Day 1 - Numpy Chapter 1

What is Numpy?

- Introduction to Numpy:-
  - Numpy is a Python Package and it stands for Numerical Python.
  - This was created in 2005 and it was developed by Travis Oliphant.
  - It Provides lot of Functions to work in a domain of Linear Algebra and Matrix.
  - It provides function related to to Arrays.
  - It creates an Array called ndarray.
  - Most of thge part of numpy created in C and C++.

Operations in Numpy

- Mathematical and Statistical and Logical Operations on Array.
- It is used for Scientific Calculation.
- Operations are related to mathematics, Statistics, Linear Algebra and Random Number Generations.
- Transformation and routine for shape manipulation of the Array.

In [1]:
```
%%timeit

a = []
for i in range(1,15):
    a.append(i**2)
```

```
3.43 µs ± 298 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

In [2]:
```
%timeit [i**2 for i in range(1,15)]
```

```
3.08 µs ± 109 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

Numpy

In [3]:
```
# importing numpy as np
import numpy as np
```

In [4]:
```
x = [1,2,3]
x
```

Out[4]:
```
[1, 2, 3]
```

In [6]:
```
type(x)
```

Out[6]:
```
list
```

In [7]:
```
x = np.array([1,2,3])
x
```

Out[7]:
```
array([1, 2, 3])
```

In [8]:
```
type(x)
```

Out[8]:
```
numpy.ndarray
```

1D Array

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

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

In [10]:
```
a = [1,2,3,4]
arr = np.array(a)
arr
```

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

In [13]:
```
# take user input
a = []
cnt = 1
for i in range(int(input("How many elements you want: "))):
    val = eval(input(f"Enter the value{cnt}: "))
    a.append(val)
    cnt+=1
b = np.array(a)
b
```

```
How many elements you want: 6
Enter the value1: 12
Enter the value2: 24
Enter the value3: 59
Enter the value4: 84
Enter the value5: 45
Enter the value6: 51
```

Out[13]:
```
array([12, 24, 59, 84, 45, 51])
```

In [18]:
```
# take user input
a = []
cnt = 1
n=int(input("How many elements you want: "))
for i in range(n):
    val = eval(input(f"Enter the value{cnt}: "))
    a.append(val)
    cnt+=1
b = np.array(a)
b.reshape(n).T
```

```
How many elements you want: 6
Enter the value1: 12
Enter the value2: 21
Enter the value3: 42
Enter the value4: 26
Enter the value5: 54
Enter the value6: 85
```

Out[18]:
```
array([12, 21, 42, 26, 54, 85])
```

2D Array

In [25]:
```
a = np.array([[1,2,3],[4,5,6]])
a
```

Out[25]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [26]:
```
a.shape
```

Out[26]:
```
(2, 3)
```

In [27]:
```
# checking the type
type(a)
```

Out[27]:
```
numpy.ndarray
```

In [30]:
```
# To check the Dimensions
print(np.ndim(a))
print(a.ndim)
```

```
2
2
```

In [31]:
```
# checking the Dtype
a.dtype
```

Out[31]:
```
dtype('int32')
```

In [32]:
```
print(a.dtype)
```

```
int32
```

3D Array (matrices, rows, columns)

In [33]:
```
a = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[1,5,9]]])
a
```

Out[33]:
```
array([[[1, 2, 3],
        [4, 5, 6]],

       [[7, 8, 9],
        [1, 5, 9]]])
```

In [36]:
```
a = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[8,5,2]],[[7,5,3],[9,5,1]]])
a
```

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

       [[7, 8, 9],
        [8, 5, 2]],

       [[7, 5, 3],
        [9, 5, 1]]])
```

In [37]:
```
a.ndim
```

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

In [38]:
```
np.ndim(a)
```

Out[38]:
```
3
```

In [39]:
```
a.shape
```

Out[39]:
```
(3, 2, 3)
```

In [40]:
```
b = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[8,5,2]],[[7,5,3],[9,5,1]],[[1,2,6],[8,7,9]]])
b
```

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

       [[7, 8, 9],
        [8, 5, 2]],

       [[7, 5, 3],
        [9, 5, 1]],

       [[1, 2, 6],
        [8, 7, 9]]])
```

In [41]:
```
b.shape
```

Out[41]:
```
(4, 2, 3)
```

In [43]:
```
b.ndim
```

Out[43]:
```
3
```

In [45]:
```
c = b.T
c
```

Out[45]:
```
array([[[1, 7, 7, 1],
        [4, 8, 9, 8]],

       [[2, 8, 5, 2],
        [5, 5, 5, 7]],

       [[3, 9, 3, 6],
        [6, 2, 1, 9]]])
```

In [46]:
```
c.shape
```

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

Attribute of Numpy

In [47]:
```
a = np.array([[1,2,3],[4,5,6]])
a
```

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

In [48]:
```
# gives shape of an matrix or array in the row and columnwise.
a.shape
# it says 2 rows and 3 columns
```

Out[48]:
```
(2, 3)
```

In [49]:
```
# Return the total no. of observation in an array.
a.size
```

Out[49]:
```
6
```

In [50]:
```
# this will change the shape of an array Temporary
a.reshape(3,2)
```

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

In [51]:
```
a.T
```

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

In [53]:
```
a.shape=(3,2)
a
```

Out[53]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [54]:
```
a
```

Out[54]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [55]:
```
a
```

Out[55]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [58]:
```
np.reshape(a)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [58], line 1
----> 1 np.reshape(a)

File <__array_function__ internals>:179, in reshape(*args, **kwargs)

TypeError: _reshape_dispatcher() missing 1 required positional argument: 'newshape'
```

In [59]:
```
# Transpose
# this will convert rpws into column amd columns into rows
a.T
# changes the shape of an array/ matrix
```

Out[59]:
```
array([[1, 3, 5],
       [2, 4, 6]])
```

In [60]:
```
a
```

Out[60]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [61]:
```
# It will return you the dimensions of an Array.
a.ndim
```

Out[61]:
```
2
```

Joining of Array

In [62]:
```
np.arange(1,7)
```

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

In [63]:
```
np.arange(1,7).reshape(3,2)
```

Out[63]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [66]:
```
np.arange(1,7).reshape(3,2).T
```

Out[66]:
```
array([[1, 3, 5],
       [2, 4, 6]])
```

In [67]:
```
np.arange(1,7).reshape(2,3).shape
```

Out[67]:
```
(2, 3)
```

In [68]:
```
np.arange(1,7).reshape(2,3).T
```

Out[68]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [69]:
```
np.arange(1,7).reshape(2,3).T.shape
```

Out[69]:
```
(3, 2)
```

In [70]:
```
a = np.arange(1,7).reshape(2,3)

b = np.arange(7,13).reshape(2,3)
```

