NUMPY

```
NumPy’s main object is the homogeneous multidimensional array. It is a
table of elements (usually numbers), all of the same type, indexed by a
tuple of non-negative integers. In NumPy dimensions are called axes.
```

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

In [2]:
```
a = np.arange(15).reshape(3, 5)
a
```

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

In [3]:
```
# ndarray.ndim
# the number of axes (dimensions) of the array.
a.ndim
```

Out[3]:
```
2
```

In [4]:
```
#  This is a tuple of integers indicating the size of the array in each dime
a.shape
```

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

In [5]:
```
# the total number of elements of the array.
a.size
```

Out[5]:
```
15
```

In [6]:
```
# an object describing the type of the elements in the array.
a.dtype
```

Out[6]:
```
dtype('int32')
```

In [7]:
```
# the size in bytes of each element of the array.
a.itemsize
```

Out[7]:
```
4
```

In [8]:
```
# the buffer containing the actual elements of the array
a.data
```

Out[8]:
```
<memory at 0x00000150B0EA4E10>
```

In [9]:
```
type(a)
```

Out[9]:
```
numpy.ndarray
```

In [10]:
```
x=np.array([[1,3],[3,4]],dtype=float)
x
```

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

In [11]:
```
x=np.array([[1,3],[3,4]],dtype=complex)
x
```

Out[11]:
```
array([[1.+0.j, 3.+0.j],
       [3.+0.j, 4.+0.j]])
```

```
The function zeros creates an array full of zeros, the function ones
creates an array full of ones, and the function empty creates an array
whose initial content is random and depends on the state of the memory. By
default, the dtype of the created array is float64, but it can be
specified via the key word argument dtype.
```

In [12]:
```
np.zeros((3, 4))
```

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

In [13]:
```
np.ones((2,5))
```

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

In [14]:
```
np.empty((2,4))
```

Out[14]:
```
array([[4.67296746e-307, 1.69121096e-306, 1.29061142e-306,
        1.89146896e-307],
       [7.56571288e-307, 3.11525958e-307, 1.24610723e-306,
        1.29061142e-306]])
```

```
To create sequences of numbers, NumPy provides the arange function which
is analogous to the Python built-in range, but returns an array.
```

In [15]:
```
np.arange(10,50,2)
```

Out[15]:
```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
       44, 46, 48])
```

Basic Operations

In [16]:
```
a=np.array([2,3,4,5,7])
b=np.ones((1,5))
print(a-b)
```

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

In [17]:
```
a*5
```

Out[17]:
```
array([10, 15, 20, 25, 35])
```

In [18]:
```
a**2
```

Out[18]:
```
array([ 4,  9, 16, 25, 49])
```

In [19]:
```
a>3
```

Out[19]:
```
array([False, False,  True,  True,  True])
```

In [20]:
```
A = np.array([[1, 1],
              [0, 1]])
B = np.array([[2, 0],
              [3, 4]])
```

In [21]:
```
# elementwise product
A*B
```

Out[21]:
```
array([[2, 0],
       [0, 4]])
```

In [22]:
```
# matric product
A@B
```

Out[22]:
```
array([[5, 4],
       [3, 4]])
```

In [23]:
```
A.dot(B)
```

Out[23]:
```
array([[5, 4],
       [3, 4]])
```

In [24]:
```
A.sum()
```

Out[24]:
```
3
```

In [25]:
```
A.min()
```

Out[25]:
```
0
```

In [26]:
```
A.max()
```

Out[26]:
```
1
```

In [27]:
```
A.mean()
```

Out[27]:
```
0.75
```

In [28]:
```
A.transpose()
```

Out[28]:
```
array([[1, 0],
       [1, 1]])
```

In [64]:
```
def f(x,y):
    return x+y;
x=np.fromfunction(f,(3,3),dtype="int64")
x
```

Out[64]:
```
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]], dtype=int64)
```

In [29]:
```
A.sort()
A
```

Out[29]:
```
array([[1, 1],
       [0, 1]])
```

In [54]:
```
A.cumsum() # cumulative sum of elements
```

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

In [31]:
```
A.sum(axis=1)
```

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

In [55]:
```
a.cumprod() # cumulative product of elements
```

Out[55]:
```
array([  2,   6,  24, 120, 840])
```

In [56]:
```
#np.compress(condition, a, axis=None, out=None)
np.compress(a>3,a)  #  a slice along that axis is returned in output for eac
```

Out[56]:
```
array([4, 5, 7])
```

In [57]:
```
# np.extract(condition, arr)
np.extract(a>2,a)  # compress is equivalent to extract.
```

Out[57]:
```
array([3, 4, 5, 7])
```

Universal Functions

In [32]:
```
s=np.arange(4)
s
```

Out[32]:
```
array([0, 1, 2, 3])
```

In [33]:
```
np.exp(A)
```

Out[33]:
```
array([[2.71828183, 2.71828183],
       [1.        , 2.71828183]])
```

In [35]:
```
np.log(B)
```

```
C:\Users\heman\AppData\Local\Temp\ipykernel_27544\4061543246.py:1: RuntimeW
arning: divide by zero encountered in log
  np.log(B)
```

Out[35]:
```
array([[0.69314718,       -inf],
       [1.09861229, 1.38629436]])
```

In [37]:
```
np.sqrt(49)
```

Out[37]:
```
7.0
```

Statistics

In [38]:
```
A.mean()
```

Out[38]:
```
0.75
```

In [39]:
```
A.std()  #standard deviation
```

Out[39]:
```
0.4330127018922193
```

In [40]:
```
A.var() #variance
```

Out[40]:
```
0.1875
```

In [44]:
```
np.cov(A)  #covariance
```

Out[44]:
```
array([[0. , 0. ],
       [0. , 0.5]])
```

Questions

In [47]:
```
A.all()
```

Out[47]:
```
False
```

In [48]:
```
A.any()
```

Out[48]:
```
True
```

In [49]:
```
A.nonzero()
```

Out[49]:
```
(array([0, 0, 1], dtype=int64), array([0, 1, 1], dtype=int64))
```

In [51]:
```
np.where(a>3)
```

Out[51]:
```
(array([2, 3, 4], dtype=int64),)
```

Indexing, Slicing

In [58]:
```
a[:3]
```

Out[58]:
```
array([2, 3, 4])
```

In [59]:
```
a[::-1]
```

Out[59]:
```
array([7, 5, 4, 3, 2])
```

In [60]:
```
w=np.array([[1,2,3],[5,6,7],[9,7,6]])
w[:,2] #all row and third col
```

Out[60]:
```
array([3, 7, 6])
```

Shape

In [67]:
```
w.shape
```

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

In [69]:
```
 w.ravel()   # returns the array, flattened
```

Out[69]:
```
array([1, 2, 3, 5, 6, 7, 9, 7, 6])
```

In [72]:
```
w.reshape(9,1)
```

Out[72]:
```
array([[1],
       [2],
       [3],
       [5],
       [6],
       [7],
       [9],
       [7],
       [6]])
```

```
The reshape function returns its argument with a modified shape, whereas
the ndarray.resize method modifies the array itself
```

In [75]:
```
w.resize(1,9)
w
```

Out[75]:
```
array([[1, 2, 3, 5, 6, 7, 9, 7, 6]])
```