Array Creation

In [5]:
```
import numpy as np
```

In [6]:
```
#Create an array and print it.

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

print(arr)

print(type(arr))

#As we can see below, it is of n-dimensinal array.
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

In [7]:
```
#If we want to print the array in the form of matrix, we can do it by -

arr = np.array([[10,20,30,40,50],[100,210,330,430,510]])

arr

#Here the important thing to remember is that every list we provide must contain same number of elements which
```

Out[7]:
```
array([[ 10,  20,  30,  40,  50],
       [100, 210, 330, 430, 510]])
```

In [8]:
```
#for instance

arr1 = np.array([[1,2,4,5,6],[3,2,4,5]])

arr1
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[8], line 3
      1 #for instance
----> 3 arr1 = np.array([[1,2,4,5,6],[3,2,4,5]])
      5 arr1

ValueError: setting an array element with a sequence. The requested array has an inhomogeneous shape after 1 dim
ensions. The detected shape was (2,) + inhomogeneous part.
```

Indexing & Slicing

In [9]:
```
arr
```

Out[9]:
```
array([[ 10,  20,  30,  40,  50],
       [100, 210, 330, 430, 510]])
```

In [10]:
```
(arr[0:1])
```

Out[10]:
```
array([[10, 20, 30, 40, 50]])
```

In [11]:
```
print((arr[0:1]))
```

```
[[10 20 30 40 50]]
```

In [12]:
```
arr2 = ([1,3,5,8,9])

arr2[0:4]
```

Out[12]:
```
[1, 3, 5, 8]
```

In [13]:
```
arr2[0:]
```

Out[13]:
```
[1, 3, 5, 8, 9]
```

In [14]:
```
arr2[:]
```

Out[14]:
```
[1, 3, 5, 8, 9]
```

In [15]:
```
arr2[:2]
```

Out[15]:
```
[1, 3]
```

In [16]:
```
arr
```

Out[16]:
```
array([[ 10,  20,  30,  40,  50],
       [100, 210, 330, 430, 510]])
```

In [17]:
```
#index one element from the first array
#index two element from the second array

arr[[0, 1],[1,2]]
```

Out[17]:
```
array([ 20, 330])
```

In [18]:
```
#print the first 2 elements of each array

arr[0:2,0:2]
```

Out[18]:
```
array([[ 10,  20],
       [100, 210]])
```

In [19]:
```
#even though we have given different boundaries for each array, it is not possible/valid in Numpy. Therefore, w

arr[0:2,0:3]
```

Out[19]:
```
array([[ 10,  20,  30],
       [100, 210, 330]])
```

In [20]:
```
np.shape(arr)
```

Out[20]:
```
(2, 5)
```

In [21]:
```
np.size(arr)
```

Out[21]:
```
10
```

In [22]:
```
np.ndim(arr)
```

Out[22]:
```
2
```

In [23]:
```
arr.dtype
```

Out[23]:
```
dtype('int32')
```

In [24]:
```
array = [[[1,2,3,4,5],[0,9,8,7,6],[11,12,13,14,15]]]

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

Out[24]:
```
[[0, 9, 8, 7, 6]]
```

In [25]:
```
#Since the list above is still a python list, it does not work with 'array.dtype()' code. We need to change tha

np_array = np.array(array)

print(np_array.dtype)
```

```
int32
```

In [26]:
```
#To convert a data type into another data type, we use .astype function as shown below.

np_array.astype(float)
```

Out[26]:
```
array([[[ 1.,  2.,  3.,  4.,  5.],
        [ 0.,  9.,  8.,  7.,  6.],
        [11., 12., 13., 14., 15.]]])
```

In [27]:
```
arr
```

Out[27]:
```
array([[ 10,  20,  30,  40,  50],
       [100, 210, 330, 430, 510]])
```

In [28]:
```
#length of th array as in how many rows are there.

print(len(arr))
```

```
2
```

In [29]:
```
#Total number of elements in all of the arrays.

np.size(arr)
```

Out[29]:
```
10
```

Mathematical Operations

In [30]:
```
#Subtraction:

arr
```

Out[30]:
```
array([[ 10,  20,  30,  40,  50],
       [100, 210, 330, 430, 510]])
```

In [31]:
```
array = np.array([[ 10,  20,  30,  40,  50],
       [100, 210, 330, 430, 510]])

array[1] - array[0]
```

