Q1. CREATING NUMPY 1-D ARRAY WHICH WE CALL AS VECTOR

In [27]:

```
import numpy as np

#create a vector as a row
vector_row = np.array([1,2,3])

#create vector as a column
vector_column = np.array([[1],[2],[3]])
print(vector_row)
print(vector_column)
```

```
[1 2 3]
[[1]
 [2]
 [3]]
```

Q2 CREATING A MATRIX- 2D ARRAY, CONTAINING 2 ROWS AND 3 COLUMNS

In [ ]:

```
import numpy as np

#create a matrix
matrix = np.array([[1,2,3],[4,5,6]])

print(matrix)
```

Q3. CREATING A SPARSE MATRIX

Sparse Matrices store only non zero elements and assume all other values will be zero, leading to significant computational savings.

In [34]:

```python
# Python program to create
# sparse matrix using csr_matrix()

# Import required package
import numpy as np
from scipy.sparse import csr_matrix

# Creating a 3 * 4 sparse matrix
sparseMatrix = csr_matrix((3, 4),
                          dtype = np.int8).toarray()

# Print the sparse matrix
print(sparseMatrix)
```

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

In [ ]:

```python
#example 2 of sparse matrix
# Python program to create
# sparse matrix using csr_matrix()

# Import required package
import numpy as np
from scipy.sparse import csr_matrix

row = np.array([0, 0, 1, 1, 2, 1])
col = np.array([0, 1, 2, 0, 2, 2])

# taking data
data = np.array([1, 4, 5, 8, 9, 6])

# creating sparse matrix
sparseMatrix = csr_matrix((data, (row, col)),
                          shape = (3, 3)).toarray()

# print the sparse matrix
print(sparseMatrix)
```

Python’s SciPy gives tools for creating sparse matrices using multiple data structures, as well as tools for converting a dense matrix to a sparse matrix. The function csr_matrix() is used to create a sparse matrix of compressed sparse row format whereas csc_matrix() is used to create a sparse matrix of compressed sparse column format.

SciPy in Python is an open-source library used for solving mathematical, scientific, engineering, and technical problems. It allows users to manipulate the data and visualize the data using a wide range of high-level Python commands.

Q4. SELECTING ELEMENTS- WHEN NEED TO SELECT MORE THAN ONE ELEMENT

In [44]:

```
import numpy as np

#creating vector as row
#vector_row = np.array([1,2,3,4,5,6])

#create a matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)

#select 3rd element of vector
print(vector_row[2])

#select 2nd row 2nd column
print(matrix[1,1])

#Select all elements of a vector
print(vector_row[:])

#Select the everything after the 3rd element
print(vector_row[3:])

#Select the last element
print(vector_row[-1])

#Select the first 2 rows and all the columns of the matrix
print(matrix[:2,:])

#Select all rows and the 2nd column of the matrix
print(matrix[:,1:2])
```

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

Q5. DESCRIBING A MATRIX

In [46]:

```
import numpy as np


#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])

#printing matrix
print(matrix)

#View the Number of Rows and Columns
print(matrix.shape)

#View the number of elements (rows*columns)
print(matrix.size)

#View the number of Dimensions(2 in this case)
print(matrix.ndim)
```

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

Q6. APPLYING OPERTIONS TO ELEMENTS

In [49]:

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)

#Create a function that adds 100 to something
add_100 =lambda i: i+100

#Convert it into a vectorized function
vectorized_add_100= np.vectorize(add_100)

#Apply function to all elements in matrix
print(vectorized_add_100(matrix))
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[101 102 103]
 [104 105 106]
 [107 108 109]]
```

Q7. Finding the max and min values

In [51]:

```python
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)

#Return the max element
print(np.max(matrix))

#Return the min element
print(np.min(matrix))

#To find the max element in each column
print(np.min(matrix,axis=0))

#To find the max element in each row
print(np.max(matrix,axis=1))
```

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

Q8. Calculating Average, Variance and Standard deviation

In [53]:

```python
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)

#Mean
print(np.mean(matrix))

#Standard Dev.
print(np.std(matrix))

#Variance
print(np.var(matrix))
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
5.0
2.581988897471611
6.666666666666667
```

Q9. Reshaping Arrays

```
When you want to reshape an array(changing the number of rows and columns)
without changing the elements.
```

In [55]:

```python
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)

#Reshape
print(matrix.reshape(9,1))

#Here -1 says as many columns as needed and 1 row
print(matrix.reshape(1,-1))

#If we provide only 1 value Reshape would return a 1-d array of that length
print(matrix.reshape(9))

#We can also use the Flatten method to convert a matrix to 1-d array
print(matrix.flatten())
```

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

Q10. Transposing a vector or a Matrix

In [56]:

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])

print(matrix)
#Transpose the matrix
print(matrix.T)
```

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

Q11. Finding the Determinant and Rank of a Matrix

In [57]:

```
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)

#Calculate the Determinant
print(np.linalg.det(matrix))

#Calculate the Rank
print(np.linalg.matrix_rank(matrix))
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
-9.51619735392994e-16
2
```

Q12. Getting the Diagonal of a Matrix

In [59]:

```
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)
#Print the Principal diagonal
print(matrix.diagonal())

#Print the diagonal one above the Principal diagonal
print(matrix.diagonal(offset=1))

#Print the diagonal one below Principal diagonal
print(matrix.diagonal(offset=-1))
```

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

Q13. Calculating the trace of a Matrix ¶

In [60]:

```
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)

#Print the Trace
print(matrix.trace())
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
15
```

```
to find the trace of the Matrix
```