```
#Task 1: Create 3D array having two rows and two columns and 10 parallel
#metrics.
import numpy as np


#array1 =np.array([[[1,2,3],[4,5,6],[7,8,9]],[[1,2,3],[4,5,6],[7,8,9]]])

array2 =10 * np.random.rand(2,2,10).reshape(2,2,10)

arr = np.floor(array2)
arr = arr.astype('i')
print(arr)

print(array1.ndim)
```

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

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

In [64]:
```
#Make a Numpy array having 5 rows and 5 columns using ones()
#function.
#After that convert it into the following shape:
#[[[[1., 9., 8., 7., 6.],
#[1., 4., 5., 8., 5.],
#[5., 2., 4., 0., 6.],
#[1., 0., 5., 6., 6.],
#[7., 8., 8., 9., 8.]]]]
from scipy import stats

array1 = np.ones((1,1,5,5))


array1[0,0,0] =[1, 9, 8, 7, 6]
array1[0,0,1] =[1., 4., 5., 8., 5.]
array1[0,0,2] =[5., 2., 4., 0., 6]
array1[0,0,3] =[1., 0., 5., 6., 6]
array1[0,0,4] =[7., 8., 8., 9., 8.]
print(array1)

Row0 = np.mean(array1[0,0,0])
print(" median row1 ",Row0)
```

```
        Row1 = np.mean(array1[0,0,1])
        print(" median row2 ",Row1)

        Row2 = np.mean(array1[0,0,1])
        print(" median row3 ",Row2)

        Row3 = np.mean(array1[0,0,2])
        print(" median row4 ",Row3)

        Row4 = np.mean(array1[0,0,3])
        print(" median row5 ",Row4)

        print('')
        Row0 = np.median(array1[0,0,0])
        print(" median row1 ",Row0)

        Row1 = np.median(array1[0,0,1])
        print(" median row2 ",Row1)

        Row2 = np.median(array1[0,0,1])
        print(" median row3 ",Row2)

        Row3 = np.median(array1[0,0,2])
        print(" median row4 ",Row3)

        Row4 = np.median(array1[0,0,3])
        print(" median row5 ",Row4)


        print('')
        Row0 = stats.mode(array1[0,0,0])
        print(" mode row1 ",Row0)

        Row1 = stats.mode(array1[0,0,1])
        print(" mode row2 ",Row1)

        Row2 = stats.mode(array1[0,0,1])
        print(" mode row3 ",Row2)

        Row3 = stats.mode(array1[0,0,2])
        print(" mode row4 ",Row3)

        Row4 = stats.mode(array1[0,0,3])
        print(" mode row5 ",Row4)
```

```
[[[[1. 9. 8. 7. 6.]
   [1. 4. 5. 8. 5.]
   [5. 2. 4. 0. 6.]
   [1. 0. 5. 6. 6.]
   [7. 8. 8. 9. 8.]]]]
 median row1  6.2
 median row2  4.6
 median row3  4.6
 median row4  3.4
 median row5  3.6

 median row1  7.0
 median row2  5.0
 median row3  5.0
 median row4  4.0
 median row5  5.0

 mode row1  ModeResult(mode=array([1.]), count=array([1]))
 mode row2  ModeResult(mode=array([5.]), count=array([2]))
 mode row3  ModeResult(mode=array([5.]), count=array([2]))
 mode row4  ModeResult(mode=array([0.]), count=array([1]))
 mode row5  ModeResult(mode=array([6.]), count=array([2]))
```

```
C:\Users\Barcha\AppData\Local\Temp\ipykernel_1596\715248882.py:54: FutureWarning: Unlike other reduction funct
ions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In
SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over wh
ich the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` t
o True or False to avoid this warning.
  Row0 = stats.mode(array1[0,0,0])
C:\Users\Barcha\AppData\Local\Temp\ipykernel_1596\715248882.py:57: FutureWarning: Unlike other reduction funct
ions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In
SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over wh
ich the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` t
o True or False to avoid this warning.
  Row1 = stats.mode(array1[0,0,1])
C:\Users\Barcha\AppData\Local\Temp\ipykernel_1596\715248882.py:60: FutureWarning: Unlike other reduction funct
ions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In
SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over wh
ich the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` t
o True or False to avoid this warning.
  Row2 = stats.mode(array1[0,0,1])
C:\Users\Barcha\AppData\Local\Temp\ipykernel_1596\715248882.py:63: FutureWarning: Unlike other reduction funct
ions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In
SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over wh
ich the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` t
o True or False to avoid this warning.
  Row3 = stats.mode(array1[0,0,2])
C:\Users\Barcha\AppData\Local\Temp\ipykernel_1596\715248882.py:66: FutureWarning: Unlike other reduction funct
ions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In
SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over wh
ich the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` t
o True or False to avoid this warning.
  Row4 = stats.mode(array1[0,0,3])
```