In [71]:
```
a
```

Out[71]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [72]:
```
b
```

Out[72]:
```
array([[ 7,  8,  9],
       [10, 11, 12]])
```

Concatenate

In [73]:
```
#a.Concatenate((arr1,arr2))
# It is used to join two different Array and it takes tuples.
```

In [74]:
```
np.concatenate((a,b))
```

Out[74]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [75]:
```
np.concatenate((a,b),axis =1)
```

Out[75]:
```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

In [78]:
```
np.concatenate((a,b),axis =1,dtype= 'float')
```

Out[78]:
```
array([[ 1.,  2.,  3.,  7.,  8.,  9.],
       [ 4.,  5.,  6., 10., 11., 12.]])
```

In [77]:
```
np.concatenate((a,b),axis =1,dtype = 'object')
```

Out[77]:
```
array([[1, 2, 3, 7, 8, 9],
       [4, 5, 6, 10, 11, 12]], dtype=object)
```

In [79]:
```
np.concatenate((b,a), axis= 1)
```

Out[79]:
```
array([[ 7,  8,  9,  1,  2,  3],
       [10, 11, 12,  4,  5,  6]])
```

In [80]:
```
np.concatenate((b,a))
```

Out[80]:
```
array([[ 7,  8,  9],
       [10, 11, 12],
       [ 1,  2,  3],
       [ 4,  5,  6]])
```

hstack and vstack

In [81]:
```
# hstack
# it is used to concatenate two differents Arrays horizontally
# (column wise)
np.hstack((a,b))
```

Out[81]:
```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

In [83]:
```
np.hstack((b,a))
```

Out[83]:
```
array([[ 7,  8,  9,  1,  2,  3],
       [10, 11, 12,  4,  5,  6]])
```

In [82]:
```
# vstack
# It is used to concatenate two different Array vertically
# (row wise)
np.vstack((a,b))
```

Out[82]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [84]:
```
np.vstack((b,a))
```

Out[84]:
```
array([[ 7,  8,  9],
       [10, 11, 12],
       [ 1,  2,  3],
       [ 4,  5,  6]])
```

In [85]:
```
# stack
# it is usewd to convert 2D Array into 3D Array

np.stack((a,b))
```

Out[85]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

In [86]:
```
np.stack((b,a))
```

Out[86]:
```
array([[[ 7,  8,  9],
        [10, 11, 12]],

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

In [87]:
```
# ravel
# It is used to convert 2D into 1D Array
# Return a contiguous flattened array.
# np.ravel(a)
print(a)
np.ravel(a)
```

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

Out[87]:
```
array([1, 2, 3, 4, 5, 6])
```

Append

In [88]:
```
# append(arr,list of element)
# It is used to add the element in the existing Array.

c = np.arange(1,7)
c
```

Out[88]:
```
array([1, 2, 3, 4, 5, 6])
```

In [89]:
```
np.append(c,[7,8,9])
```

Out[89]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [90]:
```
a
```

Out[90]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [91]:
```
np.append(a,[[7,8,9]])
```

Out[91]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [92]:
```
np.append(a,[[7,8,9]],axis=0)
```

Out[92]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [96]:
```
np.append(a,[[7,8,9]],axis=1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [96], line 1
----> 1 np.append(a,[[7,8,9]],axis=1)

File <__array_function__ internals>:180, in append(*args, **kwargs)

File ~\anaconda3\lib\site-packages\numpy\lib\function_base.py:5444, in append(arr, values, axis)
   5442     values = ravel(values)
   5443     axis = arr.ndim-1
-> 5444 return concatenate((arr, values), axis=axis)

File <__array_function__ internals>:180, in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimensio
n 0, the array at index 0 has size 2 and the array at index 1 has size 1
```

In [95]:
```
np.append(a,[[7],[8],[9]],axis=1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [95], line 1
----> 1 np.append(a,[[7],[8],[9]],axis=1)

File <__array_function__ internals>:180, in append(*args, **kwargs)

File ~\anaconda3\lib\site-packages\numpy\lib\function_base.py:5444, in append(arr, values, axis)
   5442     values = ravel(values)
   5443     axis = arr.ndim-1
-> 5444 return concatenate((arr, values), axis=axis)

File <__array_function__ internals>:180, in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimensio
n 0, the array at index 0 has size 2 and the array at index 1 has size 3
```

Insert

In [97]:
```
# insert
# it is used to add the element at specific index.
```

In [98]:
```
d = np.arange(2,6)
d
```

Out[98]:
```
array([2, 3, 4, 5])
```

In [99]:
```
np.insert(d,0,56)
```

Out[99]:
```
array([56,  2,  3,  4,  5])
```

In [100]:
```
d
```

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

In [101]:
```
np.insert(d,-1,65)
```

Out[101]:
```
array([ 2,  3,  4, 65,  5])
```

In [102]:
```
np.insert(d,-2,65)
```

Out[102]:
```
array([ 2,  3, 65,  4,  5])
```

In [103]:
```
np.insert(d,-0,65)
```

Out[103]:
```
array([65,  2,  3,  4,  5])
```

Delete

In [104]:
```
# delete(array, list of element index)
# It is used to delete element the from the array.
a = np.arange(1,6)
```

In [105]:
```
a
```

Out[105]:
```
array([1, 2, 3, 4, 5])
```

In [106]:
```
np.delete(a,[0,1])
```

Out[106]:
```
array([3, 4, 5])
```

In [107]:
```
np.delete(a,[0])
```

Out[107]:
```
array([2, 3, 4, 5])
```

In [108]:
```
np.delete(a,[-1])
```

Out[108]:
```
array([1, 2, 3, 4])
```

In [109]:
```
np.delete(a,[-2,-2])
```

Out[109]:
```
array([1, 2, 3, 5])
```

In [110]:
```
np.delete(a,[-9])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [110], line 1
----> 1 np.delete(a,[-9])

File <__array_function__ internals>:180, in delete(*args, **kwargs)

File ~\anaconda3\lib\site-packages\numpy\lib\function_base.py:5156, in delete(arr, obj, axis)
   5153 if single_value:
   5154     # optimization for a single value
   5155     if (obj < -N or obj >= N):
-> 5156         raise IndexError(
   5157             "index %i is out of bounds for axis %i with "
   5158             "size %i" % (obj, axis, N))
   5159     if (obj < 0):
   5160         obj += N

IndexError: index -9 is out of bounds for axis 0 with size 5
```

In [2]:
```
import numpy as np
```

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

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

In [4]:
```
np.delete(b,1,0)
```

Out[4]:
```
array([[ 1,  2,  3,  4],
       [ 9, 10, 11, 12]])
```

In [5]:
```
np.delete(b, np.s_[::2], 1)
```

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

In [122]:
```
np.delete(b,1)
```

Out[122]:
```
array([ 1,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [117]:
```
np.delete(a,,axis=0)
```

```
  Cell In [117], line 1
    np.delete(a,obj = [0:2],axis=0)
                       ^
SyntaxError: invalid syntax
```

