```
def multiply_loops(A,B):
    C=np.zeros((A.shape[0],B.shape[1]))
    for i in range(A.shape[1]):
        for j in range(B.shape[0]):
            C[i,j]=A[i,j]*B[j,i]
    return C


def multiply_vector(A,B):
    return A@B #@ is an internal operator of numpy for multiplication(operator overloading)


import numpy as np
X=np.random.random((1000,1000))
Y=np.random.random((1000,1000))


%timeit multiply_loops(X,Y)
```

```
626 ms ± 146 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%timeit multiply_vector(X,Y)
```

```
86.3 ms ± 23.4 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
#Scalars
print(np.random.randint(5))
print(np.random.rand(5))
print(np.random.randn(5))
```

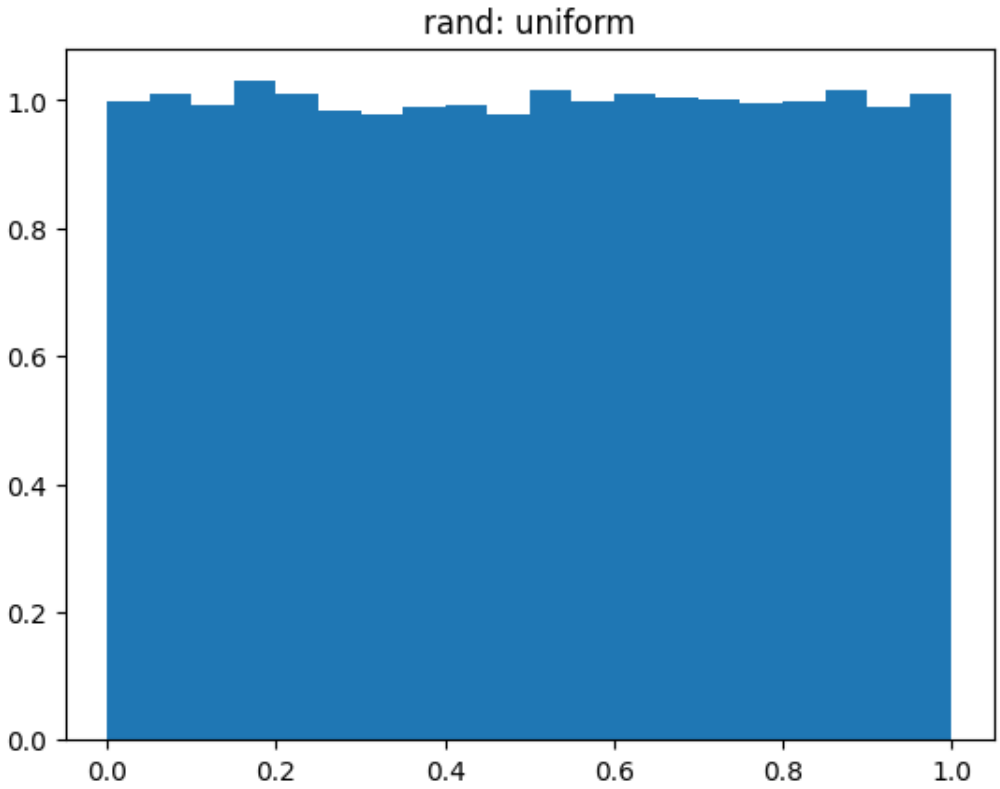
```
2
[0.46157098 0.10737273 0.28345193 0.12555727 0.77677797]
[-0.49193075 -0.82007425  2.02390781 -1.99136286 -1.24736096]
```

