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

1. [10 points] Create a Python array from the following matrix using a combination of keyword arguments

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

(a) dtype in float, int32, float64, and complex128.

(b) dimensions of 3 and 4.

What are the shapes of these arrays?

In [2]:
```
a11, a12, a13, a21, a22, a23, a31, a32, a33 = np.arange(1, 10, 1,dtype=float)
arr = np.array([a11, a12, a13, a21, a22, a23, a31, a32, a33]).reshape(3, 3)
print(arr)
print('\nShape of array: ', arr.shape)
```

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

Shape of array:  (3, 3)
```

In [3]:
```
a11, a12, a13, a21, a22, a23, a31, a32, a33 = np.arange(1, 10, 1, dtype=np.int32)
arr = np.array([a11, a12, a13, a21, a22, a23, a31, a32, a33]).reshape(3, 3)
print(arr)
print('\nShape of array: ', arr.shape)
```

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

Shape of array:  (3, 3)
```

In [4]:
```
a11, a12, a13, a21, a22, a23, a31, a32, a33 = np.arange(1, 10, 1, dtype=np.float64)
arr = np.array([a11, a12, a13, a21, a22, a23, a31, a32, a33]).reshape(3, 3)
print(arr)
print('\nShape of array: ', arr.shape)
```

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

Shape of array:  (3, 3)
```

In [5]:
```
a11, a12, a13, a21, a22, a23, a31, a32, a33 = np.arange(1, 10, 1, dtype=np.complex128)
arr = np.array([a11, a12, a13, a21, a22, a23, a31, a32, a33]).reshape(3, 3)
print(arr)
print('\nShape of array: ', arr.shape)
```

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

Shape of array:  (3, 3)
```

In [6]:
```
a11, a12, a13, a21, a22, a23, a31, a32, a33 = np.arange(1, 10, 1)
arr = np.array([[a11, a12, a13], [a21, a22, a23], [a31, a32, a33]], ndmin=3)
print(arr)
print('\nShape of the array ', np.shape(arr))
```

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

Shape of the array  (1, 3, 3)
```

In [7]:
```
a11, a12, a13, a21, a22, a23, a31, a32, a33 = np.arange(1, 10, 1)
arr = np.array([[a11, a12, a13], [a21, a22, a23], [a31, a32, a33]], ndmin=4)
print(arr)
print('\nShape of the array: ', np.shape(arr))
```

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

Shape of the array:  (1, 1, 3, 3)
```

2. [10 points] Create two array, one where all the elements are zeros and the other all ones, of size 10 × 5. Multiply the two arrays in all possible ways.

In [8]:
```
arr_zeros = np.zeros([10,5])
print(arr_zeros)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

In [9]:
```
arr_ones = np.ones([10,5])
print(arr_ones)
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

In [10]:
```
arr_zeros*arr_ones
```

Out[10]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [11]:
```
np.multiply(arr_zeros, arr_ones)
```

Out[11]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
```

```
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [12]:
```
np.dot(arr_zeros, arr_ones.T)
```

Out[12]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

In [13]:
```
arr_zeros @ arr_ones.T
```

Out[13]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

3. [10 points] Create an identity array of size 5 and take the element-by-element exponentiation of the array.

As we are looking at the exponentiation of the idenetity matrix, any power of the elements of the array will just be the identity matrix (assuming we are taking a positive exponent). Here we look at the square of the identity.

In [14]:
```
iden_5 = np.identity(5)
np.power(iden_5,2)
```

Out[14]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

4. [15 points] How can you replace ones and zeros with tile? What about replacing eye with diag and ones? Explain with an example.

The np.ones() & np.zeros() just output an array of 1's/0's with the given shape. We can therefore replicate the results of each function using the tile function, which simply repeats a given value/set of values.

The np.eye() funtion creates an identity of order n, ie. an n x n matrix with ones on the diagonal & zeros in the non-diagonal. We can therefore replicate its output using the diag() & ones() function by first creating an array of n ones, before diagonalizing the array using diag(), to create an n x n identity matrix.