In [111]:
```
np.delete(a,[:])
```

```
  Cell In [111], line 1
    np.delete(a,[:])
                  ^
SyntaxError: invalid syntax
```

In [112]:
```
np.delete(a,[,:])
```

```
  Cell In [112], line 1
    np.delete(a,[,:])
                  ^
SyntaxError: invalid syntax
```

In [125]:
```
a
```

Out[125]:
```
array([1, 2, 3, 4, 5])
```

In [126]:
```
np.delete(a,0)
```

Out[126]:
```
array([2, 3, 4, 5])
```

Day 2 - Numpy Array Creation Routine. Chapter 2

In [127]:
```
import numpy as np
```

In [130]:
```
# empty(shape,dtype)
# It creates an uninitialize Array of Specified Shape and Datatype
np.empty([3,2])
```

Out[130]:
```
array([[4.24399158e-314, 8.48798317e-314],
       [1.27319747e-313, 1.69759663e-313],
       [2.12199579e-313, 2.54639495e-313]])
```

In [132]:
```
np.empty([3,2])
```

Out[132]:
```
array([[4.24399158e-314, 8.48798317e-314],
       [1.27319747e-313, 1.69759663e-313],
       [2.12199579e-313, 2.54639495e-313]])
```

In [134]:
```
a = np.empty([2,2])
a
```

Out[134]:
```
array([[2.12199579e-314, 4.67296746e-307],
       [6.85763116e-321, 3.79442416e-321]])
```

In [135]:
```
a.dtype
```

Out[135]:
```
dtype('float64')
```

In [136]:
```
np.empty([3,2])
```

Out[136]:
```
array([[4.24399158e-314, 8.48798317e-314],
       [1.27319747e-313, 1.69759663e-313],
       [2.12199579e-313, 2.54639495e-313]])
```

In [137]:
```
np.empty([3,3])
```

Out[137]:
```
array([[0.00000000e+000, 0.00000000e+000, 0.00000000e+000],
       [0.00000000e+000, 0.00000000e+000, 1.03358533e-320],
       [1.97345609e-312, 1.37959740e-306, 2.29178686e-312]])
```

In [138]:
```
np.empty([2,3])
```

Out[138]:
```
array([[4.24399158e-314, 8.48798317e-314, 1.27319747e-313],
       [1.69759663e-313, 2.12199579e-313, 2.54639495e-313]])
```

In [139]:
```
np.empty([2, 2])        #uninitialized
```

Out[139]:
```
array([[1.03358533e-320, 1.97345609e-312],
       [1.37959740e-306, 2.29178686e-312]])
```

In [140]:
```
np.empty([3,2],dtype='str')
```

Out[140]:
```
array([['', ''],
       ['', ''],
       ['', '']], dtype='<U1')
```

In [141]:
```
np.empty([3,2],dtype='str').T
```

Out[141]:
```
array([['', '', ''],
       ['', '', '']], dtype='<U1')
```

In [142]:
```
np.empty([3,2],dtype='str').shape
```

Out[142]:
```
(3, 2)
```

In [143]:
```
# Zeros(shape, dtype)
# it Returns new array of Specialized size and type filled with ones
np.zeros([3,2])
# bydefault dtype is float
```

Out[143]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
```

In [144]:
```
np.zeros([2,2],dtype='str')
```

Out[144]:
```
array([['', ''],
       ['', '']], dtype='<U1')
```

In [145]:
```
np.zeros([2,3],dtype='int')
```

Out[145]:
```
array([[0, 0, 0],
       [0, 0, 0]])
```

In [147]:
```
np.zeros([3,2])
```

Out[147]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
```

In [148]:
```
np.zeros(5,int)
```

Out[148]:
```
array([0, 0, 0, 0, 0])
```

In [150]:
```
np.zeros([5,3],dtype = int)
```

Out[150]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

In [151]:
```
np.zeros([5,3],dtype = float)
```

Out[151]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

In [152]:
```
np.zeros([5,3],dtype = str)
```

Out[152]:
```
array([['', '', ''],
       ['', '', ''],
       ['', '', ''],
       ['', '', ''],
       ['', '', '']], dtype='<U1')
```

In [153]:
```
# ones
# it returns new array of specified size and type filled with ones
# np.ones([shape],dtype=)
np.ones([3,3])
```

Out[153]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [154]:
```
np.ones([5,4],dtype='int')
```

