NUMPY_3rdFeb

February 5, 2025

1 3rd Feb

2 NumPy

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

[2]:
```
range(5)
```

[2]:
```
range(0, 5)
```

[3]:
```
list(range(5))
```

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

[4]:
```
range(10,40)
```

[4]:
```
range(10, 40)
```

[5]:
```
range(10,40,5)
```

[5]:
```
range(10, 40, 5)
```

[6]:
```
r = list(range(10,40,5))
```

[7]:
```
r
```

[7]:
```
[10, 15, 20, 25, 30, 35]
```

[8]:
```
np.__version__
```

[8]:
```
'1.26.4'
```

[9]:
```
import sys
sys.version
```

[9]:
```
'3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [MSC v.1916
64 bit (AMD64)]'
```

[10]:
```
l = [1,2,3,4,5]
arr = np.array(l)
arr
```

[10]:
```
array([1, 2, 3, 4, 5])
```

[11]:
```
type(arr)
```

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

2.1 arange

[12]:
```
np.arange(15)
```

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

[13]:
```
np.arange(3.0)
```

[13]:
```
array([0., 1., 2.])
```

[14]:
```
np.arange(5,20)
```

[14]:
```
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

[15]:
```
np.arange(5,50,5)
```

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

[16]:
```
np.arange(50,20)
```

[16]:
```
array([], dtype=int32)
```

[17]:
```
np.arange(-20,10)
```

[17]:
```
array([-20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10,  -9,  -8,
        -7,  -6,  -5,  -4,  -3,  -2,  -1,   0,   1,   2,   3,   4,   5,
         6,   7,   8,   9])
```

[18]:
```
ar = np.arange(-20,-2)
ar
```

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

2.2 Zeros and Ones

[19]:
```
np.zeros(5)
```

[19]:
```
array([0., 0., 0., 0., 0.])
```

[20]:
```
np.zeros(5,dtype=int)
```

[20]:
```
array([0, 0, 0, 0, 0])
```

[21]:
```
np.zeros((5,5),dtype=int)
```

[21]:
```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

[22]:
```
np.zeros([5,2])
```

[22]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

[23]:
```
np.zeros((2,10))
```

[23]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

[24]:
```
np.ones(5)
```

[24]:
```
array([1., 1., 1., 1., 1.])
```

[25]:
```
np.ones(5,dtype=int)
```

[25]:
```
array([1, 1, 1, 1, 1])
```

[26]:
```
np.ones(5)
```

[26]:
```
array([1., 1., 1., 1., 1.])
```

2.3 Random Function

[27]:
```
np.random.rand(3,2)
```

[27]:
```
array([[0.39397673, 0.54099012],
       [0.67773164, 0.66637917],
       [0.3250942 , 0.16933803]])
```

[28]:
```
np.random.randint(1,6)
```

[28]: 2

[29]:
```
np.random.randint(1,6,6)
```

[29]: array([5, 4, 5, 5, 1, 5])

[30]:
```
np.random.randint(2,6)
```

[30]: 4

[31]:
```
np.random.randint(2,6)
```

[31]: 3

[32]:
```
np.random.randint(1,100,(12,12)) #last value defines the size of matrix
```

[32]:
```
array([[57, 54, 88, 25, 65, 17, 53,  6,  4, 94, 94, 74],
       [69, 59, 42, 47, 37, 83, 21,  5, 71, 86, 80, 16],
       [20, 64, 56, 82, 30, 99, 75, 98, 97, 48, 61, 62],
       [57, 78, 96, 73, 71, 73, 47, 22,  9, 62, 48,  2],
       [84, 39,  2, 19, 28, 43, 28, 40, 83, 20, 56, 67],
       [33, 56, 24, 62,  9, 38,  6, 73, 37, 37, 49, 49],
       [35, 28, 52,  7, 81, 47, 27,  6, 30, 36, 15, 59],
       [16, 91, 72, 97, 83, 46, 14, 52, 84, 62, 24, 80],
       [50, 71, 26, 51, 27, 57, 18, 44, 74, 12, 44, 36],
       [31, 79, 80, 71, 33,  1, 84, 87, 77, 87, 32, 46],
       [15, 41, 76, 86, 45, 81, 17, 31, 14, 58, 79, 27],
       [14, 92, 93, 58, 22, 22, 52, 25, 11, 90, 84, 21]])
```

2.4 Reshape

[33]:
```
n = np.arange(1,21).reshape(5,4) # We can reshape the matrix or array but no of␣
↪elements should be same
n
```

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

[34]:
```
np.arange(1,21).reshape(10,2)
```

[34]:
```
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
```

```
       [11, 12],
       [13, 14],
       [15, 16],
       [17, 18],
       [19, 20]])
```

