**Numpy**

Data types:

i – integer

b – boolean

u – unsingned integer

f – float

c – complex float

m – timedelta

M – datetime

O – object

S – string

U – unicode string

V – fixed chunl of memory for other type ( void )

* Dtype – data ni type ni korish

import numpy as np  
arr = np.array([1, 2, 3, 4])  
print(arr.dtype)

* Arr.Astype() – bu array ni type ni o’zgartirish uchun ishlatilinadi.

**Copy and view**

Copy() va View() farqi , copy bu array ni copy qiladi va yangi array yasaydi.

Yangi arraayga yani copy qilingan arrayga Qanaqadir o’zgartish kiritilsa asosiy arrayga ta’sir qilmaydi.

x = arr.copy()

arr[1] = 6

print(x)

print(arr)

result: [0 1 4 5]

[0 6 4 5]

**View**() esa bu faqatgina korsatadi, yangi variable ga assign qilgandan kyn edit qilsak nimanidir togridan togri tasir qiladi asosiy array ga, agar view bilan assign qilingan variable ni edit qilsak ham ozgaradi.

y = arr.view()

arr[2] = 7

y[3] = 8

print(y)

print(arr)

[0 6 7 8]

[0 6 7 8]

Endi osha view mi yoki yoqligini tekshirsa ham boladi. Yani Base attribute dan foydalanib, agar arrayda data bo’lsa unda u None qaytaradi, agar data bo’lmasa ya’ni view bo’lsa arrayni ozini qaytaradi. Chunki view faqat korsatib beradi, hech qanaqa data store qilmaydi o’zida.

print(y.base)

print(arr.base)

print(x.base)

[0 6 7 8]

None

None

* Arr.shape bilan array ni shapini olsak boladi Masalan:

new = np.array([arr,x])

new.shape

array([[0, 6, 7, 8],

[0, 1, 4, 5]]) result>>>> (2, 4)

* **Reshape –** Biror array ni shape ni ozgartirish hisoblanadi , Masalan ;

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

new\_cat = cat.reshape(4,3)

new\_cat

**Result:**

array([[ 1, 2, 3],

[ 4, 5, 6],

[ 7, 8, 9],

[10, 11, 12]])

Reshape ham view dek ishlaydi; Base da value ko’rsatadi.

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

print(arr.reshape(2, 4).base)

Result:

[1 2 3 4 5 6 7 8]

**Reshape** bilan -1 ni ishlatish; -1 avtomatik mos keluvchi dimension ni yoki nechta dan bo’lish kerakligini tanab beradi column ga , shuningdek -1 bitta dimension pasga tushurish uchun ham ishlatilinadi.

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

b = a.reshape(-1) # NumPy avtomatik (6,) o‘lchamga o‘zgartiradi

print(b)

*[1 2 3 4 5 6]*

* Flatten() dan foydalanish shape ni 1D ga otkazib beradi.
* nditer() – Bu array ni ichidagi elementlar ni chiqarish uchun for loop ni ishlatib o’tirgandan ancha yaxshi u nechta dimension bo’lsa ham chiqarib beraveradi .

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):

print(x)

Result:

1

2

3

4

5

6

7

8

* Nditer() bilan slice qilib olingan array ni ham chiqarsa boladi;

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

for x in np.nditer(arr[:, ::2]):

print(x)

1

3

5

7

* ndnumerate() - bu raqamlash uchun ishlatilinadi.

arr = np.array([1, 2, 3])

for idx, x in np.ndenumerate(arr):

print(idx, x)

(0,) 1

(1,) 2

(2,) 3

New\_cat = array([[ 1, 2, 3],

[ 4, 5, 6],

[ 7, 8, 9],

[10, 11, 12]])

for a, b in np.ndenumerate(new\_cat):

    print(a,b)

result:

(0, 0) 1

(0, 1) 2

(0, 2) 3

(1, 0) 4

(1, 1) 5

(1, 2) 6

(2, 0) 7

(2, 1) 8

(2, 2) 9

(3, 0) 10

(3, 1) 11

(3, 2) 12

* np.concatenate([ ]) array larni qoshish uchun ishlatilinadi.
* Np.stack()

Concataneta bilan bir xil lekin bunda axis = 1 bilan boshqacha qoshsa bo’ladi.

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.stack((arr1, arr2), axis=1)

print(arr)

[[1 4]

[2 5]

[3 6]]

* np.hstack ro boylab qoshish uchun kerak bo’ladi. Masalan:

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.hstack((arr1, arr2))

print(arr)

[1 2 3 4 5 6]

* np.vstack() esa column lar boyicha qoshib beradi;

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.vstack((arr1, arr2))

print(arr)

[[1 2 3]

[4 5 6]]

* dstack() esa balandlik boyicha qoshib beradi yani teppa va paskini.

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.dstack((arr1, arr2))

print(arr)

[[[1 4]

[2 5]

[3 6]]]

* Splitting:

np.array\_split() - bu function array lar ni bolib beradi:

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array\_split(arr, 3)

print(newarr)

[array([1, 2]), array([3, 4]), array([5, 6])]

* Agar teng bolinmasa oxiridan boshlab boshqattan boladi: Masalan:

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array\_split(arr, 4)

print(newarr)

[array([1, 2]), array([3, 4]), array([5]), array([6])]

* An alternate solution is using hsplit() opposite of hstack()
* Where() bilan array ni ichidagi index n topsak boladi. Masalan:

arr = np.array([1, 2, 3, 4, 5, 4, 4])

x = np.where(arr == 4)

print(x)

* Shu yerda qaysi index dagi elementlar 4 ga tengligi yozilyapti; va result : (array([3, 5, 6]),) chunki 3 ta index da bor

**Sorting Arrays**

Sort() method array ni value larini sort qilish uchun ishlatilinadi;

**Caution:** Faqat yodda tutish kerak u copy create qiladi asosiy array ga ta’sir qilmaydi.

sorted\_arr = np.sort(arr3)

ishlatilinishi!

filtered\_arr = sorted\_arr > 8

filtered\_arr

Result: >> array([False, False, False, False, False, False, True, True, True])

Demak, filter qilishni shunday arraydan katta deb qo’ysak yetarli shunda boshqa arrayda qaysilari conditionga to’g’ri keladi yoki yo’q aniqlab True, False qo’yib beradi.

sorted\_arr[filtered\_arr]

Result: >> array([ 9, 12, 35])  
Bu code bilan esa o’sha True result berganlarini olib chiqsa bo’ladi..

arr = np.array([1, 2, 3, 4, 5, 6, 7])  
filter\_arr = arr % 2 == 0  
newarr = arr[filter\_arr]  
# print(filter\_arr)  
print(newarr) Result: >> [2,4,6]