Out[154]:
```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

In [159]:
```
np.ones(5,dtype='float')
```

Out[159]:
```
array([1., 1., 1., 1., 1.])
```

In [157]:
```
np.ones(5,dtype='object')
```

Out[157]:
```
array([1, 1, 1, 1, 1], dtype=object)
```

In [158]:
```
np.ones(5,dtype='int')
```

Out[158]:
```
array([1, 1, 1, 1, 1])
```

In [160]:
```
np.ones(5)*5
```

Out[160]:
```
array([5., 5., 5., 5., 5.])
```

In [161]:
```
np.ones([5,5],dtype='str')
```

Out[161]:
```
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']], dtype='<U1')
```

In [162]:
```
# eye()
# it is identity matrix
# it takes square matrix ---> np.eye(3) means 3x3 matrix
# Initialize "ones" on all diagonal position.
# initialize "zeros" on non-diagonal position.
np.eye(2)
```

Out[162]:
```
array([[1., 0.],
       [0., 1.]])
```

In [163]:
```
np.eye(1)
```

Out[163]:
```
array([[1.]])
```

In [164]:
```
np.eye(3)
```

Out[164]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [165]:
```
np.eye(3,dtype='float')
```

Out[165]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [166]:
```
np.eye(4,dtype='float')
```

Out[166]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [167]:
```
np.eye(4,dtype='float')*5
```

Out[167]:
```
array([[5., 0., 0., 0.],
       [0., 5., 0., 0.],
       [0., 0., 5., 0.],
       [0., 0., 0., 5.]])
```

In [168]:
```
np.eye(4,dtype='float')/2
```

Out[168]:
```
array([[0.5, 0. , 0. , 0. ],
       [0. , 0.5, 0. , 0. ],
       [0. , 0. , 0.5, 0. ],
       [0. , 0. , 0. , 0.5]])
```

In [169]:
```
np.eye(5,dtype='float')
```

Out[169]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

In [176]:
```
np.eye(4,dtype='float')+3
```

Out[176]:
```
array([[4., 3., 3., 3.],
       [3., 4., 3., 3.],
       [3., 3., 4., 3.],
       [3., 3., 3., 4.]])
```

In [174]:
```
np.eye(4,dtype='float')
```

Out[174]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [175]:
```
np.eye(5,dtype='float')+1
```

Out[175]:
```
array([[2., 1., 1., 1., 1.],
       [1., 2., 1., 1., 1.],
       [1., 1., 2., 1., 1.],
       [1., 1., 1., 2., 1.],
       [1., 1., 1., 1., 2.]])
```

In [177]:
```
np.eye(5,5)
```

Out[177]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

In [179]:
```
np.eye(5,5)
```

Out[179]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

In [180]:
```
np.eye(5,5)+3
```

Out[180]:
```
array([[4., 3., 3., 3., 3.],
       [3., 4., 3., 3., 3.],
       [3., 3., 4., 3., 3.],
       [3., 3., 3., 4., 3.],
       [3., 3., 3., 3., 4.]])
```

In [181]:
```
np.eye(4,dtype=float)+2
```

Out[181]:
```
array([[3., 2., 2., 2.],
       [2., 3., 2., 2.],
       [2., 2., 3., 2.],
       [2., 2., 2., 3.]])
```

In [182]:
```
# full()
# this function will initailize the elemet in the given values
# np.full(shape,values)
a = np.full([2,2],100)
a
```

Out[182]:
```
array([[100, 100],
       [100, 100]])
```

In [185]:
```
a.dtype
```

Out[185]:
```
dtype('int32')
```

In [186]:
```
np.full(12,100)
```

Out[186]:
```
array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100])
```

In [184]:
```
np.full(12,100).reshape(6,2)
```

Out[184]:
```
array([[100, 100],
       [100, 100],
       [100, 100],
       [100, 100],
       [100, 100],
       [100, 100]])
```

In [187]:
```
np.full([3,4],100)
```

Out[187]:
```
array([[100, 100, 100, 100],
       [100, 100, 100, 100],
       [100, 100, 100, 100]])
```

In [188]:
```
b=np.full([4,5],12,dtype='float')
b
```

Out[188]:
```
array([[12., 12., 12., 12., 12.],
       [12., 12., 12., 12., 12.],
       [12., 12., 12., 12., 12.],
       [12., 12., 12., 12., 12.]])
```

In [189]:
```
b.dtype
```

Out[189]:
```
dtype('float64')
```

In [190]:
```
c= np.full([3,3],12.)
c
```

Out[190]:
```
array([[12., 12., 12.],
       [12., 12., 12.],
       [12., 12., 12.]])
```

In [191]:
```
print(c.dtype)
```

```
float64
```

In [192]:
```
np.diag([1,2,3,4,5])
```

Out[192]:
```
array([[1, 0, 0, 0, 0],
       [0, 2, 0, 0, 0],
       [0, 0, 3, 0, 0],
       [0, 0, 0, 4, 0],
       [0, 0, 0, 0, 5]])
```

In [195]:
```
a = np.diag(np.arange(5))
```

In [196]:
```
# it will return the diagnoanl value which in an array exists
np.diag(a)
```

Out[196]:
```
array([0, 1, 2, 3, 4])
```

In [197]:
```
np.diag([1,2,3,4]).reshape(16,)
```

Out[197]:
```
array([1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 3, 0, 0, 0, 0, 4])
```

In [198]:
```
np.diag([1,2,3,4]).reshape(1,16)
```

Out[198]:
```
array([[1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 3, 0, 0, 0, 0, 4]])
```

In [199]:
```
np.diag([1,2,3,4]).reshape(1,16)
```

Out[199]:
```
array([[1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 3, 0, 0, 0, 0, 4]])
```

In [201]:
```
np.diag(np.arange(1,16))
```

Out[201]:
```
array([[ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  3,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  4,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  5,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  6,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  7,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  8,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  9,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0, 10,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 11,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 12,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 13,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 14,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 15]])
```

In [204]:
```
np.diag(np.arange(1,16)).reshape(225,)
```

Out[204]:
```
array([ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  2,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  3,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  4,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  5,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  6,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  7,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  8,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  9,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0, 10,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0, 11,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0, 12,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0, 13,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0, 14,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0, 15])
```

In [205]:
```
np.diag(np.arange(1,16)).reshape(1,225)
```

Out[205]:
```
array([[ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         3,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         4,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         5,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         6,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         7,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         8,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         9,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        10,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        11,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        12,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        13,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        14,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        15]])
```

In [215]:
```
# randint()
# tihis functionns is used to generate a random number between a goven range
# np.random.randint(min,max,total values)
np.random.randint(1,10,10)
```

Out[215]:
```
array([9, 5, 7, 8, 8, 3, 3, 7, 1, 1])
```

In [227]:
```
np.random.seed(111)
np.random.randint(1,10,9)
```

Out[227]:
```
array([5, 5, 5, 7, 4, 3, 7, 3, 9])
```

In [243]:
```
np.random.seed(111)
np.random.randint(1,10,10)
```

Out[243]:
```
array([5, 5, 5, 7, 4, 3, 7, 3, 9, 8])
```

In [244]:
```
np.random.randint(-20,10,10)
```

Out[244]:
```
array([  5, -13,  -8, -10, -19,  -4,   4,   8,  -6, -18])
```

In [245]:
```
np.random.randint(20,10,10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [245], line 1
----> 1 np.random.randint(20,10,10)

File mtrand.pyx:746, in numpy.random.mtrand.RandomState.randint()

File _bounded_integers.pyx:1338, in numpy.random._bounded_integers._rand_int32()

