Usama Arif Rollno 14

Task 1: Create 3D array having two rows and two columns and 10 parallel metrics.

In [6]:
```
import numpy as np
np.random.seed(5)
array_3d=np.random.randint(10,size=(2,2,10))
array_3d
```

Out[6]:
```
array([[[3, 6, 6, 0, 9, 8, 4, 7, 0, 0],
        [7, 1, 5, 7, 0, 1, 4, 6, 2, 9],
        [9, 9, 9, 1, 2, 7, 0, 5, 0, 0]],

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

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

Task 2: Make a Numpy array having 5 rows and 5 columns using ones() function.Find Mean median mode of All rows separately

In [71]:
```
ones_array=np.zeros((3,3))
padding_pattern=((1,1),(1,1))
required_array=np.pad(ones_array,padding_pattern,mode='constant',constant_
required_array
```

Out[71]:
```
array([[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.]])
```

In [85]:
```
mean=np.mean(required_array,axis=1)
mean
```

Out[85]:
```
array([1. , 0.4, 0.4, 0.4, 1. ])
```

In [97]:
```
median=np.median(required_array,axis=1)
median
```

Out[97]:
```
array([1., 0., 0., 0., 1.])
```

In [94]:
```
from scipy import stats as s
mode=s.mode(required_array,axis=1).mode[:,0]
mode
```

```
C:\Users\usama\AppData\Local\Temp\ipykernel_9232\170377385.py:2: FutureWarnin
g: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default be
havior 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 which the statistic is taken will be eliminated, and the valu
e None will no longer be accepted. Set `keepdims` to True or False to avoid t
his warning.
  mode=s.mode(required_array,axis=1).mode[:,0]
```

Out[94]:
```
array([1., 0., 0., 0., 1.])
```

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

In [128]:
```
np.random.seed(5)
arr_3d=np.random.randint(10,size=(3,5,10))
arr_3d
```

Out[128]:
```
array([[[3, 6, 6, 0, 9, 8, 4, 7, 0, 0],
        [7, 1, 5, 7, 0, 1, 4, 6, 2, 9],
        [9, 9, 9, 1, 2, 7, 0, 5, 0, 0],
        [4, 4, 9, 3, 2, 4, 6, 9, 3, 3],
        [2, 1, 5, 7, 4, 3, 1, 7, 3, 1]],

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

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

In [136]:
```
arr_3d[:,1:2,:]=[5]
arr_3d
```

Out[136]:
```
array([[[3, 6, 6, 0, 9, 8, 4, 7, 0, 0],
        [5, 5, 5, 5, 5, 5, 5, 5, 5, 5],
        [9, 9, 9, 1, 2, 7, 0, 5, 0, 0],
        [4, 4, 9, 3, 2, 4, 6, 9, 3, 3],
        [2, 1, 5, 7, 4, 3, 1, 7, 3, 1]],

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

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

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.

In [147]:
```
np.random.seed(4)
arr_3=np.random.randint(10,size=(3,3,3))
replaced_arr=np.where(arr_3%2==0,0,1)
print('original array \n',arr_3)
print('replaced array with 0 and 1')
print('\n')
print(replaced_arr)

```

```
original array
 [[[7 5 1]
  [8 7 8]
  [2 9 7]]

 [[7 7 9]
  [8 4 2]
  [6 4 3]]

 [[0 7 5]
  [5 9 6]
  [6 8 2]]]
replaced array with 0 and 1


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

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

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

Task 5: Convert a 4D Numpy array having 24 elements into a 2D array having log of each element.

In [156]:
```
arr_4d=np.random.randint(10,size=(2,3,2,2))
reshaped_array=np.reshape(arr_4d,(8,3))
print('original array \n', arr_4d)
print('reshaped array \n',reshaped_array)
log=np.log(reshaped_array)
print('log of reshaped array',reshaped_array)
```

```
original array
 [[[[2 2]
   [1 3]]

  [[7 4]
   [0 5]]

  [[7 3]
   [8 5]]]


 [[[9 1]
   [5 4]]

  [[3 9]
   [5 6]]

  [[6 1]
   [0 5]]]]
reshaped array
 [[2 2 1]
 [3 7 4]
 [0 5 7]
 [3 8 5]
 [9 1 5]
 [4 3 9]
 [5 6 6]
 [1 0 5]]
log of reshaped array [[2 2 1]
 [3 7 4]
 [0 5 7]
 [3 8 5]
 [9 1 5]
 [4 3 9]
 [5 6 6]
 [1 0 5]]

C:\Users\usama\AppData\Local\Temp\ipykernel_9232\337617289.py:5: RuntimeWarni
ng: divide by zero encountered in log
  log=np.log(reshaped_array)
```

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.

In [223]:
```
import time
numbers = [i / 1000 for i in range(1000)]

def calculate_squares():
    squared_list = [x**2 for x in numbers]


start_time = time.time()
calculate_squares()
end_time = time.time()

execution_time = end_time - start_time
print(f"Execution time: {execution_time} seconds")

```

```
Execution time: 0.0009982585906982422 seconds
```

using numpy

In [224]:
```
arr_5=np.linspace(0,1,1000)
sq=np.square(arr_5)
print('time taken for calculating sqaure\n ')

%timeit sq
#print('\n')
#print('sqaure of each elemnet :' sq)
```

```
time taken for calculating sqaure

45.7 ns ± 2.35 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops eac
h)
```

Task 6-B: Increase elements up to 10000 and 1000000 and see results.

In [225]:
```
import time
numbers = [i / 1000 for i in range(10000)]

def calculate_squares():
    squared_list = [x**2 for x in numbers]


start_time = time.time()
calculate_squares()
end_time = time.time()

execution_time = end_time - start_time
print(f"Execution time: {execution_time} seconds")
```

```
Execution time: 0.00299835205078125 seconds
```

using numpy

In [227]:
```
arr_5=np.linspace(0,1,10000)
sq=np.square(arr_5)
print('time taken for calculating sqaure\n ')

%timeit sq
#print('\n')
#print('sqaure of each elemnet :' sq)
```

```
time taken for calculating sqaure

44.7 ns ± 2.48 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops eac
h)
```

numpy took less time

In [ ]:
```

```