_arr=np.concatenate((array,array1))
con_arr
```

[ ]: array([1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18,

```
       19])
```

[ ]:
```
array1 = np.array([[1,2,6,7,8],[12,13,14,15,20]])
array = np.array([[1,2,3,4,5],[16,17,18,19,20]])
print(array)
print(array1)
```

```
[  [  1  2  3  4  5
 [16 17 18 19 20]]
[[ 1 2 6 7 8]
 [12 13 14 15 20]]
```

[ ]:
```
con_array = np.concatenate((array1,array), axis=1)
con_array
```

[ ]:
```
array([[ 1,  2,  6,  7,  8,  1,  2,  3,  4,  5],
       [12, 13, 14, 15, 20, 16, 17, 18, 19, 20]])
```

[ ]:
```
array1 = np.array([[1,2,6,7,8],[12,13,14,15,20]])
array = np.array([[1,2,3,4,5],[16,17,18,19,20]])
str_arr = np.stack((array1,array))
str_arr
```

[ ]:
```
array([[[ 1,  2,  6,  7,  8],
        [12, 13, 14, 15, 20]],

       [[ 1,  2,  3,  4,  5],
        [16, 17, 18, 19, 20]]])
```

[ ]:
```
array = np.arange(1,10)
array1 = np.arange(11,20)
str_arr = np.stack((array1,array))
str_arr
```

[ ]:
```
array([[11, 12, 13, 14, 15, 16, 17, 18, 19],
       [ 1,  2,  3,  4,  5,  6,  7,  8,  9]])
```

[ ]:
```
array1 = np.array([[1,2,6,7,8],[12,13,14,15,20]])
array = np.array([[1,2,3,4,5],[16,17,18,19,20]])
str_arr = np.vstack((array1,array))
str_arr
```

[ ]:
```
array([[ 1,  2,  6,  7,  8],
       [12, 13, 14, 15, 20],
       [ 1,  2,  3,  4,  5],
       [16, 17, 18, 19, 20]])
```

[ ]:
```
array = np.arange(1,10)
array1 = np.arange(11,20)
str_arr = np.vstack((array1,array))
str_arr
```

```
[ ]: array([[11, 12, 13, 14, 15, 16, 17, 18, 19],
            [ 1,  2,  3,  4,  5,  6,  7,  8,  9]])
```

```
[ ]: array = np.arange(1,13)
     sp_array = np.array_split(array,4)
     sp_array
```

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

```
[ ]: print(sp_array[1])
```

```
     [4 5 6]
```

```
[ ]: array = np.arange(1,13)
     sp_array = np.array_split(array,8)
     sp_array
```

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

```
[ ]: array = np.array([[1,2,3,4,5],[3,5,6,7,8,]])
     hs_array = np.hsplit(array,5)
     hs_array
```

```
[ ]: [array([[1],
            [3]]),
      array([[2],
            [5]]),
      array([[3],
            [6]]),
      array([[4],
            [7]]),
      array([[5],
            [8]])]
```

```
[ ]: array = np.array([[1,2,3,4,5],[3,5,6,7,8,]])
     vs_array = np.vsplit(array,2)
     vs_array
```

```
[ ]: [array([[1, 2, 3, 4, 5]]), array([[3, 5, 6, 7, 8]])]
```

[ ]:
```
import numpy as np

arr1 = np.array([2,4,6,8,10])
arr2 = np.array([1,3,5,7,9])

arr3 = np.concatenate((arr1, arr2), axis=0)
arr3
```

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

19 Function and Operations

**Arthmetic Operations and Functions

Addition Division

Subtraction Exponentiation

Multiplication Specific functions to perform them

Vectorization -> Operation can be executed in parallel on multiple elements of the array.

Benifits

*Higher performance code

*Less verbose code

*Better maintainability

[ ]:
```
arr_one = np.arange(1,11)
arr_two = np.arange(41,51)
print("arr_three_add:\n", arr_one + arr_two)
print("arr_three_sub:\n", arr_one - arr_two)
print("arr_three_div:\n", arr_one / arr_two)
print("arr_three_mul:\n", arr_one *arr_two)
```

