Numpy

25/10/25

* import numpy as np
```
a = np.array ([1,2,3])
print (a)
print (type(a))
```

There are multi dimensional array:

- 0D array.
  Scalars, every element of an array is a 0D array.
```
a= np.array (4)
```
- 1D array.
```
a = np.array ((1,2,3))
```
- 2D array.
```
a =np.array ((1,2,3), (4,5,6))
```
- 3D array.
```
a = np.array([[[1,2],[3,4]],
              [[5,6],[7,8]]])
```

* To check a dimension of an array.

```
print (a.ndim)
```

- Higher dimensional array.

```
a = nparray ([1,2,3], ndmin = 10)
print (a.ndim)
```

↳ output : 10.

* Array indexing

```
print (a[1])
```

For 2D array.

```
print (a[i,j])
```

For 3D array.

```
print (a[i,j,k])
```

- Array slicing.

```
print (a[1:3])
print (a[:4])
print (a[4:])
```

Step slicing.

```
print (a[2::2])
```

for 2D:

```
print (a[1, 1:4])
```

for 3D:

```
print (a[1,2, 1:4])
```

→ a [index of list, index of element inside]

* Data types in numpy.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - time delta
- M - date time
- O - object
- S - string
- U - unicode string
- V - void

To check datatype

```
print (a.dtype)
```

- Creating array with defined datatype

```
a = np.array ([1,2,3], dtype = 'S')
```

For i, u, f, S and U we can define size aswell.

```
a = np.array ([1,2,3,4], dtype = 'i4')
```

- Converting data Type on existing arrays.

```
a = np.array ([1.1, 2.2])
b = a.astype ('i')
```

(or)

```
b = a.astype (int)
```

- To copy.

```
a = np.array (1, 2, 3)
b = a.copy ()
a[0] = 0.
print (b)
```

=> (1, 2, 3)

- To view

```
a = np.array (1, 2, 3)
b = a.view ().
a[0] = 0.
print (b)
```

=> (0, 2, 3)

elements get updated in view at both a and b.

- To check if array owns its data

```
print (x.base)
print (y.base)
```

Copy - owns data

View - doesn't own.

- Array shape.
  gives the order of an array.

```
a = np.array([[1,2,3,4],[5,6,7,8]])
print(a.shape)
```
↳ (2, 4)

- Array reshape.

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

2D → `b = a.reshape(4,3)`

3D → `c = a.reshape(2,3,2)`

- Array iterating.
  Use for loops for iterating

For 3D.

```
for x in a:
    for y in x:
        for z in y:
            print(z)
```

⇒ Iterating using nditer()

Instead for multiple for loops for multi dimensional array we use np.nditer(). to access each element.

```
for i in np.nditer(a):
    print(i)
```

⇒ Enumerated iteration using ndenumerate
used to print index as well as element.

```
for idx, x in np.ndenumerate(a):
    print(idx, x)
```

Output:

(0), 1
(1), 2
(2), 3

⇒ Joining arrays.

c = np.concatenate((a,b))

2D ;

arr = np.concatenate((a,b), axis=1)

Stacking

arr = np.stack((a,b), axis=1)

↳ stacking along column

arr = np.hstack((a,b)) ⇒

stacking along row.

arr = np.vstack((a,b))

↳ along column

dstack

↳ along depth

⇒ Spliting:

na = np.array_split(a, 3)

↳ no of array to be splited

na[0], na[1], na[2]

For 2D ;

np.array_split(a, 3)

or to split along columns.

np.array_split(a, 3, axis=1)

⇒ Searching.

np.where(arr == 4)

np.where(arr % 2 == 0)

⇒ search sorted.

returns where the given value must be inserted to maintain the search order

* a = nparray ([6, 7, 8, 9])
  x = np. searchsorted (a, 7)
  print (x) => 1.

* a = np.array ([1, 3, 5, 7])
  x = np. searchsorted (a, [2, 4, 6])
  print (x)
  ⇊ [1 2 3]

=> Sorting:
  a = np.array([3, 2, 0, 1])
  print (np.sort (a))
  ⇊ [0, 1, 2, 3]

=> Filter.
  It filters and take out correspond element of another where value is True.

a = np.array([41, 42, 43, 44]).
bec
b = a[[True, False, True, False]]
print (b)
  ⇊ [41, 43]

alt:
  a = np. array ([1, 2, 3, 4])
  b = a > 2
  x = a [b].
  print (x) => [3, 4]