In [4]:
```
# importing library
import numpy as np
```

In [6]:
```
a = np.arange(15).reshape(3, 5)
a.shape
```

Out[6]:
```
(3, 5)
```

In [7]:
```
a.ndim
```

Out[7]:
```
2
```

In [8]:
```
a.dtype.name
```

Out[8]:
```
'int32'
```

In [9]:
```
a.itemsize
```

Out[9]:
```
4
```

In [10]:
```
a.size
```

Out[10]:
```
15
```

In [11]:
```
type(a)
```

Out[11]:
```
numpy.ndarray
```

In [14]:
```
b = np.array([(1.5,2,3), (4,5,6)])
b
```

Out[14]:
```
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
```

In [15]:
```
c = np.array( [ [1,2], [3,4] ], dtype=complex )
c
```

Out[15]:
```
array([[1.+0.j, 2.+0.j],
       [3.+0.j, 4.+0.j]])
```

In [16]:
```
np.zeros((3, 4))
```

Out[16]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [17]:
```
np.ones( (2,3,4), dtype=np.int16 )                # dtype can also be specified
```

Out[17]:
```
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]]], dtype=int16)
```

In [18]:
```
np.empty( (2,3) )                                 # uninitialized
```

Out[18]:
```
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
```

In [19]:
```
np.arange( 10, 30, 5 )
```

Out[19]:
```
array([10, 15, 20, 25])
```

In [20]:
```
np.arange( 0, 2, 0.3 )                 # it accepts float arguments
```

Out[20]:
```
array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
```

In [22]:
```
from numpy import pi
np.linspace( 0, 2, 9 )                 # 9 numbers from 0 to 2
```

Out[22]:
```
array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
```

In [23]:
```
x = np.linspace( 0, 2*pi, 100 )        # useful to evaluate function at lots of points
```

In [25]:
```
a = np.arange(6)                         # 1d array
a
```

Out[25]:
```
array([0, 1, 2, 3, 4, 5])
```

In [26]:
```
b = np.arange(12).reshape(4,3)           # 2d array
b
```

Out[26]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [27]:
```
c = np.arange(24).reshape(2,3,4)         # 3d array
```

In [31]:
```
#Basic Operations
a = np.array( [20,30,40,50] )
b = np.arange( 4 )
b
```

Out[31]:
```
array([0, 1, 2, 3])
```

In [33]:
```
c = a-b
c
```

Out[33]:
```
array([20, 29, 38, 47])
```

In [34]:
```
b**2
```

Out[34]:
```
array([0, 1, 4, 9], dtype=int32)
```

In [35]:
```
10*np.sin(a)
```

Out[35]:
```
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
```

In [36]:
```
a<35
```

Out[36]:
```
array([ True,  True, False, False])
```

In [37]:
```
A = np.array( [[1,1],
               [0,1]] )
```

In [38]:
```
B = np.array( [[2,0],
               [3,4]] )
```

In [39]:
```
A * B                     # elementwise product
```

Out[39]:
```
array([[2, 0],
       [0, 4]])
```

In [40]:
```
A @ B                     # matrix product
```

Out[40]:
```
array([[5, 4],
       [3, 4]])
```

In [41]:
```
A.dot(B)                  # another matrix product
```

Out[41]:
```
array([[5, 4],
       [3, 4]])
```

In [42]:
```
rg = np.random.default_rng(1)     # create instance of default random number g
enerator
```

In [43]:
```
b = rg.random((2,3))
a *= 3
a
```

Out[43]:
```
array([ 60,  90, 120, 150])
```

In [46]:
```
# Indexing, Slicing and Iterating
a = np.arange(10)**3
a
```

Out[46]:
```
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729], dtype=int32)
```

In [47]:
```
a[2]
```

Out[47]:
```
8
```

In [48]:
```
a[2:5]
```

Out[48]:
```
array([ 8, 27, 64], dtype=int32)
```

In [49]:
```
# from start to position 6, exclusive, set every 2nd element to 1000
a[:6:2] = 1000
a
```

Out[49]:
```
array([1000,    1, 1000,   27, 1000,  125,  216,  343,  512,  729],
      dtype=int32)
```

In [50]:
```
# Shape Manipulation
a = np.floor(10*rg.random((3,4)))
a
```

Out[50]:
```
array([[8., 4., 5., 0.],
       [7., 5., 3., 7.],
       [3., 4., 1., 4.]])
```

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

Out[51]:
```
(3, 4)
```

In [52]:
```
a.ravel()  # returns the array, flattened
a
```

Out[52]:
```
array([[8., 4., 5., 0.],
       [7., 5., 3., 7.],
       [3., 4., 1., 4.]])
```

In [53]:
```
a.reshape(6,2)  # returns the array with a modified shape
a
```

Out[53]:
```
array([[8., 4., 5., 0.],
       [7., 5., 3., 7.],
       [3., 4., 1., 4.]])
```

In [54]:
```
a.T  # returns the array, transposed
a
```

Out[54]:
```
array([[8., 4., 5., 0.],
       [7., 5., 3., 7.],
       [3., 4., 1., 4.]])
```

In [55]:
```
# Stacking together different arrays
a = np.floor(10*rg.random((2,2)))
a
```

Out[55]:
```
array([[2., 2.],
       [7., 2.]])
```

In [56]:
```
b = np.floor(10*rg.random((2,2)))
b
```

Out[56]:
```
array([[4., 9.],
       [9., 7.]])
```

In [57]:
```
np.vstack((a,b))
```

Out[57]:
```
array([[2., 2.],
       [7., 2.],
       [4., 9.],
       [9., 7.]])
```

In [58]:
```
np.hstack((a,b))
```

Out[58]:
```
array([[2., 2., 4., 9.],
       [7., 2., 9., 7.]])
```

In [59]:
```
# View or Shallow Copy
c = a.view()
c is a
```

Out[59]:
```
False
```

In [60]:
```
c.base is a
```

Out[60]:
```
True
```

In [61]:
```
c.flags.owndata
```

Out[61]:
```
False
```

In [68]:
```
# Deep Copy
d = a.copy()              # a new array object with new data is created
d is a
```

Out[68]:
```
False
```

In [69]:
```
d.base is a
```

Out[69]:
```
False
```

In [ ]:
```

```