In [1]:

```
import math
```

In [2]:

```
dir(math)
```

...

In [3]:

```
math.sqrt(25)
```

...

In [4]:

```
math.pow(6,4)
```

...

In [7]:

```
math.pi
```

...

In [8]:

```
math.gcd(9,19)
```

...

In [9]:

```
import random
```

In [10]:

```
dir(random)
```

...

Numpy

- One of the most efficient modules of data science used for scientific computation
- Numpy stands for NumericalPython
- numpy is the module
- we can gather/collect the data and arranged in array format

In [11]:

```
import numpy as np
```

In [13]:

```
np.__version__ # version of numpy module
```

```
...
```

array()

- we will arrange in array format
- matrix format
- array is the sub module of numpy
- np.array(ele)
- array is immutable
- homogenous data structure

array()

- numpy.array(data)

In [16]:

```
# conversion of string into array
st=input("string:")
ar=np.array(st)
print(ar)
```

```
...
```

In [17]:

```
ar
```

```
...
```

In [19]:

```
# conversion of range(values) into the array
rn=range(10)
print(np.array(rn))
```

```
...
```

In [23]:

```
rn=range(10,28)
print(np.array(rn)) # 1d array:row matrix
```

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27]
```

In [24]:

```python
print(np.array(range(10,40,5)))
```

```
[10 15 20 25 30 35]
```

In [25]:

```python
# convert the tuple/list/set into array
tp=(1,3,'python',4,5,'hi','hey')
np.array(tp)
```

...

In [28]:

```python
li=list(map(int,input().split()))
ar=np.array(li)
print(li) # list
print(ar) # nd array
```

```
90 65 34 78 23 9 4 6 7 12
[90, 65, 34, 78, 23, 9, 4, 6, 7, 12]
[90 65 34 78 23  9  4  6  7 12]
```

In [29]:

```python
# dict :paired @ 2d format
# array:1d
```

In [32]:

```python
# set
s=set(li)
sar=np.array(s)
print(sar)
```

...

In [33]:

```python
type(sar)
```

Out[33]:

```
numpy.ndarray
```

In [34]:

```python
# Ndimensional matrix

```

In [35]:

```
# list of lists/tuples
mul=[[1,2],[3,4],[5,6]]
alr=np.array(mul)
print(alr) # 2d array:3x2
```

...

In [36]:

```
tps=([3,4,5],['a','e','h'])
print(np.array(tps))
```

...

In [37]:

```
# create 3x3
mt=np.array([[9,12,34],[89,45,67],(9,34,15)],dtype='float')
mt
```

...

In [39]:

```
mt.ndim # to check the dimensions
```

...

In [40]:

```
mt.dtype #data type of each item
```

...

In [41]:

```
mt.itemsize # item size of each item
```

...

In [42]:

```
mt.shape # (rows,col)
```

...

In [43]:

```
mt.size # total no.of elements
```

...

In [45]:

```
#### multi dimensional
#we can create 32 dimensional array using numpy
```

In [46]:

```
mt
```

...

In [52]:

```
new=np.array([1,2,3,4,5],ndmin=5,
             dtype='str')
print(new)
```

...

In [53]:

```
# identity matrix
# ones
# zeros
# full
# fill
# diag
# linspace
```

In [55]:

```
eye=np.eye((3)) # identity matrix
eye
```

...

In [57]:

```
eye=np.eye((3),4) # identity matrix
eye
```

...

In [60]:

```
# ones matrix
one=np.ones(3,dtype='int')
one
```

...

In [62]:

```
# ones matrix
one=np.ones((3,3)) # ones matrix
one
```

...

In [63]:

```
# zeros matrix
z=np.zeros(3)
z
```

...

In [64]:

```
# zeros matrix
z=np.zeros((4,3))
z
```

...

In [69]:

```
# full and fill
fl=np.full((3),'hi') # row with ele
fl
```

...

In [70]:

```
# full and fill
fl=np.full((4,3),5) # shape with ele
fl
```

...

In [79]:

```
fl.fill(9)
```

In [78]:

```
fl.fill('hi') # doesn't support
```

...

In [80]:

```
fl
```

...

arange()

- creates the ndimensional array with formal values starts from 0
- np.arange()

In [83]:

```
arn=np.arange(18) # works similar to range
print(arn)
```

```
...
```

In [84]:

```
type(arn)
```

```
...
```

In [85]:

```
print(np.arange(10,45))
```

```
...
```

In [92]:

```
rn=np.arange(10,45,5)
rn.dtype='float'
```

```
...
```

In [94]:

```
ar.dtype='float'
```

In [95]:

```
ar
```

```
...
```

In [89]:

```
help(np.arange)
```

```
...
```

In [90]:

```
help(np.array)
```

```
...
```

In [96]:

```
# linspace
# linear space among the interval
```

In [97]:

```
fl
```

```
...
```

In [104]:

```
for row in fl: # depends rows
    print(row) # be default
```

...

In [107]:

```
np.linspace(1,4) # 50 paritions
```

...

In [108]:

```
1,2,3,4 :50paritions
```

...

In [109]:

```
4/50
```

...

In [110]:

```
print(np.linspace(2,6,9))
```

```
[2.  2.5 3.  3.5 4.  4.5 5.  5.5 6. ]
```

In [111]:

```
2,3,4,5,6:5/9
```

Out[111]:

```
0.5555555555555556
```

reshape()

- reshapes the existed array
- reshape(row,col)

In [114]:

