1. Create a 10x5x4 array A containing 200 values logarithmically spaced between 10–3 and 10 7 . Print A[3,0,2], A[5,4,1], and the last element of A.

2. Create an array B containing and set all the elements of B larger than 4 equal to 4. Print B

3. Create a 10x10 array C that contains zeros in all elements for which either index is odd and ones in all other elements. Print C. Exercise 1: manipulating NumPy arrays

In [10]:

```
import numpy as np
# 1
A = np.logspace(-3, 7, num=200).reshape(10, 5, 4)

A_1 = A[3, 0, 2]
A_2 = A[5, 4, 1]
A_last = A[-1, -1, -1]

# 2
print("Value of A[3,0,2] ->" ,A_1 ,"\nValue of A[5,4,1] ->", A_2, " \nValue of Last


B = np.array([[-2,8,10,1],[17,9,2,0],[1,6,-4,10],[3,8,-9,4]])
B[B > 4] = 4

print(B)

# 3
C = np.zeros((10,10))

C[::2,1::2] = 1
C[1::2, ::2] - 1

print(C)
```

```
Value of A[3,0,2] -> 1.3049019780144029
Value of A[5,4,1] -> 757.525025877192
Value of Last A  -> 10000000.0
[[-2  4  4  1]
 [ 4  4  2  0]
 [ 1  4 -4  4]
 [ 3  4 -9  4]]
[[0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Use NumPy arrays to do the following: • Create two arrays x and y containing 10 equally-spaced values between –π and π, then create a 2d grid of values z containing sin(x)*cos(y) at the coordinates (x,y) given by the elements of x and y. (See the reading on meshes or np.indices.) • Find the sum over the innermost 4x4 grid points of z. • Define a function f(x) that returns x 2 if its input is a scalar. If its input is an array, return an array with the same shape containing the squares of the element values. Print f(z) and f(2). • Create a 1D array containing the maximum value of z for each value of y. (Hint: see the help file on np.max for how to control the dimensionality of its output.) Exercise 2: computing with NumPy arrays

In [12]:

```
# creating two arrays with 10 equaly spaced value from -pi to pi

x = np.linspace(-np.pi, np.pi, 10)
y = np.linspace(-np.pi, np.pi, 10)

# 2-d grid
X, Y = np.meshgrid(x, y)

# find z
Z = np.sin(X) * np.cos(Y)

in_sum = Z[3:7, 3:7].sum()
print(in_sum)
def f(x):
    if np.isscalar(x):
        return x ** 2
    else:
        return np.square(x)

f_z = f(Z)
f_2 = f(2)
print("Value of f(z) ->" , f_z, " \nValue of f(2) ->", f_2)
max_val = Z.max(axis = 1)
print(max_val)
```

```
-6.661338147750939e-16
Value of f(z) -> [[1.49975978e-32 4.13175911e-01 9.69846310e-01 7.50000000e-01
  1.16977778e-01 1.16977778e-01 7.50000000e-01 9.69846310e-01
  4.13175911e-01 1.49975978e-32]
 [8.80095168e-33 2.42461578e-01 5.69129177e-01 4.40118067e-01
  6.86453782e-02 6.86453782e-02 4.40118067e-01 5.69129177e-01
  2.42461578e-01 8.80095168e-33]
 [4.52232910e-34 1.24587782e-02 2.92444446e-02 2.26152672e-02
  3.52731162e-03 3.52731162e-03 2.26152672e-02 2.92444446e-02
  1.24587782e-02 4.52232910e-34]
 [3.74939946e-33 1.03293978e-01 2.42461578e-01 1.87500000e-01
  2.92444446e-02 2.92444446e-02 1.87500000e-01 2.42461578e-01
  1.03293978e-01 3.74939946e-33]
 [1.32432122e-32 3.64843511e-01 8.56395844e-01 6.62266666e-01
  1.03293978e-01 1.03293978e-01 6.62266666e-01 8.56395844e-01
  3.64843511e-01 1.32432122e-32]
 [1.32432122e-32 3.64843511e-01 8.56395844e-01 6.62266666e-01
  1.03293978e-01 1.03293978e-01 6.62266666e-01 8.56395844e-01
  3.64843511e-01 1.32432122e-32]
 [3.74939946e-33 1.03293978e-01 2.42461578e-01 1.87500000e-01
  2.92444446e-02 2.92444446e-02 1.87500000e-01 2.42461578e-01
  1.03293978e-01 3.74939946e-33]
 [4.52232910e-34 1.24587782e-02 2.92444446e-02 2.26152672e-02
  3.52731162e-03 3.52731162e-03 2.26152672e-02 2.92444446e-02
  1.24587782e-02 4.52232910e-34]
 [8.80095168e-33 2.42461578e-01 5.69129177e-01 4.40118067e-01
  6.86453782e-02 6.86453782e-02 4.40118067e-01 5.69129177e-01
  2.42461578e-01 8.80095168e-33]
 [1.49975978e-32 4.13175911e-01 9.69846310e-01 7.50000000e-01
  1.16977778e-01 1.16977778e-01 7.50000000e-01 9.69846310e-01
  4.13175911e-01 1.49975978e-32]]
Value of f(2) -> 4
[0.98480775 0.75440651 0.17101007 0.49240388 0.92541658 0.92541658
 0.49240388 0.17101007 0.75440651 0.98480775]
```

Construct two 1D coordinate arrays x and y, each filled with 100 random numbers drawn from a uniform (flat) distribution ranging from 0 to 1 inclusive. Sort the arrays. 2. Construct 2D meshgrids from x and y. 3. Use the meshgrids to construct the 2D uniformly sampled function z(x,y) 4. NumPy provides a histogram function that we can use to find the frequency of occurrence of different values of z. The most common usage is hist, bin_edges = np.histogram(z, bins=10) where the bins argument supplies the number of bins (or a list of bin right-edge values), hist contains the histogram counts, and bin_edges contains the bin edges (its length is len(hist) + 1). 5. Use the histogram function to construct a histogram of z values with bin width 0.1, and print the counts in a table like the one below. 1.) 0.0 0.1 9013 2.) 0.1 0.2 245 3.) 0.2 0.3 .

In [24]:
```
x = np.random.uniform(0, 1, 100)
y = np.random.uniform(0, 1, 100)


x.sort(), y.sort()

X,Y = np.meshgrid(x,y)
```

```python
Z = np.exp(-(X**2 + Y**2) / 0.05)

minz, maxz = Z.min(), Z.max()
bins = np.arange(minz,maxz, 0.1) #width of 0.1
hist, bin_edge = np.histogram(Z, bins)
bin_edge[0] = 0
bin_edge = np.around(bin_edge, 2)
table = [(i+1, bin_edge[i], bin_edge[i+1], hist[i]) for i in range(len(hist))]
table[:10]
```

Out[24]:

```
[(1, 0.0, 0.1, 9054),
 (2, 0.1, 0.2, 295),
 (3, 0.2, 0.3, 191),
 (4, 0.3, 0.4, 144),
 (5, 0.4, 0.5, 116),
 (6, 0.5, 0.6, 67),
 (7, 0.6, 0.7, 42),
 (8, 0.7, 0.8, 31),
 (9, 0.8, 0.9, 38)]
```