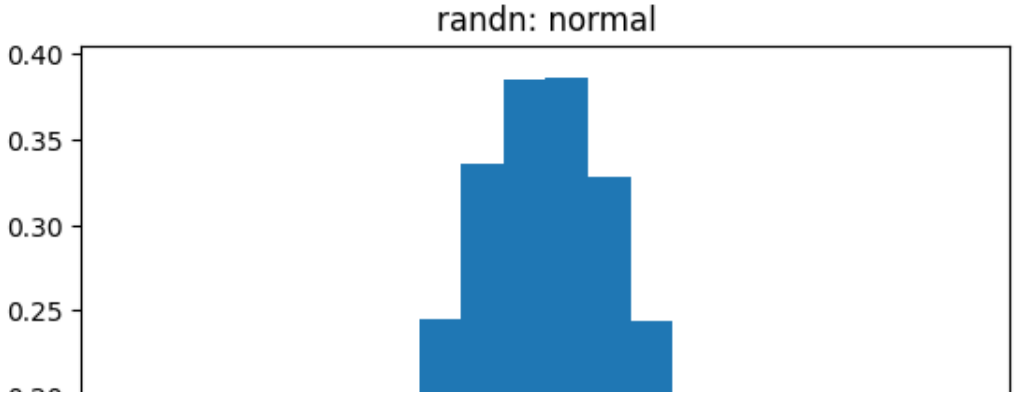
```
import numpy as np
import matplotlib.pyplot as plt

sample_size = 100000
uniform = np.random.rand(sample_size)
normal = np.random.randn(sample_size)

pdf, bins, patches = plt.hist(uniform, bins=20, range=(0, 1), density=True)
plt.title('rand: uniform')
plt.show()

pdf, bins, patches = plt.hist(normal, bins=20, range=(-4, 4), density=True)
plt.title('randn: normal')
plt.show()
```






```
import numpy as np
print(np.__version__)
```

```
1.22.4
```

```
#Vectors
import numpy as np
arr = np.array(42)
print(arr)
```

```
42
```

```
#Creating Numpy array by passing a Python list
import numpy as np
list_array1=np.array([1,2,3])
print(list_array1)

#or

sample_list = [1,2,3]
list_array = np.array(sample_list)
print(list_array)
```

```
#Converting Numpy array to list
list_array1.tolist()
```

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

```
#Creating Numpy array by passing a Python tuple
import numpy as np
tup = (1,2,3)
my_tup_array = np.array(tup)
print(my_tup_array )
```

```
[1 2 3]
```

```
my_array=np.array([1,2,3,4,5,6,7,8,9,10])
my_array=np.insert(my_array,3,14)
print("After insertion",my_array)
my_array=np.append(my_array,11)
print("After appending",my_array)
print("After deleting",np.delete(my_array,3))
my_array1=np.array([12,13,14])
print("After concatenating",np.concatenate((my_array,my_array1)))
print("After search",np.where(my_array==10))
```

```
After insertion [ 1  2  3 14  4  5  6  7  8  9 10]
After appending [ 1  2  3 14  4  5  6  7  8  9 10 11]
After deleting [ 1  2  3  4  5  6  7  8  9 10 11]
After concatenating [ 1  2  3 14  4  5  6  7  8  9 10 11 12 13 14]
After search (array([10]),)
```

```
# Creating a NumPy array by passing a range of values (using arange() function of NumPy)
my_range_array = np.arange(10,30,5)
print(my_range_array)
```

```
[10 15 20 25]
```

```
#Creating a NumPy array by passing an equally spaced range of values (using linspace() function of NumPy)
my_spaced_array = np.linspace(0,5/3,6)
print(my_spaced_array )

print(my_spaced_array.max())
print(my_spaced_array.min())
print(my_spaced_array.argmax())
print(my_spaced_array.argmin())
print(my_spaced_array.size)
```

```
[0.         0.33333333 0.66666667 1.         1.33333333 1.66666667]
1.6666666666666667
0.0
5
0
6
```

```
#Creating a NumPy array with all zeroes
my_array=np.zeros(10)
print(my_array)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
#Creating a NumPy array with all ones
my_array=np.ones(10)
print(my_array)
```

```
     [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

```
# Create an array with a given shape and a given value
np.full( (3,4), 0.11 )
```

```
     array([[0.11, 0.11, 0.11, 0.11],
            [0.11, 0.11, 0.11, 0.11],
            [0.11, 0.11, 0.11, 0.11]])
```

```
#NumPy Array Indexing
#Access Array Elements

#Get the first element from the following array:
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

```
     1
```

```
#Get third and fourth elements from the following array and add them.

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

```
     7
```

```
#Replacing selected elements of a NumPy array by a specific value
import numpy as np
my_orig_arr2=np.array([2, 9, 17, 13, 1, 4, 20, 57])
my_orig_arr2[2 : 5] = -6
print(my_orig_arr2)
```

```
     [ 2  9 -6 -6 -6  4 20 57]
```

```
#Reversing of an array elements
import numpy as np
my_orig_arr = np.array([1, 15, 3, 9, 26, 7, 89, 12])
my_rev_arr= my_orig_arr[ : : -1]
print(my_rev_arr)
```