2.5 Slicing & Indexing

[35]:
```
np.random.randint(1,21,(4,5))
```

[35]:
```
array([[12,  4, 11,  7, 12],
       [17,  5, 14, 14, 20],
       [10, 14, 14,  2,  7],
       [14, 17, 15,  5,  5]])
```

[36]:
```
b = np.random.randint(10,20,(5,4))
```

[37]:
```
b
```

[37]:
```
array([[13, 14, 19, 19],
       [18, 10, 19, 19],
       [14, 16, 18, 18],
       [19, 17, 19, 17],
       [15, 16, 10, 17]])
```

[38]:
```
b[:] # this is slicing we use colons to slice a portion
```

[38]:
```
array([[13, 14, 19, 19],
       [18, 10, 19, 19],
       [14, 16, 18, 18],
       [19, 17, 19, 17],
       [15, 16, 10, 17]])
```

[39]:
```
b[0:]
```

[39]:
```
array([[13, 14, 19, 19],
       [18, 10, 19, 19],
       [14, 16, 18, 18],
       [19, 17, 19, 17],
       [15, 16, 10, 17]])
```

[40]:
```
l = [1,2,3,4,5,6]
```

[41]:
```
l[1:5]
```

[41]:
```
[2, 3, 4, 5]
```

[42]:
```
b[0:5]
```

[42]:
```
array([[13, 14, 19, 19],
       [18, 10, 19, 19],
       [14, 16, 18, 18],
       [19, 17, 19, 17],
       [15, 16, 10, 17]])
```

[43]:
```
b[0:1,0:1] # slicing row and column both
```

[43]:
```
array([[13]])
```

[44]:
```
# if i want print 13 then i have to pass index number as [row ,column] form
b[1,1]
```

[44]:
```
10
```

[45]:
```
b[4,0]
```

[45]:
```
15
```

[46]:
```
# if want acces the 11 by indexing which is in 0th row and 0th column
b[0,0]
```

[46]:
```
13
```

[47]:
```
# if want acces the 11 by slicing which is in 0th row and 0th column
b[:1,:1]
```

[47]:
```
array([[13]])
```

[48]:
```
b
```

[48]:
```
array([[13, 14, 19, 19],
       [18, 10, 19, 19],
       [14, 16, 18, 18],
       [19, 17, 19, 17],
       [15, 16, 10, 17]])
```

[49]:
```
b[0:-2]
```

[49]:
```
array([[13, 14, 19, 19],
       [18, 10, 19, 19],
       [14, 16, 18, 18]])
```

[50]:
```
b[-5:-3]
```

[50]:
```
array([[13, 14, 19, 19],
       [18, 10, 19, 19]])
```

2.6 Operations

[51]:
```
arr2 = np.random.randint(0,100,(10,10))
```

[52]:
```
arr2
```

[52]:
```
array([[ 1, 43, 76,  7, 15, 90, 17, 37, 65, 62],
       [93, 69, 18, 78, 98,  5, 22, 82, 76, 93],
       [34, 34, 98, 46,  9,  2,  4, 20, 23, 19],
       [72, 22, 68, 49, 86, 60, 58, 61, 25, 70],
       [24,  9, 33, 69, 32, 87, 74, 60, 20, 12],
       [94, 29, 71, 22, 98, 75, 68, 46, 33, 37],
       [ 9, 49, 37, 77, 72, 46, 61, 45, 95, 57],
       [53, 99, 95, 12, 60, 86, 90, 49, 84, 75],
       [26, 53, 63, 94, 95, 17,  5, 13, 49, 28],
       [ 2, 41, 36, 37, 95, 39, 82, 42,  7, 98]])
```

[53]:
```
arr2[:5]
```

[53]:
```
array([[ 1, 43, 76,  7, 15, 90, 17, 37, 65, 62],
       [93, 69, 18, 78, 98,  5, 22, 82, 76, 93],
       [34, 34, 98, 46,  9,  2,  4, 20, 23, 19],
       [72, 22, 68, 49, 86, 60, 58, 61, 25, 70],
       [24,  9, 33, 69, 32, 87, 74, 60, 20, 12]])
```

[54]:
```
arr2[-7:-5]
```

[54]:
```
array([[72, 22, 68, 49, 86, 60, 58, 61, 25, 70],
       [24,  9, 33, 69, 32, 87, 74, 60, 20, 12]])
```

[55]:
```
arr2[4,5]
```

[55]:
```
87
```

[56]:
```
arr2[::-2]
```

[56]:
```
array([[ 2, 41, 36, 37, 95, 39, 82, 42,  7, 98],
       [53, 99, 95, 12, 60, 86, 90, 49, 84, 75],
       [94, 29, 71, 22, 98, 75, 68, 46, 33, 37],
       [72, 22, 68, 49, 86, 60, 58, 61, 25, 70],
       [93, 69, 18, 78, 98,  5, 22, 82, 76, 93]])
```

