numpy

August 28, 2023

1 Numpy for Data Science

[1]:
```
import numpy as np
```

1.1 Array Attribute

1.1.1 1-D Array :

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

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

[3]:
```
type(a) # numpy.ndarray --> n dimantion array
```

[3]:
```
numpy.ndarray
```

[4]:
```
a.ndim
```

[4]:
```
1
```

[5]:
```
a.shape       # 5 column , no rows
```

[5]:
```
(5,)
```

1.1.2 2-D Array :

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

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

[7]:
```
b.shape           # 2 rows , 3 column
```

[7]:
```
(2, 3)
```

[8]:
```
b.dtype           # dtype --> data type
```

[8]:
```
dtype('int64')
```

[9]:
```
b.size            # Total element
```

[9]:
```
6
```

1.1.3 3-D Array :

[10]:
```
c = np.array([[[1,2,3],
               [4,5,6],
               [7,8,9]],

              [[11,12,13],    # imagine that 1-9 in up of 10-18
               [14,15,16],
               [17,19,19]]])
c
```

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

       [[11, 12, 13],
        [14, 15, 16],
        [17, 19, 19]]])
```

1.1.4 Memory check :

[11]:
```
a.nbytes , b.nbytes , c.nbytes
```

[11]:
```
(40, 48, 144)
```

1.1.5 Zeros & Ones :

[12]:
```
zeros_arr = np.zeros((3,3) , dtype=int)
zeros_arr
```

[12]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

[13]:
```
ones_arr = np.ones((4,4) , dtype=int)
ones_arr
```

[13]:
```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

1.1.6 Different values :

[14]:
```
arr_3 = np.full((4,4) , 12)
arr_3
```

[14]:
```
array([[12, 12, 12, 12],
       [12, 12, 12, 12],
       [12, 12, 12, 12],
       [12, 12, 12, 12]])
```

1.1.7 Identity matrix (I) :

[15]:
```
i_3 = np.identity(3, dtype=int)
i_3
```

[15]:
```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

1.1.8 Eye :

[16]:
```
eye = np.eye(3 , dtype=int)
eye
```

[16]:
```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

[17]:
```
eye = np.eye(4, dtype= int , k = 1)
eye           # k = 1 --> mail row will up by 1
```

[17]:
```
array([[0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1],
       [0, 0, 0, 0]])
```

1.1.9 Arrange :

[18]:
```
arrange = np.arange(0,100,10)        # like a python range() function
arrange
```

[18]:
```
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

1.1.10 Linspace :

[19]:
```
linspace = np.linspace(1,100,20)      # (start , end , how many values)
linspace
```

[19]:
```
array([  1.        ,   6.21052632,  11.42105263,  16.63157895,
        21.84210526,  27.05263158,  32.26315789,  37.47368421,
        42.68421053,  47.89473684,  53.10526316,  58.31578947,
        63.52631579,  68.73684211,  73.94736842,  79.15789474,
        84.36842105,  89.57894737,  94.78947368, 100.        ])
```

[20]:
```
linspace = np.linspace(1,100,20 , dtype=int)      # (start , end , how many␣
↪values)
linspace
```

[20]:
```
array([  1,   6,  11,  16,  21,  27,  32,  37,  42,  47,  53,  58,  63,
        68,  73,  79,  84,  89,  94, 100])
```

1.1.11 Empty :

[21]:
```
empty = np.empty((1,5))
empty
```

[21]:
```
array([[4.67422725e-310, 0.00000000e+000, 4.67494390e-310,
        4.67494390e-310, 1.97626258e-323]])
```

1.2 Array Indexing :

1.2.1 Index :

[22]:
```
a = np.array([1,2,3,4,5,6,7])
a[0]            # as like normal indexing
```

[22]:
```
1
```

1.2.2 Random array :

[23]:
```
random = np.random.randint(1,100 , size = (5,5))
random        # 1-100 random number 5x5 matrix
```

[23]:
```
array([[89, 28, 72,  1, 35],
       [52, 45, 99, 53, 35],
       [93,  6,  2, 31, 35],
       [ 7, 57, 80, 87, 33],
```

```
       [81, 96, 97, 48, 31]])
```

1.2.3 Index access 2-D array :

[24]:
```
x = random[0][0]            # row 0 , culumn 0
x
```

[24]:
```
89
```

[25]:
```
y = random[0,0]            # rew 0 , culumn 0
y
```

[25]:
```
89
```

1.2.4 Index access 3-D array :

[26]:
```
random = np.random.randint(1,100 , size=(2,3,4))
random          # size(2,3,4) --> 3 : row  , 4 : column and 2 stage
```

[26]:
```
array([[[ 4, 78, 15, 48],
        [24, 49, 60, 83],
        [45, 64, 95, 54]],