In [15]:
```
def ones_tile(shape):
    ones = np.tile(1, shape)
```

```
    return ones

ones_tile(5)
```

Out[15]:
```
array([1, 1, 1, 1, 1])
```

In [16]:
```
def zeros_tile(shape):
    zeros = np.tile(0, shape)
    return zeros

zeros_tile(5)
```

Out[16]:
```
array([0, 0, 0, 0, 0])
```

In [17]:
```
def identity_diag(n):
    identity = np.diag(np.ones(n))
    return identity

identity_diag(5)
```

Out[17]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

5. [15 points] Let y = arange(24.0). Use both shape and reshape to produce all possible arrays from y. Return y to it original size.

In [18]:
```
y = np.arange(24.0)
original_shape = y.shape

for i in range(1,len(y)+1):
    if len(y) % i == 0:
        with np.printoptions(threshold=np.inf):
            y.shape = (i, len(y)//i)
            print(y, '\n')
            y.shape = original_shape
```

```
[[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23.]]

[[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11.]
 [12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23.]]

[[ 0.  1.  2.  3.  4.  5.  6.  7.]
 [ 8.  9. 10. 11. 12. 13. 14. 15.]
 [16. 17. 18. 19. 20. 21. 22. 23.]]

[[ 0.  1.  2.  3.  4.  5.]
 [ 6.  7.  8.  9. 10. 11.]
 [12. 13. 14. 15. 16. 17.]
 [18. 19. 20. 21. 22. 23.]]

[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]
 [12. 13. 14. 15.]
 [16. 17. 18. 19.]
 [20. 21. 22. 23.]]

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

```
 [ 9. 10. 11.]
 [12. 13. 14.]
 [15. 16. 17.]
 [18. 19. 20.]
 [21. 22. 23.]]

[[ 0.  1.]
 [ 2.  3.]
 [ 4.  5.]
 [ 6.  7.]
 [ 8.  9.]
 [10. 11.]
 [12. 13.]
 [14. 15.]
 [16. 17.]
 [18. 19.]
 [20. 21.]
 [22. 23.]]

[[ 0.]
 [ 1.]
 [ 2.]
 [ 3.]
 [ 4.]
 [ 5.]
 [ 6.]
 [ 7.]
 [ 8.]
 [ 9.]
 [10.]
 [11.]
 [12.]
 [13.]
 [14.]
 [15.]
 [16.]
 [17.]
 [18.]
 [19.]
 [20.]
 [21.]
 [22.]
 [23.]]
```

In [19]:

```
y = np.arange(24.0)
for i in range(1,len(y)+1):
    if len(y) % i == 0:
        with np.printoptions(threshold=np.inf):
            print(y.reshape(i, len(y)//i), '\n')

y.shape = original_shape
```

```
[[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23.]]

[[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11.]
 [12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23.]]

[[ 0.  1.  2.  3.  4.  5.  6.  7.]
 [ 8.  9. 10. 11. 12. 13. 14. 15.]
 [16. 17. 18. 19. 20. 21. 22. 23.]]

[[ 0.  1.  2.  3.  4.  5.]
 [ 6.  7.  8.  9. 10. 11.]
 [12. 13. 14. 15. 16. 17.]
```

```
 [18. 19. 20. 21. 22. 23.]]

[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]
 [12. 13. 14. 15.]
 [16. 17. 18. 19.]
 [20. 21. 22. 23.]]

[[ 0.  1.  2.]
 [ 3.  4.  5.]
 [ 6.  7.  8.]
 [ 9. 10. 11.]
 [12. 13. 14.]
 [15. 16. 17.]
 [18. 19. 20.]
 [21. 22. 23.]]

[[ 0.  1.]
 [ 2.  3.]
 [ 4.  5.]
 [ 6.  7.]
 [ 8.  9.]
 [10. 11.]
 [12. 13.]
 [14. 15.]
 [16. 17.]
 [18. 19.]
 [20. 21.]
 [22. 23.]]