```
arr_three_add:
 [42 44 46 48 50 52 54 56 58 60]
arr_three_sub:
 [-40 -40 -40 -40 -40 -40 -40 -40 -40 -40]
arr_three_div:
   [0.02439024 0.04761905 0.06976744 0.09090909 0.11111111
                                                0.13043478
 0.14893617 0.16666667 0.18367347 0.2   ]
arr_three_mul:
 [ 41  84 129 176 225 276 329 384 441 500]
```

[ ]:
```
arr_three = np.arange(2,12)
print(arr_one**arr_three)
```

```
    [          1           8          81        1024       15625
        279936     5764801   134217728 3486784401 1000000000000]
```

```
[ ]: arr_one*2
```

```
[ ]: array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

```
[ ]: np.add(arr_one,arr_two)
```

```
[ ]: array([42, 44, 46, 48, 50, 52, 54, 56, 58, 60])
```

```
[ ]: np.subtract(arr_three,arr_one)
```

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

```
[ ]: np.multiply(arr_one,arr_two)
```

```
[ ]: array([ 41,  84, 129, 176, 225, 276, 329, 384, 441, 500])
```

```
[ ]: np.mod(arr_two,arr_one)
```

```
[ ]: array([0, 0, 1, 0, 0, 4, 5, 0, 4, 0])
```

```
[ ]: np.power(arr_one,arr_three)
```

```
[ ]:                1,           8,         81,         1024,
array([
               15625,      279936,    5764801,   134217728,
          3486784401, 100000000000])
```

```
[ ]: np.sqrt(arr_one)
```

```
[ ]: array([1.    , 1.41421356, 1.73205081, 2.        ,
                                                2.23606798,
           2.44948974, 2.64575131, 2.82842712, 3.        ,
                                                3.16227766])
```

20 Broadcasting

*The term broadcasting describes how NumPy treats arrays with different shapes during arithmetic operations.

*Subject to certain constraints, the smaller array is broadcast across the larger array so that they have compatible shapes.

```
Dimesion are Compatible
```

If their axes on a one-by-one basis, they have either the same length or length of one.

```
[ ]: arr1 = np.arange(1,10).reshape(3,3)
     arr1
```

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

```
[ ]: arr2 = np.arange(1,4)
     arr2
```

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

```
[ ]: arr1 + arr2
```

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

```
[ ]: arr = np.arange(24).reshape(2,3,4)
     arr1 = np.arange(4)
```

```
[ ]: arr
```

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

            [[12, 13, 14, 15],
             [16, 17, 18, 19],
             [20, 21, 22, 23]]])
```

```
[ ]: arr-arr1
```

```
[ ]: array([[[ 0, 0, 0, 0],
             [ 4, 4, 4, 4],
             [ 8, 8, 8, 8]],
            [[12, 12, 12, 12],
             [16, 16, 16, 16],
             [20, 20, 20, 20]]]) 21
```

Aggregae Functions

```
[ ]: array = np.arange(10,100,10)
```

```
[ ]: print("sum of array:" ,array.sum())
```

```
sum of array: 450
```

```
[ ]: array1 = np.arange(10,110,10).reshape(2,5)
```

```
[ ]: print("sum of array1:",array1.sum())
```

```
sum of array1: 550
```

```
[ ]: array.sum(axis=0)
```

```
[ ]: 450
```

```
[ ]: array1.sum(axis=1)
```

```
[ ]: array([150, 400])
```

```
[ ]: array.prod()
```

```
[ ]: 362880000000000
```

```
[ ]: array.prod(axis=0)
```

```
[ ]: 362880000000000
```

```
[ ]: np.average(array)
```

```
[ ]: 50.0
```

```
[ ]: np.min(array)
```

```
[ ]: 10
```

```
[ ]: np.max(array)
```

```
[ ]: 90
```

```
[ ]: np.mean(array)
```

```
[ ]: 50.0
```

```
[ ]: np.std(array)
```

```
[ ]: 25.81988897471611
```

22 Unique items and Counts

```
[ ]: array = np.array([1,2,4,2,3,1,21,23,21,1,2,4,3])
     np.unique(array)
```