2.7 Some more common functions

[57]:
```
arr.max()
```

[57]:
```
5
```

[58]:
```
arr.min()
```

[58]:
```
1
```

[59]:
```
arr.mean()
```

[59]:
```
3.0
```

[60]:
```
from numpy import *
```

[61]:
```
median(arr)
```

[61]:
```
3.0
```

[62]:
```
arr[4] = 6
arr = np.insert(arr,0,0)
```

[63]:
```
arr.reshape(2,3)
```

[63]:
```
array([[0, 1, 2],
       [3, 4, 6]])
```

[64]:
```
arr.reshape(2,3,order='C')
```

[64]:
```
array([[0, 1, 2],
       [3, 4, 6]])
```

[65]:
```
arr.reshape(2,3,order='A')
```

[65]:
```
array([[0, 1, 2],
       [3, 4, 6]])
```

[66]:
```
arr.reshape(2,3,order='F')
```

[66]:
```
array([[0, 2, 4],
       [1, 3, 6]])
```

[67]:
```
mat = np.arange(0,100).reshape(10,10)
```

[68]:
```
mat
```

[68]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
```

```
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

[69]:
```
mat[:,6]
```

[69]:
```
array([ 6, 16, 26, 36, 46, 56, 66, 76, 86, 96])
```

[70]:
```
arr2
```

[70]:
```
array([[ 1, 43, 76,  7, 15, 90, 17, 37, 65, 62],
       [93, 69, 18, 78, 98,  5, 22, 82, 76, 93],
       [34, 34, 98, 46,  9,  2,  4, 20, 23, 19],
       [72, 22, 68, 49, 86, 60, 58, 61, 25, 70],
       [24,  9, 33, 69, 32, 87, 74, 60, 20, 12],
       [94, 29, 71, 22, 98, 75, 68, 46, 33, 37],
       [ 9, 49, 37, 77, 72, 46, 61, 45, 95, 57],
       [53, 99, 95, 12, 60, 86, 90, 49, 84, 75],
       [26, 53, 63, 94, 95, 17,  5, 13, 49, 28],
       [ 2, 41, 36, 37, 95, 39, 82, 42,  7, 98]])
```

[71]:
```
arr2[:,6]
```

[71]:
```
array([17, 22,  4, 58, 74, 68, 61, 90,  5, 82])
```

[72]:
```
mat[1,:]
```

[72]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

[73]:
```
mat[:,-1]
```

[73]:
```
array([ 9, 19, 29, 39, 49, 59, 69, 79, 89, 99])
```

[74]:
```
mat
```

[74]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

[75]:
```
mat[:-1,-2]
```

[75]:
```
array([ 8, 18, 28, 38, 48, 58, 68, 78, 88])
```

[76]:
```
mat[0:10:3]
```

[76]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

2.8 Masking

[77]:
```
mat[mat<=50]
```

[77]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50])
```

[78]:
```
mat>50
```

[78]:
```
array([[False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True]])
```

[79]:
```
mat[mat==50]
```

[79]:
```
array([50])
```

[80]:
```
mat == 50
```

[80]:
```
array([[False, False, False, False, False, False, False, False, False,
        False],
```

```
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [ True, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False]])
```

2.9 Array Creation Function

[85]:
```
# create an array from a list
a = np.array([1,2,3,4,5,6,7,8,9])
print(a)
```

```
[1 2 3 4 5 6 7 8 9]
```

[98]:
```
# create an array with 2 steps between each array
b = np.arange(0,10,2)
b
```

[98]:
```
array([0, 2, 4, 6, 8])
```

[102]:
```
#create an array with linearly spaced value
c = np.linspace(1,10,7)
c
```

[102]:
```
array([ 1. ,  2.5,  4. ,  5.5,  7. ,  8.5, 10. ])
```

[105]:
```
#Create an array filled with zeros
a = np.zeros((2,2))
a
```

[105]:
```
array([[0., 0.],
       [0., 0.]])
```

[106]:
```
a = np.zeros((2,2),dtype = int) #Hyperparameter tunining with int
a
```

[106]:
```
array([[0, 0],
       [0, 0]])
```

[108]:
```
# Create an array filled with ones
a = np.ones((2,2))
a
```

[108]:
```
array([[1., 1.],
       [1., 1.]])
```

[113]:
```
# Create an dentity matrix using eye function
a = np.eye(2)
a
```

[113]:
```
array([[1., 0.],
       [0., 1.]])
```

2.10 Array Manipulation Functions

[116]:
```
a1 = np.array([1,2,3,4,5,6])
reshaped = np.reshape(a1,(2,3))
print(reshaped)
```

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

[120]:
```
#Flatten an array 2d to 1d
f1 = np.array([[1,2,3],[4,5,6]])
flattned = np.ravel(f1)
print(flattned)
```

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

[132]:
```
#Transpose an array
e1 = np.array([[1,2],[2,3]])
transposed = np.transpose(e1)
print(e1,' \n ')
print(transposed)
```

