Numpy

August 19, 2022

1 Numpy

[1]:
```
import numpy as np
l=[1,2,5,7]
a=np.array(l)
print(a)
a=a+3
print(a)
a=a*3
print(a)
```

```
[1 2 5 7]
[ 4  5  8 10]
[12 15 24 30]
```

[2]:
```
import time
st=time.time()
l1=list(range(0,500000))
l2=list(range(0,500000))
sum=[]
for i in range(len(l1)):
    sum.append(l1[i]+l2[i])
print('list time',(time.time()-st)*1000,'ms')
st1=time.time()
a1=np.array(l1)
a2=np.array(l2)
a=a1+a2
print('list time',(time.time()-st1)*1000,'ms')
```

```
list time 161.7412567138672 ms
list time 73.65989685058594 ms
```

[3]:
```
ar1=np.array([5,6,3,9])
print(ar1)
print(ar1.ndim) # gives dimension of data
print(ar1.size) # gives no of elements in a array
print(ar1.shape) #gives shape of data (no of rows,no ofcolumns)
print(ar1.dtype) # gives dtattype

ar2=np.array([4,5,6,7.5])
print(ar2)
print(ar2.dtype)

ar3=np.array([4,5,6.5,'7'])
print(ar3)
print(ar3.dtype)
ar4=ar3.astype('float') # for typecasting
print(ar4)
```

```
[5 6 3 9]
1
4
(4,)
int64
[4.  5.  6.  7.5]
float64
['4' '5' '6.5' '7']
<U32
[4.  5.  6.5 7. ]
```

[4]:
```
ar=np.arange(0,10) #generate elements in seq order
print(ar)
ar1=np.linspace(1,10,20) #create equally spaced nos within specified range
print(ar1)
ar2=np.random.random(10) # always returns float between 0 to 1 to generate␣
 ↪random values
print(ar2)
ar3=np.random.seed(4) #added to generate fixed random numbers 4 is some logic␣
 ↪used for random no generation
ar3=np.random.random(10)
print(ar3)
ar4=np.random.randint(10) # generates only one number in the range of 10 arg␣
 ↪used is the range in which you want
print(ar4)
ar5=np.random.randint(1,10,5) #generates 5 random nos in between the range of␣
 ↪1-10 last arg in no of element you want
print(ar5)
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 1.          1.47368421  1.94736842  2.42105263  2.89473684  3.36842105
  3.84210526  4.31578947  4.78947368  5.26315789  5.73684211  6.21052632
  6.68421053  7.15789474  7.63157895  8.10526316  8.57894737  9.05263158
  9.52631579 10.        ]
[0.66200179 0.54757919 0.07893593 0.38757326 0.43125693 0.48959498
 0.33667835 0.50186315 0.47100272 0.69618985]
[0.96702984 0.54723225 0.97268436 0.71481599 0.69772882 0.2160895
 0.97627445 0.00623026 0.25298236 0.43479153]
2
[7 5 4 1 8]
```

[5]:
```
#2d array
ar=np.array([[5,6,8.5,9],[2,3,5,6]])
print(ar)
print(ar.ndim)
print(ar.shape)
ar2=ar.astype('int')
print(ar2)
ar3=np.random.rand(3,5) #generate random nos in 2d arg->no of rows,columns
print(ar3)
ar4=np.random.randint(1,10,(3,5))  # generate random integers 3rd arg-.> used␣
 ↪to declare no of rows and column
print(ar4)
```

```
[[5.  6.  8.5 9. ]
 [2.  3.  5.  6. ]]
2
(2, 4)
[[5 6 8 9]
 [2 3 5 6]]
[[0.39176619 0.81194775 0.61252534 0.26647378 0.64391427]
 [0.8110621  0.01902474 0.03978036 0.94500385 0.4463492 ]
 [0.44134853 0.06570954 0.17586123 0.86588276 0.84352812]]
[[3 3 2 1 2]
 [1 8 3 7 4]
 [8 9 3 7 8]]
```

[6]:
```
# indexing/slicing for 2D array
np.random.seed(4)
ar1=np.random.randint(1,10,(4,5))
print(ar1)
ar2=ar1[0:2,0:3]
print(ar2)
ar3=ar1[:,0:3]
print(ar3)
ar4=ar1[0:3,:]
print(ar4)
```

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

[13]:
```
#reshaping
np.random.seed(4)
ar1=np.random.randint(1,10,(3,4))
print(ar1)
ar2=ar1.reshape(2,6) #indexing is main -->cannot go out of bound -->it should␣
 ↪cover all elements should not be less than that for ex if total is 12 we␣
 ↪cannot give 3,3 it will throw error
print(ar2)
ar3=ar1.reshape(2,-1) # -1 is default value and it takes its value in such a␣
 ↪way to match the total no of elements
print(ar3)
ar4=ar1.reshape(-1,2)
print(ar4)
```

```
[[8 6 2 9]
 [8 9 3 8]
 [8 8 9 5]]
[[8 6 2 9 8 9]
 [3 8 8 8 9 5]]
[[8 6 2 9 8 9]
 [3 8 8 8 9 5]]
[[8 6]
 [2 9]
 [8 9]
 [3 8]
 [8 8]
 [9 5]]
```

[23]:
```
#Basic opns
ar=np.array([2,3,4,5])
ar1=np.array([5,6,7,9])
print(ar*ar1)
ar2=np.array([[2,3,4,5],[5,6,7,9],[10,20,30,40]])
print(ar2)
b=ar2.sum()
c=np.sum(ar2)
```

```
print(b)
print(c)
d=np.var(ar2) #--variance
e=np.mean(ar2)#--avg
f=np.std(ar2)#--std deviation
print(d)
print(e)
print(f)
g=ar2.sum(axis=1) #row wise addition
h=ar2.sum(axis=0) #column wise addition
print(g,h)
i=ar2.mean(axis=1) #row wise addition
j=ar2.mean(axis=0) #column wise addition
print(i,j)
k=np.sum(ar2[:,1:3])
print(k)
```

```
[10 18 28 45]
[[ 2  3  4  5]
 [ 5  6  7  9]
 [10 20 30 40]]
141
141
132.35416666666666
11.75
11.504528094044826
[ 14  27 100] [17 29 41 54]
[ 3.5   6.75 25.  ] [ 5.66666667  9.66666667 13.66666667 18.        ]
70
```

[29]:
```
#where condition
ar2=np.array([[2,3,4,5],[5,6,7,9],[10,20,30,40]])
print(ar2)
#where condition
ar2=np.where(ar2<10,0,ar2)
print(ar2)
```

```
[[ 2  3  4  5]
 [ 5  6  7  9]
 [10 20 30 40]]
[[ 0  0  0  0]
 [ 0  0  0  0]
 [10 20 30 40]]
```

[35]:
```
ar2=np.array([[2,3,4,5],[5,6,7,9],[10,20,40,40]])
print(ar2)
print(np.min(ar2))
```

```python
print(ar2.max())
print(np.max(ar2))
print(np.argmin(ar2))  #--position of max element
print(ar2.argmin())
print(np.argmax(ar2))  #--position of min element
```

```
[[ 2  3  4  5]
 [ 5  6  7  9]
 [10 20 40 40]]
2
40
40
0
0
10
```