Practical 2 Exercises:

In [1]:
```
import numpy as np
```

Question 1

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

In [2]:
```
# Q1
q1 = np.arange(9).reshape(3, 3)
print("q1 is")
print(q1)

# Sample answer:
# a1 is [[0 1 2]
#  [3 4 5]
#  [6 7 8]]
```

```
q1 is
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Question 2

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

In [3]:
```
# Q2
q2 = np.random.random((10, 10))
# print(q2)
print("max value of q2 is", np.max(q2))
print("min value of q2 is", np.min(q2))

# Sample answer:
# max value of a2 is 0.9992937628379465
# min value of a2 is 0.012441575894276191
```

```
max value of q2 is 0.9902116553624828
min value of q2 is 0.03288893521020175
```

Question 3

1. Create a 8x8 matrix and fill it with a checkerboard pattern

In [4]:
```
# Q3
q3 = np.zeros((8, 8))
q3[1::2, ::2] = 1
q3[::2, 1::2] = 1
print("q3 is")
print(q3)

# Sample answer (!!wrong!!):
# a3 is
# [[1. 1. 1. 1. 1. 1. 1. 1.]
#  [0. 0. 0. 0. 0. 0. 0. 0.]
#  [1. 1. 1. 1. 1. 1. 1. 1.]
#  [0. 0. 0. 0. 0. 0. 0. 0.]
#  [1. 1. 1. 1. 1. 1. 1. 1.]
#  [0. 0. 0. 0. 0. 0. 0. 0.]
#  [1. 1. 1. 1. 1. 1. 1. 1.]
#  [0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
q3 is
[[0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]]
```

Question 4

1. Create random vector of size 10 and replace the maximum value by 0

In [5]:
```
# Q4
q4 = np.random.rand(10)
print("original q4 is")
print(q4)
max_index = np.argmax(q4)
q4[max_index] = 0
print("new q4 is")
print(q4)

# Sample answer:
# original a4 is
# [0.32642686 0.842169   0.3270934  0.76815538 0.72293528 0.72772311
#  0.31843191 0.41676299 0.0269001  0.44165806]
# new a4 is
# [0.32642686 0.         0.3270934  0.76815538 0.72293528 0.72772311
#  0.31843191 0.41676299 0.0269001  0.44165806]
```

```
original q4 is
[0.95499748 0.82597215 0.77841279 0.71767224 0.95954109 0.79457117
 0.24178283 0.64083669 0.53328774 0.45344192]
new q4 is
[0.95499748 0.82597215 0.77841279 0.71767224 0.         0.79457117
 0.24178283 0.64083669 0.53328774 0.45344192]
```

Question 5

1. Create a $4 * 4$ identity matrix.

In [6]:
```
# Q5
q5 = np.eye(4)
print("matrix identity q5 is")
print(q5)

# Sample answer:
# matrix identity a5 is
# [[1. 0. 0. 0.]
#  [0. 1. 0. 0.]
#  [0. 0. 1. 0.]
#  [0. 0. 0. 1.]]
```

```
matrix identity q5 is
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Question 6

1. Generate the 2D array

In [7]:
```
# Q6
q6 = np.random.rand(3, 3)
print("2-D array q6 is")
print(q6)

# Sample answer:
# 2-D array a6 is
# [[0.51586491 0.66398948 0.97664544]
#  [0.04570482 0.89286948 0.96746235]
#  [0.58957188 0.62527391 0.15207168]]
```

```
2-D array q6 is
[[0.29299351 0.6801015  0.64070941]
 [0.22896768 0.96508168 0.04578627]
 [0.70248292 0.07067561 0.12894765]]
```