```
     [12 89  7 26  9  3 15  1]
```

```
#Multidimensional array/Matrix¶
#Creating Numpy matrix by passing size using rand(randomly initialized with numbers between 0 and 1)
import numpy as np
my_rand_array = np.random.rand(3, 4)
print(my_rand_array)

#Creating an uninitialized Numpy matrix by passing size as a Python tuple
tup_dim = (3, 4)
my_uninitialized_array = np.empty(tup_dim)
print(my_uninitialized_array)
```

```
#Creating an initialized Numpy matrix by passing a list
my_initialized_array = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(my_initialized_array)

#Accessing matrix elements
print(my_uninitialized_array[2,1])
print(my_uninitialized_array[2][1])
```

```
[[0.25500895 0.02437215 0.7205901  0.1545722 ]
 [0.79557494 0.49053191 0.37296    0.98552497]
 [0.99012875 0.97673155 0.5411965  0.7767402 ]]
[[0.25500895 0.02437215 0.7205901  0.1545722 ]
 [0.79557494 0.49053191 0.37296    0.98552497]
 [0.99012875 0.97673155 0.5411965  0.7767402 ]]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
0.9767315519438332
0.9767315519438332
```

```
#Numpy - Arrays - Indexing and Array Slicing
#Extracting a slice(portion) of a NumPy array
apple = np.array([1, 8, 23, 3, 18, 91, 7, 15])
apple_slice=apple[1 : 4]
print(apple_slice)
print(apple)
apple_slice[1] = 99999
print(apple_slice)
print(apple)

#If you assign a single value to an ndarray slice, it is copied across the whole slice.
#ndarray slices are actually views on the same data buffer.
#If you modify it, it is going to modify the original ndarray as well.
#If you want a copy of the data, you need to use the copy method as another_slice
apple_slice_new = apple[2 : 5].copy()
apple_slice_new[1] = 222222
print(apple_slice_new )
print(apple)
```

```
[ 8 23  3]
[ 1  8 23  3 18 91  7 15]
[    8 99999     3]
[    1     8 99999     3    18    91     7    15]
[ 99999 222222     18]
[    1     8 99999     3    18    91     7    15]
```

```
#Printing array dimension,shape and data type.The NumPy's array class is called ndarray.
#The important attributes of a ndarray object are

import numpy as np
my_array=np.array([ [1, 4, 5, 6], [7, 8, 9, 10], [11, 12, 14, 16] ])
print(my_array.ndim)#no of axes
print(my_array.shape)
print(my_array.dtype)
print("space taken by each item",my_array.itemsize)
```

```
2
(3, 4)
int64
space taken by each item 8
```

```
import numpy as np
a = np.array(42)
```

```
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

NumPy - Arrays - Resizing an Array resize() function is used to create a new array of different sizes and dimensions. resize() can create an array of larger size than the original array. To convert the original array into a bigger array, resize() will add more elements (than available in the original array) by copying the existing elements (repeated as many times as required) to fill the desired larger size.

```
import numpy as np
my_arr = np.array([1,2,3,4,5,6,7,8])
my_arr.resize(3,4)
print(my_arr)
```

```
[[1 2 3 4]
 [5 6 7 8]
 [0 0 0 0]]
```

NumPy - Arrays - Reshaping an Array reshape()

reshape() function is used to create a new array of the same size (as the original array) but of different desired dimensions.

reshape() function will create an array with the same number of elements as the original array, i.e. of the same size as that of the original array. If you want to convert the original array to a bigger array, reshape() can't add more elements(than available in the original array) to give you a bigger array.

```
B = [1, 2, 3, 4, 5, 6, 7, 8, 9]
C=np.array(B)
C.reshape(3,3)
```

