CHANDAN KUMAR

Registration ID: GO_STP_13267

1. Import the numpy package under the name np and Print the numpy version and the configuration

In [2]:
```
import numpy as np
print(np.__version__)
np.show_config()
```

```
1.18.1
blas_mkl_info:
  NOT AVAILABLE
blis_info:
  NOT AVAILABLE
openblas_info:
    library_dirs = ['C:\\projects\\numpy-wheels\\numpy\\build\\openblas_inf
o']
    libraries = ['openblas_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
blas_opt_info:
    library_dirs = ['C:\\projects\\numpy-wheels\\numpy\\build\\openblas_inf
o']
    libraries = ['openblas_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
lapack_mkl_info:
  NOT AVAILABLE
openblas_lapack_info:
    library_dirs = ['C:\\projects\\numpy-wheels\\numpy\\build\\openblas_lapac
k_info']
    libraries = ['openblas_lapack_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
lapack_opt_info:
    library_dirs = ['C:\\projects\\numpy-wheels\\numpy\\build\\openblas_lapac
k_info']
    libraries = ['openblas_lapack_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
```

2. Create a null vector of size 10

In [2]:
```
suy = np.zeros(10)
print(suy)
```

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

3. Create Simple 1-D array and check type and check data types in array

In [6]:
```
kp = np.array([5, 8, 12,14,90])
print(kp)
print(type(kp))
print(kp.dtype)
```

```
[ 5  8 12 14 90]
<class 'numpy.ndarray'>
int32
```

4.How to find number of dimensions, bytes per element and bytes of memory used?

In [10]:
```
print(kp.ndim)  #dimension
print(kp.itemsize)  #Number of bytes for each element
```

```
1
5
4
```

In [ ]:
```
```

In [13]:
```
print("%d bytes" % (kp.size * kp.itemsize)) #memory size of any array
print(kp.nbytes) #Total bytes consumed by the elements
```

```
20 bytes
20
```

5. Create a null vector of size 10 but the fifth value which is 1

In [14]:
```
mafia = np.zeros(10)
mafia[4] = 1
print(mafia)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

6. Create a vector with values ranging from 10 to 49

In [15]:
```
suy = np.arange(10,50)
print(suy)
```

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]
```

7. Reverse a vector (first element becomes last)

In [16]:
```
suy1 = np.arange(12)
suy1 = suy1[::-1]
print(suy1)
```

```
[11 10  9  8  7  6  5  4  3  2  1  0]
```

8. Create a 3x3 matrix with values ranging from 0 to 8

In [17]:
```
vin = np.arange(9).reshape(3, 3)
print(vin)
```

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

9. Find indices of non-zero elements from [1,2,0,0,4,0]

In [18]:
```
teju = np.nonzero([1,2,0,0,4,0])
print(teju)
```

```
(array([0, 1, 4], dtype=int64),)
```

10. Create a 3x3 identity matrix

```
In [19]: kp2 = np.eye(3)
         print(kp2)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

11. Create a 3x3x3 array with random values

```
In [20]: vine = np.random.random((3,3,3))
         print(vine)
```

```
[[[0.45828749 0.70766179 0.79094543]
  [0.39583169 0.03627437 0.08425765]
  [0.54773361 0.46118071 0.66795309]]

 [[0.7060931  0.61704544 0.88292139]
  [0.79110916 0.51674125 0.25050999]
  [0.3554896  0.01912211 0.03823314]]

 [[0.3746411  0.27581089 0.09224528]
  [0.27883888 0.92836859 0.82046903]
  [0.63265754 0.45625584 0.05547714]]]
```

12. Create a 10x10 array with random values and find the minimum and maximum values

```
In [21]: vin = np.random.random((10,10))
         vinmin, vinmax = vin.min(), vin.max()
         print(vinmin, vinmax)
```

```
0.0031606420233230015 0.9965609123364018
```

13. Create a random vector of size 30 and find the mean value

```
In [22]: vin1 = np.random.random(30)
         yp = vin1.mean()
         print(yp)
```