```
[ ]: array([ 1,  2,  3,  4, 21, 23])
```

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

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

```
[ ]: np.unique(array,axis=0)
```

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

```
[ ]: np.unique(array,axis=1) #unique column
```

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

```
[ ]: np.unique(array,return_index=True)
```

```
[ ]: (array([1, 2, 3, 4, 6]), array([ 0, 2, 3, 14, 13]))
```

```
[ ]: np.unique(array,return_counts=True)
```

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

```
[ ]: array = np.arange(12).reshape((3,4))
     array
```

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

```
[ ]: np.transpose(array)
```

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

```
[ ]: array = np.arange(6).reshape(3,2)
     array
```

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

```
[ ]: np.transpose(array)
```

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

```
[ ]: np.transpose(array,(1,0))
```

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

```
[ ]: array = np.arange(24).reshape(2,3,4)
     array
```

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

```
           [ 8,  9, 10, 11]],

          [[12, 13, 14, 15],
           [16, 17, 18, 19],
           [20, 21, 22, 23]]])
```

[ ]:
```
np.moveaxis(array,0,-1)
```

[ ]:
```
array([[[ 0, 12], [ 1, 13],
        [ 2, 14],
        [ 3, 15]],

       [[ 4, 16],
        [ 5, 17],
        [ 6, 18],
        [ 7, 19]],

       [[ 8, 20],
        [ 9, 21],
        [10, 22],
        [11, 23]]])
```

[ ]:
```
np.swapaxes(array,0,2)
```

[ ]:
```
array([[[ 0, 12], [
        4, 16],
        [ 8, 20]],

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

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

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

[ ]:
```
array = [10,2,3,4,1,6,7,5,9]
array
```

[ ]:
```
[10, 2, 3, 4, 1, 6, 7, 5, 9]
```

[ ]:
```
array[::-1]
```

[ ]:
```
[9, 5, 7, 6, 1, 4, 3, 2, 10]
```

```
[ ]: np.flip(array)
```

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

```
[ ]: array = np.arange(9).reshape(3,3)
     array
```

```
[ ]: array([[0, 1,
             2], [3, 4,
             5],
             [6, 7, 8]])
```

```
[ ]: np.flip(array)
```

```
[ ]: array([[8, 7,
             6], [5, 4,
             3],
             [2, 1, 0]])
```

```
[ ]: np.flip(array,1)
```

```
[ ]: array([[2, 1,
             0], [5, 4,
             3],
             [8, 7, 6]])
```

```
[ ]: array = np.arange(24).reshape(2,3,4)
     array
```

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

            [[12, 13, 14, 15],
             [16, 17, 18, 19],
             [20, 21, 22, 23]]])
```

```
[ ]: np.flip(array,1)
```

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

            [[20, 21, 22, 23],
             [16, 17, 18, 19],
             [12, 13, 14, 15]]])
```

```
[ ]: np.flip(array,2)
```

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

            [[15, 14, 13, 12],
             [19, 18, 17, 16],
             [23, 22, 21, 20]]])
```

23 very useful NumPy methods.

The method **np.zeros()** initializes an array with 0 values.

The method **np.ones()** initializes an array with 1 values.

The method **np.empty()** initializes an array with 0 values.

The method **np.arange()** provides a range of numbers:

The method **np.shape()** displays the shape of an object:

The method **np.reshape()** <= very useful!

The method **np.linspace()** <= useful in regression

The method **np.mean()** computes the mean of a set of numbers.

The method **np.std()** calculate Standard Deviation.

```
[ ]:
```