In [71]:
```
#Task 2: Make a Numpy array having 5 rows and 5 columns using ones()
#function.
#After that convert it into the following shape:
#[[1., 1., 1., 1., 1.],
#[1., 0., 0., 0., 1.],
#[1., 0., 0., 0., 1.],
#[1., 0., 0., 0., 1.],
#[1., 1., 1., 1., 1.]]

array1 = np.ones((5,5))

array1[0] =[1., 1., 1., 1., 1.]
array1[1] =[1., 0., 0., 0., 1]
array1[2] =[1., 0., 0., 0., 1.]
```

```
array1[3] =[1., 0., 0., 0., 1.]
array1[4] =[1., 1., 1., 1., 1.]
print(array1)

Row0 = np.mean(array1[0])
print(" median row1 ",Row0)

Row1 = np.mean(array1[1])
print(" median row2 ",Row1)

Row2 = np.mean(array1[2])
print(" median row3 ",Row2)

Row3 = np.mean(array1[3])
print(" median row4 ",Row3)

Row4 = np.mean(array1[4])
print(" median row5 ",Row4)

print('')
Row0 = np.median(array1[0])
print(" median row1 ",Row0)

Row1 = np.median(array1[1])
print(" median row2 ",Row1)

Row2 = np.median(array1[2])
print(" median row3 ",Row2)

Row3 = np.median(array1[3])
print(" median row4 ",Row3)

Row4 = np.median(array1[4])
print(" median row5 ",Row4)


print('')
Row0 = stats.mode(array1[0])
print(" mode row1 ",Row0)

Row1 = stats.mode(array1[1])
print(" mode row2 ",Row1)

Row2 = stats.mode(array1[2])
print(" mode row3 ",Row2)
```

```
Row3 = stats.mode(array1[3])
print(" mode row4 ",Row3)

Row4 = stats.mode(array1[4])
print(" mode row5 ",Row4)
```

```
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
 median row1  1.0
 median row2  0.4
 median row3  0.4
 median row4  0.4
 median row5  1.0

 median row1  1.0
 median row2  0.0
 median row3  0.0
 median row4  0.0
 median row5  1.0

 mode row1  ModeResult(mode=array([1.]), count=array([5]))
 mode row2  ModeResult(mode=array([0.]), count=array([3]))
 mode row3  ModeResult(mode=array([0.]), count=array([3]))
 mode row4  ModeResult(mode=array([0.]), count=array([3]))
 mode row5  ModeResult(mode=array([1.]), count=array([5]))
```

```
C:\Users\Barcha\AppData\Local\Temp\ipykernel_1596\2749532.py:52: FutureWarning: Unlike other reduction functio
ns (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In Sc
iPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over whic
h the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to
True or False to avoid this warning.
  Row0 = stats.mode(array1[0])
C:\Users\Barcha\AppData\Local\Temp\ipykernel_1596\2749532.py:55: FutureWarning: Unlike other reduction functio
ns (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In Sc
iPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over whic
h the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to
True or False to avoid this warning.
  Row1 = stats.mode(array1[1])
C:\Users\Barcha\AppData\Local\Temp\ipykernel_1596\2749532.py:58: FutureWarning: Unlike other reduction functio
ns (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In Sc
iPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over whic
h the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to
True or False to avoid this warning.
  Row2 = stats.mode(array1[2])
C:\Users\Barcha\AppData\Local\Temp\ipykernel_1596\2749532.py:61: FutureWarning: Unlike other reduction functio
ns (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In Sc
iPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over whic
h the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to
True or False to avoid this warning.
  Row3 = stats.mode(array1[3])
C:\Users\Barcha\AppData\Local\Temp\ipykernel_1596\2749532.py:64: FutureWarning: Unlike other reduction functio
ns (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In Sc
iPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over whic
h the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to
True or False to avoid this warning.
  Row4 = stats.mode(array1[4])
```