ValueError: low >= high
```

In [246]:
```
# rand()
# this function is used to generate a random value,
# between 0 to 1
np.random.rand(2,3)
```

Out[246]:
```
array([[0.23772645, 0.08119266, 0.66960024],
       [0.62124292, 0.27425353, 0.46622141]])
```

In [247]:
```
np.random.rand(2,3).T
```

Out[247]:
```
array([[0.11836775, 0.79396256],
       [0.07395756, 0.84056965],
       [0.90077418, 0.81520746]])
```

In [249]:
```
a  = np.random.rand(4,4)
a
```

Out[249]:
```
array([[0.7278491 , 0.32281393, 0.40054555, 0.3189664 ],
       [0.94722183, 0.91870836, 0.81384438, 0.03407709],
       [0.94337287, 0.95043916, 0.80659105, 0.48127804],
       [0.96675923, 0.41697908, 0.31884013, 0.01644282]])
```

In [250]:
```
type(a)
```

Out[250]:
```
numpy.ndarray
```

In [251]:
```
a.dtype
```

Out[251]:
```
dtype('float64')
```

In [252]:
```
a.ndim
```

Out[252]:
```
2
```

In [253]:
```
# randn
# it will genrate a random value closed to 'zero'
# it will return a positive or negative value close to 'zero'
np.random.randn(5)
```

Out[253]:
```
array([ 0.24350846, -1.51720425, -0.43893216, -0.43991633,  0.44882178])
```

In [254]:
```
np.random.randn(2,3)
```

Out[254]:
```
array([[-0.85928929,  1.5594253 , -0.18016486],
       [-0.66653779,  1.91584296, -0.93996453]])
```

In [272]:
```
np.random.seed(111)
np.random.randn(2,4)
```

Out[272]:
```
array([[-1.13383833,  0.38431919,  1.49655378, -0.3553823 ],
       [-0.78753354, -0.45943891, -0.05916877, -0.3541735 ]])
```

In [274]:
```
np.random.randn(4,4)
```

Out[274]:
```
array([[-0.4080822 , -0.06794759, -0.95242666, -0.1106774 ],
       [ 0.57059429,  0.91542017, -1.66934051,  0.48271364],
       [-0.31047267,  2.39468973,  1.55093092, -0.64646518],
       [-0.92893724, -1.65497614,  0.3501925 , -0.14175652]])
```

In [273]:
```
np.random.randn(12,)
```

Out[273]:
```
array([-0.73552305, -1.18393989,  0.23889413, -0.58992026, -1.44058512,
        0.77370311, -1.02796733, -0.09098625,  0.492003  ,  0.4246722 ,
        1.28304882,  0.31598645])
```

In [275]:
```
np.random.randn(1,12)
```

Out[275]:
```
array([[ 0.52108179, -0.02090086, -1.74384395, -0.79915888, -1.30357043,
         0.178105  , -0.33440236, -0.30602713, -0.33240557,  1.96294734,
         0.71924249,  1.14288651]])
```

seed() functions

we know that randint() generates random number a new set of number everytime we run the program a ne w set of random numbers is generated but what if we want to fix this generation of number in this ca se we use 'seed' function

In [276]:
```
np.random.seed(0)
np.random.randint(1,100,10)
```

Out[276]:
```
array([45, 48, 65, 68, 68, 10, 84, 22, 37, 88])
```

In [277]:
```
np.random.seed(0)
np.random.randint(1,100,11)
```

Out[277]:
```
array([45, 48, 65, 68, 68, 10, 84, 22, 37, 88, 71])
```

In [288]:
```
a = np.random.seed(0)
a
np.random.randint(1,100,11)
```

Out[288]:
```
array([45, 48, 65, 68, 68, 10, 84, 22, 37, 88, 71])
```

In [289]:
```
a = np.random.seed(0)
# a
np.random.randint(1,100,11)
```

Out[289]:
```
array([45, 48, 65, 68, 68, 10, 84, 22, 37, 88, 71])
```

In [290]:
```
np.random.seed(1)
np.random.randint(1,3,10)
```

Out[290]:
```
array([2, 2, 1, 1, 2, 2, 2, 2, 2, 1])
```

In [291]:
```
np.random.seed(0)
np.random.randint(1,100,11)
```

Out[291]:
```
array([45, 48, 65, 68, 68, 10, 84, 22, 37, 88, 71])
```

Reshape of an Array

In [301]:
```
a = np.random.randint(1,100,12)
```

In [302]:
```
a
```

Out[302]:
```
array([24, 36, 76, 56, 29, 35,  1,  1, 37, 54,  6, 39])
```

In [303]:
```
a.reshape(4,3)
```

Out[303]:
```
array([[24, 36, 76],
       [56, 29, 35],
       [ 1,  1, 37],
       [54,  6, 39]])
```

In [304]:
```
a.reshape(3,4)
```

Out[304]:
```
array([[24, 36, 76, 56],
       [29, 35,  1,  1],
       [37, 54,  6, 39]])
```

In [305]:
```
a.reshape(12,)
```

Out[305]:
```
array([24, 36, 76, 56, 29, 35,  1,  1, 37, 54,  6, 39])
```

In [306]:
```
a.reshape(1,12)
```

Out[306]:
```
array([[24, 36, 76, 56, 29, 35,  1,  1, 37, 54,  6, 39]])
```

Functions in Numpy

In [314]:
```
# arange()
# it is an array of evenly spaced numbers just like a python's list
# array with numerical python
# np.arange(start,stop, step, dtype)
np.arange(1,20,2)
```

Out[314]:
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])
```

In [316]:
```
np.arange(1,20,2,dtype='float')
```

Out[316]:
```
array([ 1.,  3.,  5.,  7.,  9., 11., 13., 15., 17., 19.])
```

In [317]:
```
np.arange(20,10,-1)
```

Out[317]:
```
array([20, 19, 18, 17, 16, 15, 14, 13, 12, 11])
```

In [318]:
```
np.arange(20,10,-1).reshape(2,5)
```

Out[318]:
```
array([[20, 19, 18, 17, 16],
       [15, 14, 13, 12, 11]])
```

In [319]:
```
np.arange(20,10,-1).reshape(2,5)
```

Out[319]:
```
array([[20, 19, 18, 17, 16],
       [15, 14, 13, 12, 11]])
```

In [320]:
```
np.arange(20,10,-1).reshape(2,5).T
```

Out[320]:
```
array([[20, 15],
       [19, 14],
       [18, 13],
       [17, 12],
       [16, 11]])
```

In [321]:
```
# linspace()
# np.linspace(start,stop,num,endpoint,retstep)
a=np.linspace(10,20)
a
# bydefault it will print 50 no. of obseravtions
```

Out[321]:
```
array([10.        , 10.20408163, 10.40816327, 10.6122449 , 10.81632653,
       11.02040816, 11.2244898 , 11.42857143, 11.63265306, 11.83673469,
       12.04081633, 12.24489796, 12.44897959, 12.65306122, 12.85714286,
       13.06122449, 13.26530612, 13.46938776, 13.67346939, 13.87755102,
       14.08163265, 14.28571429, 14.48979592, 14.69387755, 14.89795918,
       15.10204082, 15.30612245, 15.51020408, 15.71428571, 15.91836735,
       16.12244898, 16.32653061, 16.53061224, 16.73469388, 16.93877551,
       17.14285714, 17.34693878, 17.55102041, 17.75510204, 17.95918367,
       18.16326531, 18.36734694, 18.57142857, 18.7755102 , 18.97959184,
       19.18367347, 19.3877551 , 19.59183673, 19.79591837, 20.        ])
```

In [322]:
```
a.size
```

Out[322]:
```
50
```

In [323]:
```
np.linspace(10,20,num=20)
```

Out[323]:
```
array([10.        , 10.52631579, 11.05263158, 11.57894737, 12.10526316,
       12.63157895, 13.15789474, 13.68421053, 14.21052632, 14.73684211,
       15.26315789, 15.78947368, 16.31578947, 16.84210526, 17.36842105,
       17.89473684, 18.42105263, 18.94736842, 19.47368421, 20.        ])
```