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

```
import numpy as np
my_first_arr=np.array([1,2,3,4,5,6,7,8])
my_new_arr= my_first_arr.reshape(2,4)
print(my_new_arr)
print(my_first_arr)

my_second_arr=np.array([0,1,2,3,4,5,6,7,8])
my_updated_arr= my_second_arr.reshape(-1,3)
print(my_updated_arr)
print(my_second_arr)
```

```
[[1 2 3 4]
 [5 6 7 8]]
[1 2 3 4 5 6 7 8]
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[0 1 2 3 4 5 6 7 8]
```

```
#Numpy - Mathematical Operations on NumPy Arrays - Addition and Subtraction
import numpy as np
a_arr = np.array([60, 70, 80, 90])
b_arr = np.arange(4)
c_arr=a_arr  + b_arr
print(c_arr)
d_arr = np.array([60, 70, 80, 90])
e_arr = np.arange(4)
f_arr= d_arr  - e_arr
print(f_arr)
```

```
[60 71 82 93]
[60 69 78 87]
```

```
#Numpy - Mathematical Operations on NumPy Arrays - Multiplication and Dot Product
import numpy as np
A_arr = np.array([ [ 5,9], [4, 7] ])
B_arr = np.array( [ [2, 8], [1, 6] ] )
M_arr=A_arr*B_arr
print(M_arr)
C_arr = np.array([ [ 5,9], [4, 7] ])
D_arr = np.array( [ [2, 8], [1, 6] ] )
P_arr= np.dot(C_arr,D_arr)
print(P_arr)
```

```
[[10 72]
 [ 4 42]]
[[19 94]
 [15 74]]
```

```
#Numpy - Mathematical Operations on NumPy Arrays - Division, Modulus
import numpy as np
R_div = np.array([ [ 25,65], [40, 14] ])
S_div = np.array([ [2, 10], [4, 5] ] )
T_div = R_div / S_div
print(T_div)

R_int_div = np.array([ [ 25,65], [40, 70] ])
S_int_div = np.array([ [2, 10], [8, 3] ] )
T_int_div = R_int_div // S_int_div
print(T_int_div)

R_mod = np.array([ [ 20,65], [40, 70] ])
S_mod = np.array([ [2, 10], [8, 3] ] )
T_mod = R_mod % S_mod
print(T_mod)
```

```
[[12.5  6.5]
 [10.   2.8]]
[[12  6]
 [ 5 23]]
[[0 5]
 [0 1]]
```

```
#Numpy - Mathematical Operations on NumPy Arrays - Exponents
import numpy as np
R_exp = np.array([ [ 4, 2], [3, 4 ] ] )
S_exp = np.array( [ [2, 10], [4, 5] ] )
T_exp=R_exp ** S_exp
print(T_exp)
```

```
[[  16 1024]
 [  81 1024]]
```

```
#Numpy - Mathematical Operations on NumPy Arrays - Conditional Operators
import numpy as np
U = np.array([ [ 22, 45], [90, 4 ] ] )
print(U<45)
```

```
[[ True False]
 [False  True]]
```

```
#Numpy - Mathematical and Statistical functions on NumPy Arrays
import numpy as np
X_stat=np.array([ [ 1, 2, 3], [ 4, -5, 6] ])
print("mean = ",X_stat.mean())
print("Variance = ",X_stat.var())
print("Standard Deviation = ",X_stat.std())
print("min = ",X_stat.min())
print("max = ",X_stat.max())
print("sum = ",X_stat.sum())
print("product = ",X_stat.prod())
X=np.array([9,16,27])
print(np.sqrt(X))
print(np.log(X))
```

```
mean =  1.8333333333333333
Variance =  11.805555555555557
Standard Deviation =  3.435921354681384
min =  -5
max =  6
sum =  11
product =  -720
[3.         4.         5.19615242]
[2.19722458 2.77258872 3.29583687]
```

```
#Numpy - Mathematical and Statistical functions on NumPy Arrays - Sum(across axes)
import numpy as np
Z= np.arange(18).reshape(2, 3, 3)
print(Z)
print( Z.sum(axis=2))
```

```
[[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]]

 [[ 9 10 11]
  [12 13 14]
  [15 16 17]]]
[[ 3 12 21]
 [30 39 48]]
```

```
#Numpy - Mathematical and Statistical functions on NumPy Arrays - Transpose
import numpy as np
N = np.arange(6).reshape(3,2)
```

```
print(N)
print("Transpose = ", N.T)
```

```
[[0 1]
 [2 3]
 [4 5]]
Transpose =  [[0 2 4]
 [1 3 5]]
```

```
#Numpy - Broadcasting in NumPy Arrays
import numpy as np
X_broad=np.ones((3,3))
print(X_broad)
Y_broad = np.arange(3)
print(Y_broad)
Z_broad = X_broad + Y_broad
print (Z_broad)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[0 1 2]
[[1. 2. 3.]
 [1. 2. 3.]
 [1. 2. 3.]]
```

NumPy Array Copy vs View

The Difference Between Copy and View ● The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array. ● The copy owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy. ● The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```

✓ 0s completed at 2:26 PM