       [[26, 60, 11, 75],
        [35, 96, 47, 22],
        [19, 40, 49,  6]]])
```

[27]:
```
x = random[0][0][0]
x
```

[27]:
```
4
```

[28]:
```
x = random[0,0,0]
x
```

[28]:
```
4
```

1.3 Array Slicing :

1.3.1 Slicing 1-D array :

[29]:
```
a = np.array([1,2,3,4,5,6,7,8,9])
a[1:4]
```

[29]:
```
array([2, 3, 4])
```

[30]:
```
a[:6]
```

[30]:
```
array([1, 2, 3, 4, 5, 6])
```

[31]:
```
a[2:]
```

[31]:
```
array([3, 4, 5, 6, 7, 8, 9])
```

1.3.2 Slicing 2-D array :

[32]:
```
b = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
b[0:2 , 0:1]   # row , column
```

[32]:
```
array([[1],
       [4]])
```

1.4 Manipulate array shape :

[33]:
```
# Total 5 way:
    # 1. reshape()
    # 2. resize()
    # 3. revel()
    # 4. flatten()
    # 5. defining array shape
```

1.4.1 1. reshape() :

[34]:
```
b = np.array([[1, 2, 3] ,
              [4, 5, 6]])
np.reshape(b,(3,2))     # if element is not emough it will show element error
```

[34]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

1.4.2 2. resize() :

[35]:
```
# to solve this element error we can use resize()
b = np.array([[1, 2, 3] ,
              [4, 5, 6]])
np.resize(b, (4,3))   # repeat the element if extra need
```

[35]:
```
array([[1, 2, 3],
       [4, 5, 6],
```

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

1.4.3 3. revel () :

[36]:
```
# change main array
b = np.array([[1, 2, 3] ,
              [4, 5, 6]])
ravel = np.ravel(b)
ravel
```

[36]:
```
array([1, 2, 3, 4, 5, 6])
```

[37]:
```
ravel[0] = 100              # return a view of array and change main array
ravel
```

[37]:
```
array([100,   2,   3,   4,   5,   6])
```

1.4.4 4. flatten() :

[38]:
```
# convart any array to 1-D array
# don't change main array
b = np.array([[1, 2, 3] ,
              [4, 5, 6]])
flatten = b.flatten()       # return a copy of arry and can't change main array
flatten
```

[38]:
```
array([1, 2, 3, 4, 5, 6])
```

[39]:
```
flatten[0] = 100            # change flatten only not b
b
```

[39]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

1.4.5 5. defining array shape :

[40]:
```
b = np.array([[1, 2, 3] ,
              [4, 5, 6]])
b.shape = (3, 2)            # need equal element
b
```

[40]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

1.5 Array staking :

1.5.1 hstake() :

[41]:
```
a = np.arange(1,10).reshape(3,3)     # create a 2-D array 1 to 9
b = 2*a                              # 2 * all element
b
```

[41]:
```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

[42]:
```
x = np.hstack((a,b))                 # a + b in horizontal
x
```

[42]:
```
array([[ 1,  2,  3,  2,  4,  6],
       [ 4,  5,  6,  8, 10, 12],
       [ 7,  8,  9, 14, 16, 18]])
```

1.5.2 vstake() :

[43]:
```
y = np.vstack((a,b))                 # a + b in vartical
y
```

[43]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

1.5.3 column stake :

[44]:
```
z = np.column_stack((a , b))         # semilar to hstack
z
```

[44]:
```
array([[ 1,  2,  3,  2,  4,  6],
       [ 4,  5,  6,  8, 10, 12],
       [ 7,  8,  9, 14, 16, 18]])
```

[45]:
```
z = np.row_stack((a , b))            # semilar to vstack
z
```

[45]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [ 2,  4,  6],
```

```
       [ 8, 10, 12],
       [14, 16, 18]])
```

1.5.4 concatinate :

[46]:
```
z = np.concatenate((a,b) , axis=1)  # axis 1 = horaizonal and azis 0 = vartical
z
```

[46]:
```
array([[ 1,  2,  3,  2,  4,  6],
       [ 4,  5,  6,  8, 10, 12],
       [ 7,  8,  9, 14, 16, 18]])
```