In [324]:
```
np.linspace(10,20,num=20).size
```

Out[324]:
```
20
```

In [325]:
```
np.linspace(10,20,num=20,endpoint=False)
# endpoint will not consider the last element which was 20 in this case
```

Out[325]:
```
array([10. , 10.5, 11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. ,
       15.5, 16. , 16.5, 17. , 17.5, 18. , 18.5, 19. , 19.5])
```

In [326]:
```
np.linspace(10,20,num=20,endpoint=False,retstep=True)
# restep will let you know what is the increment size betn the obseravtions
```

Out[326]:
```
(array([10. , 10.5, 11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. ,
        15.5, 16. , 16.5, 17. , 17.5, 18. , 18.5, 19. , 19.5]),
 0.5)
```

In [327]:
```
10/20
```

Out[327]:
```
0.5
```

In [328]:
```
np.linspace(10,20,num=10,endpoint=False,retstep=True)
# restep will let you know what is the increment size betn the obseravtions
```

Out[328]:
```
(array([10., 11., 12., 13., 14., 15., 16., 17., 18., 19.]), 1.0)
```

In [329]:
```
10/10
```

Out[329]:
```
1.0
```

In [330]:
```
# logspace
# logspace doesn't have retstep
np.logspace(10,20,num=20)
# it will give us logarithm values
# base bydefault is 10
# # bydefault it will print 50 no. of obseravtions
```

Out[330]:
```
array([1.00000000e+10, 3.35981829e+10, 1.12883789e+11, 3.79269019e+11,
       1.27427499e+12, 4.28133240e+12, 1.43844989e+13, 4.83293024e+13,
       1.62377674e+14, 5.45559478e+14, 1.83298071e+15, 6.15848211e+15,
       2.06913808e+16, 6.95192796e+16, 2.33572147e+17, 7.84759970e+17,
       2.63665090e+18, 8.85866790e+18, 2.97635144e+19, 1.00000000e+20])
```

In [331]:
```
# lets change the base
np.logspace(10,20,num=20,base=3)
```

Out[331]:
```
array([5.90490000e+04, 1.05275908e+05, 1.87691864e+05, 3.34627706e+05,
       5.96593265e+05, 1.06364033e+06, 1.89631834e+06, 3.38086396e+06,
       6.02759614e+06, 1.07463405e+07, 1.91591857e+07, 3.41580835e+07,
       6.08989697e+07, 1.08574139e+08, 1.93572137e+08, 3.45111392e+08,
       6.15284174e+08, 1.09696354e+09, 1.95572886e+09, 3.48678440e+09])
```

In [332]:
```
# lets try to save an array
a=np.arange(1,7).reshape(2,3)
```

In [333]:
```
a
```

Out[333]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [334]:
```
# lets save this array
np.savetxt('arrray.txt',a)
```

In [335]:
```

# let read the array file which we saved in a txt format
y = np.loadtxt('arrray.txt')
y
# the dtype is 'float'
```

Out[335]:
```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

In [336]:
```

# lets change the dtype
y = np.loadtxt('arrray.txt',dtype='int')
y
```

```
C:\Users\HRISHIKESH\AppData\Local\Temp\ipykernel_13204\1743094509.py:2: DeprecationWarning: loadtxt(): Parsi
ng an integer via a float is deprecated.  To avoid this warning, you can:
    * make sure the original data is stored as integers.
    * use the `converters=` keyword argument.  If you only use
      NumPy 1.23 or later, `converters=float` will normally work.
    * Use `np.loadtxt(...).astype(np.int64)` parsing the file as
      floating point and then convert it.  (On all NumPy versions.)
  (Deprecated NumPy 1.23)
  y = np.loadtxt('arrray.txt',dtype='int')
```

Out[336]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [ ]:
```

```

Slicing in Numpy - Chapter 3 Practice 1

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

1D Array

In [2]:
```
a = np.array([10,20,30,40,50,60])
a
```

Out[2]: array([10, 20, 30, 40, 50, 60])

In [3]:
```
a[::]
```

Out[3]: array([10, 20, 30, 40, 50, 60])

In [4]:
```
a[0:6:1]
```

Out[4]: array([10, 20, 30, 40, 50, 60])

In [5]:
```
a[0:]
```

Out[5]: array([10, 20, 30, 40, 50, 60])

In [6]:
```
a[::-1]
```

Out[6]: array([60, 50, 40, 30, 20, 10])

In [7]:
```
a[0:6:2]
```

Out[7]: array([10, 30, 50])

In [8]:
```
a[1::2]
```

Out[8]: array([20, 40, 60])

In [9]:
```
a[1:6:2]
```

Out[9]: array([20, 40, 60])

In [10]:
```
a[:4]
```

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

In [11]:
```
a[2:]
```

Out[11]: array([30, 40, 50, 60])

In [12]:
```
a[::2]
```

Out[12]: array([10, 30, 50])

In [13]:
```
a[-1::1]
```

Out[13]: array([60])

In [14]:
```
a[-1::-1]
```

Out[14]: array([60, 50, 40, 30, 20, 10])

In [15]:
```
a[::-1]
```

Out[15]: array([60, 50, 40, 30, 20, 10])

In [17]:
```
a[::3]
```

Out[17]: array([10, 40])

In [18]:
```
a[::4]
```

Out[18]: array([10, 50])

In [19]:
```
a[::5]
```

Out[19]:
```
array([10, 60])
```

In [20]:
```
a[::6]
```

Out[20]:
```
array([10])
```

2D

In [21]:
```
b = np.array([[10,20,30],[40,50,60]])
b
```

Out[21]:
```
array([[10, 20, 30],
       [40, 50, 60]])
```

In [22]:
```
b[1,1]
```

Out[22]:
```
50
```

In [23]:
```
# if you see the abouve output it just giving the output
# but not in the form of an array
# if i try the same abpve output in array form
b[1:2,1:2]
# here i did the slicing and due to that we got the output in,
# array form
```

Out[23]:
```
array([[50]])
```

In [27]:
```
b[:,1:]
```

Out[27]:
```
array([[20, 30],
       [50, 60]])
```

In [28]:
```
b[0:2,0:2]
```

Out[28]:
```
array([[10, 20],
       [40, 50]])
```

In [29]:
```
b[0:2,0:3]
```

Out[29]:
```
array([[10, 20, 30],
       [40, 50, 60]])
```

In [30]:
```
b= np.array([[10,20,30],[40,50,60],[70,80,90],[100,110,120]])
b
```

Out[30]:
```
array([[ 10,  20,  30],
       [ 40,  50,  60],
       [ 70,  80,  90],
       [100, 110, 120]])
```

In [31]:
```
b[2,1]
```

Out[31]:
```
80
```

In [32]:
```
b[2,1:2]
```

Out[32]:
```
array([80])
```

In [37]:
```
b[2:3,1:2]
```

Out[37]:
```
array([[80]])
```

In [38]:
```
b[0,1:2]
```

Out[38]:
```
array([20])
```

Arithmatic Operator

In [39]:
```
a = np.array([[1,2,3],[4,5,6],[7,8,9]])

b = np.array([[1,1,1],[1,1,1],[1,1,1]])
```

In [40]:
```
a
```

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

In [41]:
```
b
```

Out[41]:
```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