```
[[1 2]
 [2 3]]

[[1 2]
 [2 3]]
```

[134]:
```
# stack array vertically
a = np.array([1,2,3])
b = np.array([4,5,6])
stacked = np.vstack([a,b]) # a then b vertically
```

```
print(stacked)
```

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

[135]:
```
# stack array horizentally
a = np.array([1,2,3])
b = np.array([4,5,6])
stacked = np.hstack([a,b]) # a then b horizentally
print(stacked)
```

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

2.11 Mathematical Functions

[137]:
```
#Add two arrays
g = np.array([1,2,3,4])
added = np.add(g,2)  #this will add 2 in each element of array
print(added)
```

```
[3 4 5 6]
```

[138]:
```
#Square each element
squared = np.power(g,2)
print(squared)
```

```
[ 1  4  9 16]
```

[140]:
```
# square root of each element
sqrt = np.sqrt(squared)
print(sqrt)
```

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

[148]:
```
a = np.insert(a,3,4)
a
```

[148]:
```
array([1, 2, 3, 4])
```

[149]:
```
g
```

[149]:
```
array([1, 2, 3, 4])
```

[150]:
```
dprod = np.dot(a,g)
print(dprod)
```

```
30
```

2.12 Statistical Functions

[154]:
```
# Mean of array s
s = np.array([1,2,3,4,5])
mean = np.mean(s)
mean
```

[154]: 3.0

[156]:
```
# Standard Deviation
std = np.std(s)
print(std)
```

```
1.4142135623730951
```

[157]:
```
# Minimum element of an array
mini = np.min(s)
print(mini)
```

```
1
```

[158]:
```
# Maximum element of an array
maxx = np.max(s)
print(maxx)
```

```
5
```

2.13 linear Algebra Functions

[162]:
```
# create a matrix
matrix = np.arange(1,5).reshape(2,2)
matrix
```

[162]:
```
array([[1, 2],
       [3, 4]])
```

[163]:
```
# Determinant of Matix
dm = np.linalg.det(matrix)
```

[164]:
```
dm
```

[164]: -2.0000000000000004

[165]:
```
# Inverse of matrix
inv = np.linalg.inv(matrix)
inv
```

[165]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

2.14 Random Sampling Functions

[170]:
```
# Generate random value between 0 and 1
random = np.random.rand(3)
random
```

[170]:
```
array([0.51987321, 0.7329671 , 0.58476912])
```

[172]:
```
# Set seed for reproductibility
np.random.seed(0) # this will help to reproduce same random number
# Generate random values between 0 and 1
random_vals = np.random.rand(3)  # Array of 3 random values between 0 and 1
print("Random values:", random_vals)
```

```
Random values: [0.5488135  0.71518937 0.60276338]
```

[175]:
```
# Generate random integers
randum = np.random.randint(0,10,size=10)
randum
```

[175]:
```
array([8, 1, 6, 7, 7, 8, 1, 5, 9, 8])
```

[184]:
```
rng = np.random.default_rng()  # Create a default random number generator
random_integers = rng.integers(0, 9, size=4)
print(random_integers)
```

```
[0 1 4 7]
```

2.15 Boolean & Logical Functions

[186]:
```
# Check if all elements are True
# all function checks for True value in variable pr array
logical = np.array([True,False,True])
all = np.all(logical)
print(all)
```

```
False
```

[187]:
```
# Check if all elements are True
# all function checks for True value in variable pr array
logical = np.array([True,True,True])
all = np.all(logical)
print(all)
```

```
True
```

2.16 Set Operations

[191]:
```
# Intersection of two arrays
set_a = np.array([1, 2, 3, 4])
set_b = np.array([3, 4, 5, 6])
intersection = np.intersect1d(set_a,set_b)
print('Intersection of a , b - ',intersection)
```

```
Intersection of a , b -  [3 4]
```

[190]:
```
# Union of two arrays
union = np.union1d(set_a,set_b)
print(union)
```

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

2.17 Array Attribute Function

[193]:
```
# Array attributes
a = np.array([1, 2, 3])
shape = a.shape  # Shape of the array
size = a.size    # Number of elements
dimensions = a.ndim  # Number of dimensions
dtype = a.dtype   # Data type of the array

print("Shape of a:", shape)
print("Size of a:", size)
print("Number of dimensions of a:", dimensions)
print("Data type of a:", dtype)
```

```
Shape of a: (3,)
Size of a: 3
Number of dimensions of a: 1
Data type of a: int32
```

2.18 Other Functions

[195]:
```
a = np.array([1, 2, 3])
copied = np.copy(a) # creates a copy of an array
print(copied)
```

```
[1 2 3]
```

[ ]:
```

```