```
0.5000035131608737
```

14. Create a 2d array with 1 on the border and 0 inside

In [23]:
```
don = np.ones((10,10))
don[1:-1,1:-1] = 0
print(don)
```

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
```

15. How to add a border (filled with 0's) around an existing array?

In [24]:
```
suy7 = np.ones((10,10))
suy7 = np.pad(suy7, pad_width=1, mode='constant', constant_values=0)
print(suy7)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

In [25]:
```
#Method-2  (fancy indexing used)
suy7[:, [0, -1]] = 0
suy7[[0, -1], :] = 0
print(suy7)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

16. How to Accessing/Changing specific elements, rows, columns, etc in Numpy array?

Example -

[[1 2 3 4 5 6 7]

[8 9 10 11 12 13 14]]

Get 13, get first row only, get 3rd column only, get [2, 4, 6], replace 13 by 20

In [3]:
```
b = np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
b[1,5:-1] #getting value 13
```
Out[3]:
```
array([13])
```

In [4]:
```
b[0,] #getting first row only
```
Out[4]:
```
array([1, 2, 3, 4, 5, 6, 7])
```

In [5]:
```
b[0:,2:3] #getting 3rd column only
```
Out[5]:
```
array([[ 3],
       [10]])
```

In [6]:
```
b[0,1::2]   #get [2, 4, 6],
```
Out[6]:
```
array([2, 4, 6])
```

In [7]:
```
#replace 13 by 20
b[b == 13] = 20
print(b)
```
```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 20 14]]
```

17. How to Convert a 1D array to a 2D array with 2 rows

In [28]:
```
suyash = np.array([0, 1, 2, 3, 4, 15,16, 17, 18, 19])
vineet = np.reshape(suyash, (2,5))
print(vineet)
```
```
[[ 0  1  2  3  4]
 [15 16 17 18 19]]
```

18 . Create the following pattern without hardcoding. Use only numpy functions and the below input array a.

Input:

a = np.array([1,2,3]) ` Desired Output:

> array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])

```
In [32]: a = np.array([1,2,3])
         np.r_[np.repeat(a, 3), np.tile(a, 3)]
```

```
Out[32]: array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])
```

19. Write a program to show how Numpy taking less memory compared to Python List?

```
In [31]: import numpy as np
         import time
         import sys
         suyash1 = np.arange(100)
         print("Size of NumPy array: ", suyash1.itemsize * suyash1.itemsize)
         vineet = range(0, 10)
         print("Size of list: ", sys.getsizeof(1)*len(vineet))
```

```
Size of NumPy array:  16
Size of list:  280
```

Conclusion: NumPy array consumes less memory as compared to the Python list.

20. Write a program to show how Numpy taking less time compared to Python List?

In [30]:
```
import numpy
import time
size = 1000000
vin1 = range(size)
vin2 = range(size)

suy1 = numpy.arange(size)
suy2 = numpy.arange(size)


initialTime = time.time()
resultantList = [(a * b) for a, b in zip(vin1, vin2)]

print("Time taken by Lists :",(time.time() - initialTime),"seconds")


initialTime = time.time()
resultantArray = suy1 * suy2

print("Time taken by NumPy Arrays :",(time.time() - initialTime),"seconds")
```

```
Time taken by Lists : 0.15259051322937012 seconds
Time taken by NumPy Arrays : 0.001995086669921875 seconds
```

Compare numpy and list on the basic of speed

In [33]:
```
import numpy
import time
size = 1000000
vin1 = range(size)
vin2 = range(size)

suy1 = numpy.arange(size)
suy2 = numpy.arange(size)


initialTime = time.time()
resultantList = [(a + b) for a, b in zip(vin1, vin2)]

print("Time taken by Lists :",(time.time() - initialTime),"seconds")


initialTime = time.time()
resultantArray = suy1 + suy2

print("Time taken by NumPy Arrays :",(time.time() - initialTime),"seconds")
```

```
Time taken by Lists : 0.13868260383605957 seconds
Time taken by NumPy Arrays : 0.012601613998413086 seconds
```

Conclusion from above is numpy take less time and speed

Thank You