In [43]:
```
np.shape(a)
```

Out[43]:
```
(3, 3)
```

In [44]:
```
np.shape(a)
```

Out[44]:
```
(3, 3)
```

In [45]:
```
np.shares_memory(a,b)
```

Out[45]:
```
False
```

In [48]:
```
a.shape
```

Out[48]:
```
(3, 3)
```

In [49]:
```
b.shape
```

Out[49]:
```
(3, 3)
```

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

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

In [52]:
```
b
```

Out[52]:
```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

In [50]:
```
# Addition
np.add(a,b)
```

Out[50]:
```
array([[ 2,  3,  4],
       [ 5,  6,  7],
       [ 8,  9, 10]])
```

In [53]:
```
np.add(b,a)
```

Out[53]:
```
array([[ 2,  3,  4],
       [ 5,  6,  7],
       [ 8,  9, 10]])
```

In [54]:
```
c = np.array([10,20,30])
c
```

Out[54]:
```
array([10, 20, 30])
```

In [55]:
```
a
```

Out[55]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [56]:
```
b
```

Out[56]:
```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

In [57]:
```
np.add(a,c)
```

Out[57]:
```
array([[11, 22, 33],
       [14, 25, 36],
       [17, 28, 39]])
```

In [60]:
```
np.add(c,a)
```

Out[60]:
```
array([[11, 22, 33],
       [14, 25, 36],
       [17, 28, 39]])
```

In [59]:
```
np.add(b,c)
```

Out[59]:
```
array([[11, 21, 31],
       [11, 21, 31],
       [11, 21, 31]])
```

In [61]:
```
np.add(c,b)
```

Out[61]:
```
array([[11, 21, 31],
       [11, 21, 31],
       [11, 21, 31]])
```

In [63]:
```
a
```

Out[63]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [64]:
```
c
```

Out[64]:
```
array([10, 20, 30])
```

In [62]:
```
np.subtract(a,c)
```

Out[62]:
```
array([[ -9, -18, -27],
       [ -6, -15, -24],
       [ -3, -12, -21]])
```

In [65]:
```
np.divide(a,b)
```

Out[65]:
```
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])
```

In [66]:
```
np.mod(c,a)
```

Out[66]:
```
array([[0, 0, 0],
       [2, 0, 0],
       [3, 4, 3]])
```

View And Copy - Chapter 4

- when we slice a subarray from an array it may be done by two Array
  - 1. it is making view
  - 2. Using a copy of it
- But there is some difference between both of them.
- if you make a view of an array any changes made to the array will be replicated to Original Array,
- whereas if you make a copy of it, and make any changes to copied subarray will bot reflect on the Original Array.
- Bydefault an Array is set on View.

View

In [67]:
```
import numpy as np
```

In [68]:
```
a = np.array([10,20,30,40,50,60,70,80,90])
a
```

Out[68]:
```
array([10, 20, 30, 40, 50, 60, 70, 80, 90])
```

In [69]:
```
slic = a[3:6]
```

In [70]:
```
slic
```

Out[70]:
```
array([40, 50, 60])
```

In [72]:
```
# lets try to change the value of variable 'slicing'
slic[:] = 0
slic
```

Out[72]:
```
array([0, 0, 0])
```

In [73]:
```
# let try to print the Original Array
# to see that if it has effected our original array or not
a
# and it has did the changes oin the Original Array
```

Out[73]:
```
array([10, 20, 30,  0,  0,  0, 70, 80, 90])
```

Copy

In [79]:
```
b = np.array([10,20,30,40,50,60,70,80,90])
b
```

Out[79]:
```
array([10, 20, 30, 40, 50, 60, 70, 80, 90])
```

In [80]:
```
slic = b[3:6].copy()
```

In [81]:
```
slic
```

Out[81]:
```
array([40, 50, 60])
```

In [82]:
```
slic[:]=0
```

In [83]:
```
slic
```

Out[83]:
```
array([0, 0, 0])
```

In [1]:
```
b
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [1], line 1
----> 1 b

NameError: name 'b' is not defined
```

In [ ]:

In [2]:
```
b[:1:]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [2], line 1
----> 1 b[:1:]

NameError: name 'b' is not defined
```

In [26]:
```
b[:2:]
```

Out[26]:
```
array([[10, 20, 30],
       [40, 50, 60]])
```

Numpy Array Selection

This is called process Masking.

In [85]:
```
a = np.array([1,2,3,4,5,6,7,8,9,10])
a
```

Out[85]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [86]:
```
a>4
```

Out[86]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True])
```

In [87]:
```
mask = a>4
mask
```

Out[87]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True])
```

In [88]:
```
a[mask]
```

Out[88]:
```
array([ 5,  6,  7,  8,  9, 10])
```

In [89]:
```
a[a>4]
```

Out[89]:
```
array([ 5,  6,  7,  8,  9, 10])
```

In [90]:
```
a[a>3]
```

Out[90]:
```
array([ 4,  5,  6,  7,  8,  9, 10])
```

In [91]:
```
a[a>=4]
```

Out[91]:
```
array([ 4,  5,  6,  7,  8,  9, 10])
```

In [92]:
```
a[a>=4].shape
```

Out[92]:
```
(7,)
```

In [93]:
```
a[a>=4].reshape(1,7)
```

Out[93]:
```
array([[ 4,  5,  6,  7,  8,  9, 10]])
```

In [94]:
```
name = np.array(["rishi","rohit","aditya","sanjil","nishant"])
name
```

Out[94]:
```
array(['rishi', 'rohit', 'aditya', 'sanjil', 'nishant'], dtype='<U7')
```

In [95]:
```
x = 'rishi'
name == x
```

Out[95]:
```
array([ True, False, False, False, False])
```

In [96]:
```
name[name==x]
```

Out[96]:
```
array(['rishi'], dtype='<U7')
```

In [97]:
```
name!=x
```

Out[97]:
```
array([False,  True,  True,  True,  True])
```

In [98]:
```
name[name!=x]
```

Out[98]:
```
array(['rohit', 'aditya', 'sanjil', 'nishant'], dtype='<U7')
```

In [99]:
```
name = np.array(["rishi","rohit","aditya","sanjil","nishant","rishi","rishi","sanjil"])
name
```