Question 7

1. Generate a random $4 \times 4 \times 4$ array of Gaussianly (Normal) distributed numbers.

In [8]:
```
# Q7
q7 = np.random.randn(4, 4, 4)
print("3-D array q7 is")
print(q7)

# Sample answer:
# 3-D array a7 is
# [[[ 1.24532934 -0.21856698 -0.73979455  0.97734318]
#   [-0.39611546  2.08889299  1.5800086  -1.49495715]
#   [-0.4902839  -0.32530276  0.36206673 -1.81680732]
#   [-0.78322585  0.77552733  0.10999834 -1.99478335]]

#  [[ 0.42654637 -2.00054199 -0.73081536  1.5650001 ]
#   [ 0.52327309  0.44793719 -0.69319204 -0.50252064]
#   [ 0.75281488 -0.98192208 -1.09342367 -0.16395248]
#   [-0.39731477  0.8379485   0.28013044  0.46306382]]

#  [[-0.37884152 -0.76614664 -0.70402131 -0.20758354]
#   [-1.08024839 -0.23268198 -1.57187888 -0.06078278]
#   [-1.20464148 -0.08984497 -0.47294482 -0.68721336]
#   [ 1.32338635 -1.04715013 -0.619475   -0.83355595]]

#  [[ 0.40107191  0.43863124  1.61187035 -1.19535028]
#   [ 0.95272223 -1.06955353 -0.78299704  1.51630909]
#   [-0.74651708  0.84885255  1.47740253  0.06694602]
#   [ 0.4545381  -1.94715794 -1.0914511   0.5649201 ]]]
```

```
3-D array q7 is
[[[-1.81531699e+00 -1.04457366e+00  7.54664969e-01 -3.33505862e-01]
  [-5.67523176e-01 -5.97250656e-01  1.93564977e+00  8.02507882e-01]
  [ 4.48818835e-01  1.26089059e+00  7.62945664e-02  2.09293006e+00]
  [ 6.33947907e-02 -1.01832785e-01  7.44216440e-01  1.81028246e+00]]

 [[ 3.22518455e-01  7.72158524e-01 -8.65445944e-01 -1.19624417e+00]
  [-3.77440508e-01  1.92030315e+00 -9.50797763e-01 -2.14669049e-01]
  [-5.92813264e-02 -1.00317572e-01 -5.70142442e-02  3.39354075e-01]
  [ 1.37640543e+00  7.32334690e-01 -1.18050263e+00 -3.84516477e-01]]

 [[ 5.45419323e-01  7.91991217e-01  8.37163501e-01  1.12767059e+00]
  [ 2.14693785e-01  1.69217186e-01  1.96609625e-03  7.58260193e-01]
  [ 6.99284174e-02 -5.37951235e-01  1.02706257e+00 -6.30223690e-01]
  [ 4.29453040e-01 -1.85204504e+00  1.55170499e+00 -2.74745038e-02]]

 [[ 2.10151459e-01 -1.24571114e+00  2.49989333e-01  1.06552764e+00]
  [-6.15017001e-02  2.17628515e+00 -4.44976476e-01 -1.89893089e-01]
  [-5.97328246e-01 -9.44290973e-01 -2.32800075e+00  6.85795506e-01]
  [ 1.01248907e+00 -7.02785314e-01  1.21018612e+00 -1.40230240e+00]]]
```

Question 8

1. Generate `n` evenly spaced intervals between 0. and 1.

In [9]:
```python
# Q8
# q8 = np.arange(0,10,1) # this is for integer
q8 = np.linspace(0, 1, 11) # this is for non-integer step, such as 0.1
# np.linspace(start, stop, num),
# num: Number of samples to generate. Default is 50. Must be non-negative.
# reference: https://numpy.org/doc/stable/reference/generated/numpy.linspace.html#numpy.

print("q8 is")
print(q8)

# Sample answer:
#  a8 is
# [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

```
q8 is
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

Question 9

1. Create a vector and then reverse the vector (first element becomes last)

In [10]:
```python
# Q9
q9 = np.arange(10)
print("q9 is")
print(q9)
print("the reverse of q9 is")
print(q9[::-1])

# Sample answer:
# a9 is
# [0 1 2 3 4 5 6 7 8 9]
# the reverse of a9 is
# [9 8 7 6 5 4 3 2 1 0]
```

```
q9 is
[0 1 2 3 4 5 6 7 8 9]
the reverse of q9 is
[9 8 7 6 5 4 3 2 1 0]
```