Out[31]:
```
array([ 90, 190, 300, 390, 460])
```

In [32]:
```
array [1] / array [0]
```

Out[32]:
```
array([10.  , 10.5 , 11.  , 10.75, 10.2 ])
```

In [33]:
```
array[1]*array[0]
```

Out[33]:
```
array([ 1000,  4200,  9900, 17200, 25500])
```

In [34]:
```
np.exp(arr)
```

Out[34]:
```
array([[2.20264658e+004, 4.85165195e+008, 1.06864746e+013,
        2.35385267e+017, 5.18470553e+021],
       [2.68811714e+043, 1.59162664e+091, 2.07576903e+143,
        5.57991031e+186, 3.09161760e+221]])
```

In [35]:
```
np.sqrt(arr)
```

Out[35]:
```
array([[ 3.16227766,  4.47213595,  5.47722558,  6.32455532,  7.07106781],
       [10.        , 14.49137675, 18.16590212, 20.73644135, 22.58317958]])
```

In [36]:
```
array = [10,9,8,7,6]
array2 = [1,2,3,4,5]

np.power(array,array2)
```

Out[36]:
```
array([  10,   81,  512, 2401, 7776])
```

In [37]:
```
#When we use plus, we might think that addition operation happens, but here "concatenation" operation happens a

a = [12,13,14,15,16]
b = [10,19,18,17,12]

a+b
```

Out[37]:
```
[12, 13, 14, 15, 16, 10, 19, 18, 17, 12]
```

In [38]:
```
#if we want to perfom addition operation, we need do as follows:

a = [12,13,14,15,16]
b = [10,19,18,17,12]

ara = np.array(a)
arb = np.array(b)

sum = ara + arb

sum
```

Out[38]:
```
array([22, 32, 32, 32, 28])
```

In [39]:
```
#Concatenate using axis as a reference 1

cona = np.array([[1,2,3],[4,5,6]])
conb = np.array([[7,8,9],[10,12,13]])

np.concatenate([cona,conb], axis = 0)
```

Out[39]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 12, 13]])
```

In [40]:
```
#Concatenate using axis as a reference 2
```

```
cona = np.array([[1,2,3],[4,5,6]])
conb = np.array([[7,8,9],[10,12,13]])

np.concatenate([cona,conb], axis = 1)
```

Out[40]:
```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 12, 13]])
```

In [41]:
```
#We can achieve the same by using hstack function i.e, "Horizontal Concatenation"

cona = np.array([[1,2,3],[4,5,6]])
conb = np.array([[7,8,9],[10,12,13]])

np.hstack([cona,conb])
```

Out[41]:
```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 12, 13]])
```

In [42]:
```
#We can achieve the another dimension by using vstack function i.e, "Vertical Concatenation"

cona = np.array([[1,2,3],[4,5,6]])
conb = np.array([[7,8,9],[10,12,13]])

np.vstack([cona,conb])
```

Out[42]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 12, 13]])
```

In [43]:
```
#Splitting array into whatever no.of arrays we want.

p = ([11,22,33,44,55])
np.array_split(p,3)
```

Out[43]:
```
[array([11, 22]), array([33, 44]), array([55])]
```

In [44]:
```
# If we want ot retrieve speciic element from the element we use -
# Here we are splitting the array into 2.
# Then the indexing changes, afer the indexing changes we are asking for the index number 1 which is [44,55] in

p = ([11,22,33,44,55])
q = np.array_split(p,2)
print(q[1])
```

```
[44 55]
```

In [45]:
```
p1 = np.array([[11,22,33],[66,44,55]])
q1 = np.array_split(p1,5)
print(q1)
```

```
[array([[11, 22, 33]]), array([[66, 44, 55]]), array([], shape=(0, 3), dtype=int32), array([], shape=(0, 3), dty
pe=int32), array([], shape=(0, 3), dtype=int32)]
```

Adding & Deleting Elements

In [46]:
```
#append, insert, delete

apparray = np.array([1,2,3,4])
np.append(apparray,30)
```

Out[46]:
```
array([ 1,  2,  3,  4, 30])
```

In [47]:
```
apparray = np.array([1,2,3,4])
np.append(apparray,[30,50])

#we can append elements in the form of array as well.
#Main point to be remembered is that append adds the elements from the last, while insert gives us the facility
```

Out[47]:
```
array([ 1,  2,  3,  4, 30, 50])
```

In [48]:
```
apparray = np.array([1,2,3,4])
np.insert(apparray,2,50) #array,index,value to be inserted
```