Out[99]:
```
array(['rishi', 'rohit', 'aditya', 'sanjil', 'nishant', 'rishi', 'rishi',
       'sanjil'], dtype='<U7')
```

In [100]:
```
(name=='rishi') | (name=='sanjil')
```

Out[100]:
```
array([ True, False, False,  True, False,  True,  True,  True])
```

In [101]:
```
name[(name=='rishi') | (name=='sanjil')]
```

Out[101]:
```
array(['rishi', 'sanjil', 'rishi', 'rishi', 'sanjil'], dtype='<U7')
```

In [102]:
```
# select all the values which is not rishi and
# assign those value as 'Javed'
name
```

Out[102]:
```
array(['rishi', 'rohit', 'aditya', 'sanjil', 'nishant', 'rishi', 'rishi',
       'sanjil'], dtype='<U7')
```

In [103]:
```
name[name!='rishi']='javed'
```

In [104]:
```
name
```

Out[104]:
```
array(['rishi', 'javed', 'javed', 'javed', 'javed', 'rishi', 'rishi',
       'javed'], dtype='<U7')
```

In [105]:
```
a = np.array([27,4,1,5,7,39])
a
```

Out[105]:
```
array([27,  4,  1,  5,  7, 39])
```

In [106]:
```
np.min(a)
```

Out[106]: 1

In [107]:
```
a.min()
```

Out[107]: 1

In [108]:
```
a.max()
```

Out[108]: 39

In [110]:
```
np.max(a)
```

Out[110]: 39

In [111]:
```
np.argmin(a)
```

Out[111]: 2

In [113]:
```
a.argmin()
```

Out[113]: 2

In [114]:
```
a.argmax()
```

Out[114]: 5

In [115]:
```
np.sqrt(a)

np.sin(a)

np.cos(a)

np.quantile(a,0.25)
# Quantile 1 - 25%

np.quantile(a,0.75)
# Quantile 3 - 75%

np.quantile(a,0.50)
# Quantile 2 - 50% Median
```

Out[115]: 6.0

In [116]:
```
np.sqrt(a)
```

Out[116]:
```
array([5.19615242, 2.        , 1.        , 2.23606798, 2.64575131,
       6.244998  ])
```

In [117]:
```
np.sin(a)
```

Out[117]:
```
array([ 0.95637593, -0.7568025 ,  0.84147098, -0.95892427,  0.6569866 ,
        0.96379539])
```

In [118]:
```
np.cos(a)
```

Out[118]:
```
array([-0.29213881, -0.65364362,  0.54030231,  0.28366219,  0.75390225,
        0.26664293])
```

In [119]:
```
np.quantile(a,0.25)
# Quantile 1 - 25%
```

Out[119]: 4.25

In [120]:
```
np.quantile(a,0.75)
# Quantile 3 - 75%
```

Out[120]: 22.0

In [121]:
```
np.quantile(a,0.50)
# Quantile 2 - 50% Median
```

Out[121]: 6.0

In [ ]:
```

```

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

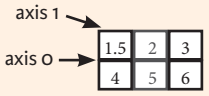
Use the following import convention:

```
>>> import numpy as np
```

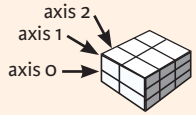
NumPy Arrays

1D array

2D array



3D array



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                    Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)    Create an array of ones
>>> d = np.arange(10,25,5)             Create an array of evenly
                                       spaced values (step value)
>>> np.linspace(0,2,9)                 Create an array of evenly
                                       spaced values (number of samples)
>>> e = np.full((2,2),7)               Create a constant array
>>> f = np.eye(2)                      Create a 2X2 identity matrix
>>> np.random.random((2,2))            Create an array with random values
>>> np.empty((3,2))                    Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64       Signed 64-bit integer types
>>> np.float32     Standard double-precision floating point
>>> np.complex     Complex numbers represented by 128 floats
>>> np.bool        Boolean type storing TRUE and FALSE values
>>> np.object      Python object type
>>> np.string_     Fixed-length string type
>>> np.unicode_    Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape             Array dimensions
>>> len(a)              Length of array
>>> b.ndim              Number of array dimensions
>>> e.size              Number of array elements
>>> b.dtype             Data type of array elements
>>> b.dtype.name        Name of data type
>>> b.astype(int)       Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                          Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                   Subtraction
>>> b + a                              Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                        Addition
>>> a / b                              Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                     Division
>>> a * b                              Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                   Multiplication
>>> np.exp(b)                          Exponentiation
>>> np.sqrt(b)                         Square root
>>> np.sin(a)                          Print sines of an array
>>> np.cos(b)                          Element-wise cosine
>>> np.log(a)                          Element-wise natural logarithm
>>> e.dot(f)                           Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                             Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                              Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)               Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                            Array-wise sum
>>> a.min()                            Array-wise minimum value
>>> b.max(axis=0)                      Maximum value of an array row
>>> b.cumsum(axis=1)                   Cumulative sum of the elements
>>> a.mean()                           Mean
>>> b.median()                         Median
>>> a.corrcoef()                       Correlation coefficient
>>> np.std(b)                          Standard deviation
```

Copying Arrays

```
>>> h = a.view()          Create a view of the array with the same data
>>> np.copy(a)            Create a copy of the array
>>> h = a.copy()          Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()              Sort an array
>>> c.sort(axis=0)        Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting


```
>>> a[2]
  3
```

Select the element at the 2nd index


```
>>> b[1,2]
  6.0
```

Select the element at row 1 column 2 (equivalent to `b[1][2]`)

Slicing


```
>>> a[0:2]
  array([1, 2])
```

Select items at index 0 and 1


```
>>> b[0:2,1]
  array([ 2.,  5.])
```

Select items at rows 0 and 1 in column 1


```
>>> b[:1]
  array([[1.5, 2., 3.]])
```

Select all items at row 0 (equivalent to `b[0:1, :]`)

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

Same as `[1,:,:]`

```
>>> a[ : :-1]
  array([3, 2, 1])
```

Reversed array `a`

Boolean Indexing


```
>>> a[a<2]
  array([1])
```

Select elements from `a` less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
  array([ 4. , 2. , 6. , 1.5])
```

Select elements $(1,0)$, $(0,1)$, $(1,2)$ and $(0,0)$

```
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
  array([[ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5],
         [ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5]])
```

Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)        Permute array dimensions
>>> i.T                        Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                  Flatten the array
>>> g.reshape(3,-2)            Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))            Return a new array with shape (2,6)
>>> np.append(h,g)             Append items to an array
>>> np.insert(a, 1, 5)         Insert items in an array
>>> np.delete(a,[1])           Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)        Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                    Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                          Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                    Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))              Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                          Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                      Split the array horizontally at the 3rd index
  [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)                      Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

DataCamp
Learn Python for Data Science Interactively