In [77]:
```
#Task 3: Create 3D array having three rows and five columns and 10 parallel
#metrics. Convert all elements of second rows equal to 5.

array = 10*np.random.rand(3,5,10)
array = np.floor(array)
array = array.astype('i')

array[0,1]=5
array[1,1]=5
array[2,1]=5


print(array)
```

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

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

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

In [89]:
```
#Task 4: Make a Numpy array of 3x3x3 of random numbers and place 1 if the
#element is odd and 0 if element is even.
#Hint: np.random.randint(start,end,(size))

array =10*np.random.rand(3,3,3)
array = array.astype('i')
print(array)
print(' ')
print('')
print(' ')
array = np.where(array%2 == 0,0,1)
print(array)
```

```
[[[4 4 8]
  [0 0 8]
  [4 3 7]]

 [[0 0 3]
  [7 0 4]
  [0 1 2]]

 [[8 7 2]
  [1 4 4]
  [2 3 8]]]
after
[[[0 0 0]
  [0 0 0]
  [0 1 1]]

 [[0 0 1]
  [1 0 0]
  [0 1 0]]

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

In [106]:
```
#Task 5: Convert a 4D Numpy array having 24 elements into a 2D array having
#log of each element.

array = (10*np.random.rand(1,1,4,6))+1
array = array.astype('i')
print('array' , array)
print('')
array = array.reshape(2,-1)
print('array')
print(array)
array = np.log(array)
print('')
print('log')
print(array)
```

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

array
[[ 5  2  6  4  8  5  7  1  3  2  3  2]
 [ 7  9  5 10  1  2 10  8  1  1  2  6]]

log
[[1.60943791 0.69314718 1.79175947 1.38629436 2.07944154 1.60943791
  1.94591015 0.         1.09861229 0.69314718 1.09861229 0.69314718]
 [1.94591015 2.19722458 1.60943791 2.30258509 0.         0.69314718
  2.30258509 2.07944154 0.         0.         0.69314718 1.79175947]]
```

In [126]:

```
#Task 6-A: Make a list of 1000 elements between 0 and 1. Calculate square of

#each element and print time taken for execution. Repeat it for Numpy and
#compare time.
import time

start=time.time()
array = np.random.rand(10000)
print(array)
array = np.square(array)
stop=time.time()

totaltime = stop - start
print('total with numpy',totaltime)


start_time =time.time()
mylist = list(range(1, 10001))

[i**2 for i in mylist]
[1, 4, 9, 16]

stop_list =time.time()

total = stop_list - start_time
print('total without numpy',total)
```

```
[0.3578502  0.37518314 0.05506896 ... 0.24636479 0.34707781 0.77907503]
total with numpy 0.001994609832763672
total without numpy 0.008975744247436523
```

In [129]:
```
#Task 6-B: Increase elements up to 10000 and 1000000 and see results.
import time

start=time.time()
array = np.random.rand(1000000)
print(array)
array = np.square(array)
stop=time.time()

totaltime = stop - start
print('total with numpy',totaltime)


start_time =time.time()
mylist = list(range(1, 1000001))

[i**2 for i in mylist]
[1, 4, 9, 16]

stop_list =time.time()

total = stop_list - start_time
print('total without numpy',total)
```

```
[0.42851219 0.87462113 0.94502737 ... 0.17339744 0.60466328 0.32227323]
total with numpy 0.027646064758300078
total without numpy 0.45430636405944824
```