Out[48]:
```
array([ 1,  2, 50,  3,  4])
```

In [49]:
```
apparray = np.array([[1,2,3,4],[5,6,7,8]])

np.insert(apparray,2,[40,50],axis = 1)
```

Out[49]:
```
array([[ 1,  2, 40,  3,  4],
       [ 5,  6, 50,  7,  8]])
```

In [50]:
```
apparray = np.array([[1,2,3,4],[5,6,7,8]])
print(apparray)

np.delete(apparray,1,axis=1)
```

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

Out[50]:
```
array([[1, 3, 4],
       [5, 7, 8]])
```

Serach, Sort & Filter

In [51]:
```
ar = np.array([10,8,4,11,3])
print(np.sort(ar))
```

```
[ 3  4  8 10 11]
```

In [52]:
```
ar = np.array([10,8,4,11,3])
s = np.where(ar == 11)

print(s)
```

```
(array([3], dtype=int64),)
```

In [53]:
```
ar = np.array([10,8,4,11,3])
s = np.where(ar %2 == 0)

print(s)
```

```
(array([0, 1, 2], dtype=int64),)
```

In [54]:
```
#There is a function known as "searchsorted" in which in order to get the searched value, the array needs to be

ar = np.array([10,8,4,11,3])
ss = np.searchsorted(ar, 11)

print(ss)

#Even though we got the answer here, it is wrong and the function is assuming the array to be sorted.
```

```
5
```

In [55]:
```
#In order to correct the above problem, we need to do as below -

ar = np.array([3,4,8,10,11])
ss = np.searchsorted(ar, 11)

print(ss)
```

```
4
```

In [56]:
```
ar = np.array([3,4,8,10,11])

filterarray = ar > 4

arr = ar[filterarray]

arr
```

Out[56]:
```
array([ 8, 10, 11])
```

Aggregating Functions

In [57]:
```
ap = np.array([10,20,30,40,50])

print(np.sum(ap))
print(np.prod(ap))
print(np.min(ap))
print(np.max(ap))
print(np.count(ap))
```

```
150
12000000
10
50
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[57], line 7
      5 print(np.min(ap))
      6 print(np.max(ap))
----> 7 print(np.count(ap))

File ~\anaconda3\Lib\site-packages\numpy\__init__.py:333, in __getattr__(attr)
    330     "Removed in NumPy 1.25.0"
    331     raise RuntimeError("Tester was removed in NumPy 1.25.")
--> 333 raise AttributeError("module {!r} has no attribute "
    334                       "{!r}".format(__name__, attr))

AttributeError: module 'numpy' has no attribute 'count'
```

In [58]:
```
#In order to count the no.of elements, we need to use size function

print(np.size(ap))

print(np.mean(ap))
print(np.cumsum(ap))
print(np.cumprod(ap))
```

```
5
30.0
[ 10  30  60 100 150]
[      10      200     6000   240000 12000000]
```

In [59]:
```
ap = np.array([100,220,330,450,520,100])

print(np.mean(ap))
print(np.median(ap))
print(np.mode(ap))
```

```
286.6666666666667
275.0
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[59], line 5
      3 print(np.mean(ap))
      4 print(np.median(ap))
----> 5 print(np.mode(ap))

File ~\anaconda3\Lib\site-packages\numpy\__init__.py:333, in __getattr__(attr)
    330     "Removed in NumPy 1.25.0"
    331     raise RuntimeError("Tester was removed in NumPy 1.25.")
--> 333 raise AttributeError("module {!r} has no attribute "
    334                       "{!r}".format(__name__, attr))

AttributeError: module 'numpy' has no attribute 'mode'
```

In [72]:
```
#in order to get mode, we need to use 'stats' function

import numpy as np
from scipy import stats

ap = np.array([100,220,330,450,520,100])

print(stats.mode(ap))
print(np.std(ap))
print(np.var(ap))
```

```
ModeResult(mode=100, count=2)
161.82981458584473
26188.888888888887
```

In [69]:
```
# -1 represents inversely proportional relationship.
# 1 represents directly proportional relationship.
# 0 represents no relationship

alcohol = [100,300,200,130,240]
casualties = [12,9,25,10,30]

print(np.corrcoef([alcohol,casualties]))
```

```
[[1.        0.2291924]
 [0.2291924 1.       ]]
```