Numpy Practice

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

In [2]:
```
food = np.array(["samosa","pakora","ratia"])
food
```

Out[2]:
```
array(['samosa', 'pakora', 'ratia'], dtype='<U6')
```

In [3]:
```
price = np.array([5,5,5,])
price
```

Out[3]:
```
array([5, 5, 5])
```

In [4]:
```
type(price)
```

Out[4]:
```
numpy.ndarray
```

In [5]:
```
type(food)
```

Out[5]:
```
numpy.ndarray
```

In [6]:
```
len(food)
```

Out[6]:
```
3
```

In [7]:
```
price[0]
```

Out[7]:
```
5
```

In [8]:
```
food[0:]
```

Out[8]:
```
array(['samosa', 'pakora', 'ratia'], dtype='<U6')
```

In [9]:
```
#zeros
np.zeros(6)
```

Out[9]:
```
array([0., 0., 0., 0., 0., 0.])
```

In [10]:
```
# ones
np.ones(5)
```

Out[10]:
```
array([1., 1., 1., 1., 1.])
```

Empty numpy array Assignment

- The np.empty function actually does fill the array with values. It's just that the values are completely arbitrary
- the values are just arbitrary. You can think of them like arbitrary, "garbage" values.

In [11]:
```
# empty
np.empty(5)
```

Out[11]:
```
array([1., 1., 1., 1., 1.])
```

About Arrays

- When we are creating arrays so the indexing starts from whole numbers means from 0 not from 1 or natural numbers so if we want to output last index so we have to start counting from 0 untill the desired index according to my knowledge

In [12]:
```
np.arange(10)
```

Out[12]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [13]:
```
# specific range with specific intervel
np.arange(2,20,2)
```

Out[13]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [14]:
```
# table of five
np.arange(5,55,5)
```

Out[14]:
```
array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])
```

In [17]:
```
# line space
np.linspace(1,10,num=5)
```

Out[17]:
```
array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])
```

In [18]:
```
# specify data type
np.ones(5,dtype=np.int64)
```

Out[18]:
```
array([1, 1, 1, 1, 1], dtype=int64)
```

In [19]:
```
np.ones(30,dtype=np.float64)
```

Out[19]:
```
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 functions

In [21]:
```
a= np.array([10,12,15,2,4,100,10.5,0.12])
a
```

```
array([ 10.  ,  12.  ,  15.  ,   2.  ,   4.  , 100.  ,  10.5 ,   0.12])
```

Out[21]:

In [22]:
```
a.sort()
a
```

Out[22]:
```
array([  0.12,   2.  ,   4.  ,  10.  ,  10.5 ,  12.  ,  15.  , 100.  ])
```

In [23]:
```
b=np.array([10.2,3.5,6.6,90,105.5])
b
```

Out[23]:
```
array([ 10.2,   3.5,   6.6,  90. , 105.5])
```

In [26]:
```
c=np.concatenate((a,b))
c
```

Out[26]:
```
array([  0.12,   2.  ,   4.  ,  10.  ,  10.5 ,  12.  ,  15.  , 100.  ,
        10.2 ,   3.5 ,   6.6 ,  90.  , 105.5 ])
```

In [28]:
```
c.sort()
c
```

Out[28]:
```
array([  0.12,   2.  ,   3.5 ,   4.  ,   6.6 ,  10.  ,  10.2 ,  10.5 ,
        12.  ,  15.  ,  90.  , 100.  , 105.5 ])
```

2-D Array

In [31]:
```
a=np.array([[1,2,3,4,5],[6,7,8,9,10]])
a
```

Out[31]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

In [32]:
```
b=np.array([[11,12,13,14,15],[16,17,18,19,20]])
b
```

Out[32]:
```
array([[11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20]])
```

The error was that both the arrays was of different orders so for concantenation we have to create array of same order like the one i created.

In [33]:
```
c = np.concatenate((a,b))
c
```

Out[33]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20]])
```

In [35]:

```
c = np.concatenate((a,b), axis=1)
c
```

Out[35]:
```
array([[ 1,  2,  3,  4,  5, 11, 12, 13, 14, 15],
       [ 6,  7,  8,  9, 10, 16, 17, 18, 19, 20]])
```

In [36]:
```
# 3-D Array
d = np.array([[[0,1,2,3],[4,5,6,7]],
              [[0,1,2,3],[4,5,6,7]],
              [[0,1,2,3],[4,5,6,7]]])
d
```

Out[36]:
```
array([[[0, 1, 2, 3],
        [4, 5, 6, 7]],

       [[0, 1, 2, 3],
        [4, 5, 6, 7]],

       [[0, 1, 2, 3],
        [4, 5, 6, 7]]])
```

In [37]:
```
#find the number of dimensions
d.ndim
```

Out[37]:
```
3
```

In [38]:
```
#number of elements
d.size
```

Out[38]:
```
24
```

In [40]:
```
#shape shows that d have 3 dimensions and 2by4 order
d.shape
```

Out[40]:
```
(3, 2, 4)
```

In [44]:
```
e = np.arange(9)
e
```

Out[44]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [45]:
```
# reshape
e.reshape(3,3)
```

Out[45]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [46]:
```
np.reshape(e, newshape=(1,9), order = 'c')
```

Out[46]:
```
array([[0, 1, 2, 3, 4, 5, 6, 7, 8]])
```

In [47]:
```
# convert 1D to 2D
a = np.array([1,2,3,4,5,6,7,8,9])
a
```

Out[47]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [48]:
```
# Row wise
b = a[np.newaxis, :]
b
```

Out[48]:
```
array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

In [49]:
```
# column wise
c = a[: , np.newaxis]
c
```

Out[49]:
```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [8],
       [9]])
```

In [50]:
```
c[2:6]
```

Out[50]:
```
array([[3],
       [4],
       [5],
       [6]])
```

In [51]:
```
a
```

Out[51]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [52]:
```
a*6
```

Out[52]:
```
array([ 6, 12, 18, 24, 30, 36, 42, 48, 54])
```

In [53]:
```
a+6
```

Out[53]:
```
array([ 7,  8,  9, 10, 11, 12, 13, 14, 15])
```

In [54]:
```
a.sum()
```

Out[54]:
```
45
```

In [55]:
```
a.mean()
```

Out[55]:
```
5.0
```

More on numpy

In [56]:
```
#printing array of index 0
d
```

Out[56]:
```
array([[[0, 1, 2, 3],
        [4, 5, 6, 7]],

       [[0, 1, 2, 3],
        [4, 5, 6, 7]],

       [[0, 1, 2, 3],
        [4, 5, 6, 7]]])
```

In [57]:
```
print(d[0])
```

```
[[0 1 2 3]
 [4 5 6 7]]
```

In [58]:
```
print(d[d < 5])
```

```
[0 1 2 3 4 0 1 2 3 4 0 1 2 3 4]
```

In [60]:
```
#divisible by 2
k=d[d%2==0]
k
```

Out[60]:
```
array([0, 2, 4, 6, 0, 2, 4, 6, 0, 2, 4, 6])
```

In [61]:
```
#printing an array greater than 2 and less than 7
h=d[(d > 2) & (d < 7)]
h
```

Out[61]:
```
array([3, 4, 5, 6, 3, 4, 5, 6, 3, 4, 5, 6])
```

In [62]:
```
#creating new array from existing one
e
```

Out[62]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [63]:
```
z=e[3:8]
z
```

Out[63]:
```
array([3, 4, 5, 6, 7])
```

In [64]:
```
#reversing an array
np.flip(e)
```

Out[64]:
```
array([8, 7, 6, 5, 4, 3, 2, 1, 0])
```