numpy2

September 15, 2025

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

[4]:
```
a = np.arange(1, 17)
```

[5]:
```
a
```

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

[6]:
```
a.ndim
```

[6]:
```
1
```

[8]:
```
a.shape
```

[8]:
```
(16,)
```

[9]:
```
a = a.reshape(4, 4)
```

[10]:
```
a
```

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

[11]:
```
a.shape
```

[11]:
```
(4, 4)
```

[12]:
```
a.ndim
```

[12]:
```
2
```

[14]:
```
a.reshape(2, -1)
```

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

```
[16]: # a.reshape(3, -1)
```

```
[17]: # Length of this array?
```

```
[19]: a
```

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

```
[18]: len(a)
```

```
[18]: 4
```

```
[20]: len(a[0])
```

```
[20]: 4
```

```
[21]: a.size
```

```
[21]: 16
```

```
[26]: a1 = np.arange(12).reshape(3, 4)
```

```
[28]: a1.shape
```

```
[28]: (3, 4)
```

```
[29]: a1.T
```

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

```
[30]: a1
```

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

```
[31]: a1.transpose()
```

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

```
[32]: a1
```

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

```
[ ]:
```

0.1 Indexing and Slicing in 2D arrays

```
[40]: a = np.arange(0, 12).reshape(3, 4)
```

```
[41]: a
```

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

```
[52]: a[2][3]
```

```
[52]: 11
```

```
[53]: a[-1][-1]
```

```
[53]: 11
```

```
[48]: a[0]
```

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

```
[51]: a[0][3]
```

```
[51]: 3
```

```
[43]: a[1]
```

```
[43]: array([4, 5, 6, 7])
```

```
[44]: a[2]
```

```
[44]: array([ 8,  9, 10, 11])
```

```
[54]: a
```

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

```
[58]: a[1][3]
```

```
[58]: 7
```

```
[72]: a[1, 3] # alternate to a[1][3]
```

```
[72]: 7
```

```
[63]: # Accessing multiple indexes at a time
```

```
[64]: arr = np.arange(10)
```

```
[65]: arr
```

```
[65]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
[71]: # arr[1, 2, 4]
```

```
[73]: arr[[1, 3, 4, 5, 1, 2]]
```

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

```
[74]: a
```

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

```
[75]: a[1, 2]
```

```
[75]: 6
```

```
[80]: # Following code will return : (1, 1), (1, 2), (2, 3)
      a[[1, 1, 2], [1, 2, 3]]
```

```
[80]: array([ 5,  6, 11])
```

```
[81]: # a[[1, 1, 2, 4], [1, 2, 3]]
```

```
[ ]:
```

```
[82]: # slicing
```

[83]:
```
a
```

[83]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[87]:
```
arr
```

[87]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[88]:
```
arr[:]
```

[88]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[89]:
```
arr[:3]
```

[89]:
```
array([0, 1, 2])
```

[90]:
```
a
```

[90]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[91]:
```
a[:]
```

[91]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[92]:
```
a[:2]
```

[92]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

[98]:
```
# Get last two rows?

a[1:3]
```

[98]:
```
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[95]:
```
a[:-2]
```

[95]:
```
array([[0, 1, 2, 3]])
```

[99]:
```
a
```

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

```
[100]: a1 = np.arange(0, 16).reshape(4, 4)
```

```
[101]: a1
```

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

```
[103]: # Getting only columns
```

```
[102]: a1[ : , :2 ]
```

```
[102]: array([[ 0,  1],
              [ 4,  5],
              [ 8,  9],
              [12, 13]])
```

```
[104]: a1
```

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

```
[105]: a1[ 1: , 1:3 ]
```

```
[105]: array([[ 5,  6],
              [ 9, 10],
              [13, 14]])
```

```
[106]: # Using jump in my slicing
```

```
[107]: a1[ : , 1::2]
```

```
[107]: array([[ 1,  3],
              [ 5,  7],
              [ 9, 11],
              [13, 15]])
```

```
[114]: a1[ : , [1, 3]]
```

```
[114]: array([[ 1,  3],
              [ 5,  7],
              [ 9, 11],
              [13, 15]])
```

```
[112]: # a1[[0, 1, 2, 3], [0, 2, 3, 2]]
```

```
[ ]:
```

0.1.1 Masking of values in an array

```
[115]: a
```

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

```
[116]: a < 6
```

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

```
[117]: a[a < 6]
```

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

```
[ ]:
```

0.1.2 Aggregate Functions

```
[118]: a
```

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

```
[122]: # sum(a)
```

```
[123]: a.sum()
```

```
[123]: 66
```

```
[124]: np.sum(a)
```

```
[124]: 66
```

```
[126]: a
```

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

```
[125]: a.max()
```

```
[125]: 11
```

```
[127]: a.min()
```

```
[127]: 0
```

```
[129]: a.mean()
```

```
[129]: 5.5
```

```
[130]: a
```

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

```
[131]: a.sum(axis = 0)
```

```
[131]: array([12, 15, 18, 21])
```

```
[132]: a.sum(axis = 1)
```

```
[132]: array([ 6, 22, 38])
```

```
[133]: a.min(axis = 0)
```

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