[[ 0.]
 [ 1.]
 [ 2.]
 [ 3.]
 [ 4.]
 [ 5.]
 [ 6.]
 [ 7.]
 [ 8.]
 [ 9.]
 [10.]
 [11.]
 [12.]
 [13.]
 [14.]
 [15.]
 [16.]
 [17.]
 [18.]
 [19.]
 [20.]
 [21.]
 [22.]
 [23.]]
```

6. Construct an array from the following covariance matrix of returns of a 10-asset portfolio:

$$X = \begin{bmatrix} 0.146 & 0.161 & 0.136 & 0.115 & 0.121 & 0.119 & 0.126 & 0.114 & 0.082 & 0.147 \\ 0.161 & 0.576 & 0.296 & 0.243 & 0.318 & 0.235 & 0.246 & 0.151 & 0.094 & 0.308 \\ 0.136 & 0.296 & 0.234 & 0.198 & 0.233 & 0.168 & 0.176 & 0.132 & 0.085 & 0.222 \\ 0.115 & 0.243 & 0.198 & 0.391 & 0.173 & 0.148 & 0.127 & 0.108 & 0.125 & 0.189 \\ 0.121 & 0.318 & 0.233 & 0.173 & 0.467 & 0.233 & 0.217 & 0.156 & 0.054 & 0.234 \\ 0.119 & 0.235 & 0.168 & 0.148 & 0.233 & 0.259 & 0.157 & 0.123 & 0.081 & 0.188 \\ 0.126 & 0.246 & 0.176 & 0.127 & 0.217 & 0.157 & 0.214 & 0.121 & 0.058 & 0.192 \\ 0.114 & 0.151 & 0.132 & 0.108 & 0.156 & 0.123 & 0.121 & 0.195 & 0.070 & 0.150 \\ 0.082 & 0.094 & 0.085 & 0.125 & 0.054 & 0.081 & 0.058 & 0.070 & 0.157 & 0.091 \\ 0.147 & 0.308 & 0.222 & 0.189 & 0.234 & 0.188 & 0.192 & 0.150 & 0.091 & 0.273 \end{bmatrix}$$

(a) [20 points] Perform the following operations on the array using NumPy built-in functions

(i) what is the minimum and maximum values in X.

(ii) what is the mean, median, variance, and standard deviation of X.

(iii) repeat the questions in (i) and (ii) above along the axis of the array (that is both vertically and horizontally).

In [20]:
```
X = np.array([[0.146, 0.161, 0.136, 0.115, 0.121, 0.119, 0.126, 0.114, 0.082, 0.147]
    ,[0.161, 0.576, 0.296, 0.243, 0.318, 0.235, 0.246, 0.151, 0.094, 0.308]
    ,[0.136, 0.296, 0.234, 0.198, 0.233, 0.168, 0.176, 0.132, 0.085, 0.222]
    ,[0.115, 0.243, 0.198, 0.391, 0.173, 0.148, 0.127, 0.108, 0.125, 0.189]
    ,[0.121, 0.318, 0.233, 0.173, 0.467, 0.233, 0.217, 0.156, 0.054, 0.234]
    ,[0.119, 0.235, 0.168, 0.148, 0.233, 0.259, 0.157, 0.123, 0.081, 0.188]
    ,[0.126, 0.246, 0.176, 0.127, 0.217, 0.157, 0.214, 0.121, 0.058, 0.192]
    ,[0.114, 0.151, 0.132, 0.108, 0.156, 0.123, 0.121, 0.195, 0.070, 0.150]
    ,[0.082, 0.094, 0.085, 0.125, 0.054, 0.081, 0.058, 0.070, 0.157, 0.091]
    ,[0.147, 0.308, 0.222, 0.189, 0.234, 0.188, 0.192, 0.150, 0.091, 0.273]])

print('i)\tThe minimum of X is: ', np.min(X),
      '\n\tThe maximum of X is: ', np.max(X))

print('\nii)\tThe mean of X is: %.4f' % np.mean(X),
      '\n\tThe median of X is: ', np.median(X),
      '\n\tThe variance of X is: %.4f' % np.var(X),
      '\n\tThe standard deviation of X is: %.4f' % np.std(X))

print('\niii)\tThe minimums of each row of X are:\n\t', np.min(X, axis=0),
      '\n\n\tThe maximums of each row of X are:\n\t', np.max(X, axis=0),
      '\n\n\tThe minimums of each column of X are:\n\t', np.min(X, axis=1),
      '\n\n\tThe maximums of each column of X are:\n\t', np.max(X, axis=1))


print('\n\tThe mean of each row of X is:\n\t', np.mean(X,axis=0),
      '\n\n\tThe median of each row of X is:\n\t', np.median(X,axis=0),
      '\n\n\tThe variance of each row of X is:\n\t', np.round(np.var(X, axis=0),4),
      '\n\n\tThe standard deviation of each row of X is:\n\t', np.round(np.std(X, axis=0