```
ln=np.linspace(5,10,12).reshape(3,4) # 50
print(ln)
```

...

In [115]:

```
n=np.arange(28).reshape(7,4)
print(n)
```

...

In [116]:

```
432: values
    # pairs

```

...

In [121]:

```
kl=np.arange(432).reshape(18,-1) # unknown
print(kl)
```

...

In [122]:

```
print(np.arange(45).reshape(-1,9))
```

...

In [126]:

```
# 3d array
# 3 dimensions: (no.of arrays,rows,col)
xy=np.arange(45).reshape(3,5,3)
print(xy)
```

...

In [124]:

```
xy[0] # first array
```

...

In [127]:

```
xy[1]
```

...

In [132]:

```
xyz=np.array([[[1,2],[3,4]],[['a','e'],['o','p']]])
xyz

```

...

In [133]:

```
xyz.ndim
```

Out[133]:

```
3
```

In [138]:

```
new=arn.reshape(6,3)
new
```

...

In [139]:

```
new[0]
```

...

In [140]:

```
new[4] # 5th row
```

...

In [141]:

```
new[3][1] # 2d indexing
```

...

In [142]:

```
# slicing
new[:4]
```

...

In [144]:

```
new[2:4]
```

...

In [146]:

```
 #alternate rows
new[::2]
```

...

In [147]:

```
# columns
new[:,2]
```

...

In [150]:

```
new[2][1:] # 3rd row
```

Out[150]:

```
array([7, 8])
```

In [152]:

```
new[:,1:]
```

```
...
```

In [153]:

```
# alternate col in alternate rows
new[::2,::2]
```

```
...
```

In [154]:

```
new
```

```
...
```

random

- generates the random values
- np.random()

np.random()

- np.random.randint(stop)
- np.random.randint(st,sp)
- np.random.randint(st,sp,stp)
- np.random.random()
- np.random.rand()
- np.random.randd()

In [164]:

```
print(np.random.randint(10))
# random digit in the range
```

```
9
```

In [173]:

```
np.random.randint(30,80)
```

```
...
```

In [176]:

```
np.random.randint(-9,10)
```

```
...
```

In [178]:

```
np.random.randint(-100,-8)
```

...

In [189]:

```
np.random.randint(-200,-17,1000).reshape(20,50)
```

...

In [194]:

```
np.random.random((2,4))
# random floating values from 0 to 1
```

...

In [196]:

```
np.random.rand(3,5)
```

...

In [203]:

```
np.random.randn(10,20)
```

...

In [204]:

```
# fancy indexing
new
```

...

In [214]:

```
knew=np.random.randint(100,500,25).reshape(5,-1)
print(knew)
```

...

In [220]:

```
# array whose values >340
knew[knew>320]
```

...

broadcasting

- apply scalar quantity on arrays
- causes big difference

In [222]:

```
knew+19
```

...

In [223]:

```
#### arithmetic operations
addition,subtraction
```

...

In [224]:

```
fml=np.arange(100,125).reshape(5,5)
fml
```

...

In [225]:

```
print("shiva")

```

```
shiva
```

In [227]:

```
num=int(input())
for dig in range(1,11):
    print(num,"x",dig,"=",num*dig)
```

...

In [228]:

```
knew+fml
```

...

In [229]:

```
knew-fml
```

...

In [230]:

```
knew*fml
```

...

In [231]:

```
knew.transpose() # rows arranged as cols
```

...

In [234]:

```
knew.dot(fml) # dot product (vectors)
```

```
...
```

In [237]:

```
knew.min() # minimum value of knew
```

```
...
```

In [239]:

```
knew.max()
```

```
...
```

In [240]:

```
knew.sum() # summation of all the values of knew
```

```
...
```

In [241]:

```
knew.mean()
```

```
...
```

In [242]:

```
knew.std()
```

```
...
```

Scientific computation

- logarithms and exponentials

In [243]:

```
np.log(10)
```

```
...
```

In [244]:

```
np.log([1,2,3,4,10])
```

```
...
```

In [245]:

```
np.log(knew)
```

```
...
```

In [246]:

```
dir(np)
```

```
 'unravel_index',
 'unsignedinteger',
 'unwrap',
 'use_hugepage',
 'ushort',
 'vander',
 'var',
 'vdot',
 'vectorize',
 'version',
 'void',
 'void0',
 'vsplit',
 'vstack',
 'warnings',
 'where',
 'who',
 'zeros',
 'zeros_like']
```

In [249]:

```
help(np.invert)
```

...

In [250]:

```
np.invert(knew)
```

...

In [251]:

```
knew
```

...

In [252]:

```
np.exp(8)
```

...

In [253]:

```
np.exp([8,3,4])
```

...

In [255]:

```
np.exp(fml)
```

...

vectorized function

- takes a python func that returns the vectorized function which is used to speed up the python code without loop

In [256]:

```python
def greater(a,b):
    if a>b:
        return a
    else:return b
```

In [259]:

```python
greater(14,5)
```

...

In [258]:

```python
greater(9,13)
```

...

In [260]:

```python
greater([4,5,19],[10,8,4])
```

Out[260]:

```
[10, 8, 4]
```

In [263]:

```python
g=np.vectorize(greater)
g([43,5,61,19],[9,10,8,4])
```

...

In [267]:

```python
x,y=[4,5,19],[10,8,4]
res=[]
for a,b in zip(x,y): # multiple iterables
    if a>b:
        res.append(a)
    else:
        res.append(b)
res
```

...

In [ ]:

```python

```