```
[134]: a.min(axis = 1)
```

```
[134]: array([0, 4, 8])
```

```
[136]: np.min(a, axis = 1)
```

```
[136]: array([0, 4, 8])
```

```
[ ]:
```

```
[138]: # import this
```

[ ]:

```
```

0.2 Logical Operators

- any
- all

[143]:

```
# any: will give true even if one of the value satisfies the condition
```

[144]:

```
# Item prices on myntra shopping list
prices = np.array([50, 45, 25, 20, 35])

# budget
budget = 30

# Check if there's at least one item that you can afford
can_afford = np.any(prices <= budget)

if can_afford:
    print("Hurrah! I can buy atleast one item :)")
else:
    print("Sorry, better luck next time.")
```

```
Hurrah! I can buy atleast one item :)
```

[145]:

```
prices <= budget
```

[145]:

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

[146]:

```
# all: it gives true only if all values gets satisfied with the condition
```

[148]:

```
# Item prices on myntra shopping list
prices = np.array([50, 45, 25, 20, 35])

# budget
budget = 30

# Check if there's at least one item that you can afford
can_afford = np.all(prices <= budget)

if can_afford:
    print("Hurrah! I can any product that I want :)")
else:
    print("Sorry, better luck next time.")
```

```
Sorry, better luck next time.
```

[ ]:

```
```

[150]:

```python
# np.where
```

[151]:

```python
# product prices:
prices = np.array([45, 55, 60, 75, 40, 90])

# Apply a 10% discount to prices above $50
discounted_prices = np.where(prices > 50, prices * .9, prices)

print("Original prices:", prices)
print("Discounted prices:", discounted_prices)
```

```
Original prices: [45 55 60 75 40 90]
Discounted prices: [45.  49.5 54.  67.5 40.  81. ]
```

[ ]:

```python

```

0.2.1 Sorting

[152]:

```python
ar = np.array([4, 7, 3, 6, 8, 1, 0, 2, 5, 9])
```

[153]:

```python
ar
```

[153]:

```
array([4, 7, 3, 6, 8, 1, 0, 2, 5, 9])
```

[160]:

```python
# np.sort() returns a copy of original array having all elements sorted in ascending order
```

[161]:

```python
b = np.sort(ar)
```

[162]:

```python
b
```

[162]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[163]:

```python
ar
```

[163]:

```
array([4, 7, 3, 6, 8, 1, 0, 2, 5, 9])
```

[167]:

```python
# following will sort the array in place: arr.sort()
```

[168]:

```python
ar.sort()
```

[169]:

```python
ar
```

[169]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[172]:

```python
# Jugaad
```

```
        ar.sort()
```

```
[174]: ar[::-1]
```

```
[174]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

```
[ ]:
```

```
[176]: # sorting a 2D array
```

```
[187]: a = np.array([[1, 5, 3], [2, 0, 7], [40, 20, 30]])
```

```
[188]: a
```

```
[188]: array([[ 1,  5,  3],
              [ 2,  0,  7],
              [40, 20, 30]])
```

```
[189]: # by default axis is 1 in sort method
```

```
[190]: np.sort(a)
```

```
[190]: array([[ 1,  3,  5],
              [ 0,  2,  7],
              [20, 30, 40]])
```

```
[191]: np.sort(a, axis = 1)
```

```
[191]: array([[ 1,  3,  5],
              [ 0,  2,  7],
              [20, 30, 40]])
```

```
[193]: a
```

```
[193]: array([[ 1,  5,  3],
              [ 2,  0,  7],
              [40, 20, 30]])
```

```
[192]: np.sort(a, axis = 0)
```

```
[192]: array([[ 1,  0,  3],
              [ 2,  5,  7],
              [40, 20, 30]])
```

```
[ ]:
```

0.2.2 Vectorization

- Vectorization in NumPy refers to performing operations on entire arrays or array elements simultaneously, which is significantly faster and more efficient than using explicit loops.

```
[194]: a = np.arange(10)
```

```
[195]: a
```

```
[195]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
[196]: def random_operation(x):
            if x % 2 == 0:
                x += 2
            else:
                x -= 2

            return x
```

```
[198]: # random_operation(a)
```

```
[210]: # random_operation(a)
```

```
[200]: random_operation(1)
```

```
[200]: -1
```

```
[201]: operation = np.vectorize(random_operation)
```

```
[202]: type(operation)
```

```
[202]: numpy.vectorize
```

```
[203]: operation(a)
```

```
[203]: array([ 2, -1,  4,  1,  6,  3,  8,  5, 10,  7])
```

```
[204]: def square(x):
            return x ** 2
```

```
[211]: square(a)
```

```
[211]: array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

```
[205]: a
```

```
[205]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[206]:

```
sq = np.vectorize(square)
```

[207]:

```
sq(a)
```

[207]:

```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

[208]:

```
for i in a:
    print(square(i))
```

```
0
1
4
9
16
25
36
49
64
81
```

[212]:

```
a1
```

[212]:

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

[213]:

```
sq(a1)
```

[213]:

```
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121],
       [144, 169, 196, 225]])
```

[215]:

```
a1
```

[215]:

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

[214]:

```
operation(a1)
```

[214]:

```
array([[ 2, -1,  4,  1],
       [ 6,  3,  8,  5],
       [10,  7, 12,  9],
       [14, 11, 16, 13]])
```

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```