      '\n\n\tThe mean of each column of X is:\n\t', np.mean(X,axis=1),
      '\n\n\tThe median of each column of X is:\n\t', np.median(X,axis=1),
      '\n\n\tThe variance of each column of X is:\n\t', np.round(np.var(X, axis=1),4),
      '\n\n\tThe standard deviation of each column of X is:\n\t', np.round(np.std(X, axi
```

```
i)      The minimum of X is:  0.054
        The maximum of X is:  0.576

ii)     The mean of X is: 0.1735
        The median of X is:  0.156
```

```
The variance of X is: 0.0071
The standard deviation of X is: 0.0843
```

iii)
```
The minimums of each row of X are:
 [0.082 0.094 0.085 0.108 0.054 0.081 0.058 0.07  0.054 0.091]

The maximums of each row of X are:
 [0.161 0.576 0.296 0.391 0.467 0.259 0.246 0.195 0.157 0.308]

The minimums of each column of X are:
 [0.082 0.094 0.085 0.108 0.054 0.081 0.058 0.07  0.054 0.091]

The maximums of each column of X are:
 [0.161 0.576 0.296 0.391 0.467 0.259 0.246 0.195 0.157 0.308]

The mean of each row of X is:
 [0.1267 0.2628 0.188  0.1817 0.2206 0.1711 0.1634 0.132  0.0897 0.1994]

The median of each row of X is:
 [0.1235 0.2445 0.187  0.1605 0.225  0.1625 0.1665 0.1275 0.0835 0.1905]

The variance of each row of X is:
 [0.0004 0.0157 0.0034 0.0065 0.0115 0.003  0.0029 0.001  0.0009 0.0036]

The standard deviation of each row of X is:
 [0.0211 0.1253 0.0584 0.0807 0.1072 0.0546 0.0538 0.0319 0.0294 0.0601]

The mean of each column of X is:
 [0.1267 0.2628 0.188  0.1817 0.2206 0.1711 0.1634 0.132  0.0897 0.1994]

The median of each column of X is:
 [0.1235 0.2445 0.187  0.1605 0.225  0.1625 0.1665 0.1275 0.0835 0.1905]

The variance of each column of X is:
 [0.0004 0.0157 0.0034 0.0065 0.0115 0.003  0.0029 0.001  0.0009 0.0036]

The standard deviation of each column of X is:
 [0.0211 0.1253 0.0584 0.0807 0.1072 0.0546 0.0538 0.0319 0.0294 0.0601]
```

(b) [10 points] Verify the following:

(i) the sum of the eigenvalues of X is the same as the trace.

(ii) the product of the eigenvalues of X is the determinant

In [21]:
```
print('i)\tTrace of X: %.4f' % np.trace(X))
print('\tSum of eigenvalues of X: %.4f' % sum(np.linalg.eig(X)[0]))

print('\nii)\tDeterminant of X: ', np.linalg.det(X))
print('\tProduct of eigenvalues of X: ', np.product(np.linalg.eig(X)[0]))
```

```
i)      Trace of X: 2.9120
        Sum of eigenvalues of X: 2.9120

ii)     Determinant of X:  4.0490414474945064e-10
        Product of eigenvalues of X:  4.0490414474945183e-10
```

(c) [10 points] Verify the following:

(i) the inverse of X is equal to V D−1V ′, where V is the array containing the eignevectors and D is a diagonal array containing eigenvectors

In [22]:
```
X_inv = np.linalg.inv(X)
#print(X_inv)
```

In [23]:
```
D = np.diag(np.linalg.eig(X)[0])
D_inv = np.linalg.inv(D)
V = np.linalg.eig(X)[1]

X_decomp = V @ D_inv @ V.T
#print(X_decomp)
```

In [24]:
```
X_decomp.all() == X_inv.all()
```

Out[24]:
```
True
```

We see that each element of the decomposed array $VD^{-1}V'$ is equal to that of X^{-1}