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

In [3]:
```
# Task 1.
# Create a 3x3 matrix with values ranging from 0 to 8.
matrix = np.arange(9).reshape((3, 3))
print(matrix)
```

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

In [5]:
```
# Task 2.
# Create a 3x3 matrix where the diagonal elements are [1, 2, 3] and the off-diagonal e
off_diagonal_zero = np.zeros((3,3))
for i in range(len(off_diagonal_zero)):
    off_diagonal_zero[i,i] = i+1
print(off_diagonal_zero)
```

```
[[1. 0. 0.]
 [0. 2. 0.]
 [0. 0. 3.]]
```

In [7]:
```
#Task 3
#calculate mean , median , s.d
array = np.array([1,2,3,4,5])
print("Mean : ",np.mean(array))
print("Median : ",np.median(array))
print("Standard Deviation : ",np.std(array))
```

```
Mean :  3.0
Median :  3.0
Standard Deviation :  1.4142135623730951
```

In [11]:
```
#Task 4
a = np.array([1,2,3])
b = np.array([4,5,6])
result = np.concatenate((a,b))
print(result)
```

```
[1 2 3 4 5 6]
```

In [14]:
```
#task 5 dot product
A =np.array([[1, 2],
             [3, 4]])
B = np.array([[5, 6],
              [7, 8]])
result = np.dot(A,B)
print(result)
```

```
[[19 22]
 [43 50]]
```

In [15]:
```
#task 6 replace even by zeros
array = np.array([1,2,3,4,5])
array[array % 2 == 0] = 0
print(array)
```

```
[1 0 3 0 5]
```

In [16]:
```
# Task 7
# .Given a matrix [[1, 2, 3], [4, 5, 6], [7, 8, 9]], calculate the sum of each column
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
result = np.sum(matrix , axis = 0)
print(result)
```

```
[12 15 18]
```

In [22]:
```
# Task 8
# .Write a function that takes two arrays A and B, and returns the indices where
# the elements of A are greater than the corresponding elements of B
def returnGreaterIndexes(a,b):
    greaterIndexes = []
    for i in range(len(a)):
        if a[i] > b[i] :
            greaterIndexes.append(i)
    return np.array(greaterIndexes)
a = np.array([1,2,3,4,5])
b = np.array([0,2,1,3,9])
indexes = returnGreaterIndexes(a , b)
print(indexes)
```

```
[0 2 3]
```

In [25]:
```
# Task 9
# .Given a 2D array, calculate the sum of the diagonal elements that are divisible
# by 3.
x = np.arange(9).reshape(3,3)
summ = 0
for i in range(len(x)):
    if x[i,i] % 3 == 0:
        summ+=x[i,i]
print("Sum of diagonal elements that are divisible by 3 is : ",summ)
```

```
Sum of diagonal elements that are divisible by 3 is :  0
```

In [48]:
```
# Task 10.
# Create a function that takes a 2D array and returns the row with the maximum
# sum of its elements.
def returnMaxRow(x):
    rowsum = []
    m = 0
    rowsum = np.sum(x , axis = 1)
    m = np.max(rowsum)
    for i in range(len(rowsum)):
        if m == rowsum[i]:
            return x[i]
x = np.arange(5,9).reshape(2,2)
print(x)
maxrow = returnMaxRow(x)
print("Max row is : ",maxrow)
```

```
[[5 6]
 [7 8]]
Max row is :  [7 8]
```

In [38]:
```
# Task 11.
# Generate a random 5x5 matrix and replace all negative values with zeros, while
```

```
# keeping the positive values intact.
ran = np.random.randn(5,5)
ran[ran < 0] = 0
print(ran)
```

```
[[0.         0.64057847 0.         0.87705578 0.        ]
 [0.         0.09966257 0.         0.         0.        ]
 [2.50965986 0.         0.99616139 0.         0.        ]
 [0.         0.         0.86398172 0.         0.        ]
 [0.6663097  0.         0.         0.         0.0557859 ]]
```

In [51]:

```
# Task 12.
# Given a 1D array, calculate the moving average over a window of size 3. The
# mov-ing average at index i is the average of elements at indices i-1, i, and i+1.
x = np.array([1,2,3,4,5,6,7])
print(len(x))
averageofele = []
for i in range(len(x)):
    if i == 0 :
        averageofele.append((x[i] + x[i+1])/2)
    elif i == len(x) - 1:
        averageofele.append((x[i] + x[i - 1])/2)
    else:
        averageofele.append((x[i] + x[i+1] + x[i-1])/3)
result = averageofele
print(result)
```

```
7
[1.5, 2.0, 3.0, 4.0, 5.0, 6.0, 6.5]
```

In [2]:

```
# Task 13.
# Implement the Euclidean distance function that takes two arrays as inputs and
# re-turns the Euclidean distance between them.
array1 = np.array([1,2,3,4,5])
array2 = np.array([7,6,5,4,3])

temp = array1 - array2
distance = np.sqrt(np.sum(np.square(temp)))


print("Euclidean Distance: ", distance)
```

```
Euclidean Distance:  7.745966692414834
```

In [7]:

```
# Task 14:
# Create a function that takes an array and returns a new array with the
# elements normalized between
def arrayNormalizer(npArray):
    m = np.max(npArray)
    array = np.copy(npArray)
    return array/m

x = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
result = arrayNormalizer(x)
print(result)
```

```
[[0.11111111 0.22222222 0.33333333]
 [0.44444444 0.55555556 0.66666667]
 [0.77777778 0.88888889 1.        ]]
```

In [ ]:
```
#top
x[i-1 , j]
#right
x[i , j+1]
#left
x[i , j-1]
#bottom
x[i+1 , j]
#top left
x[i-1,j-1]
#top right
x[i-1 , j+1]
#bottom right
x[i+1,j+1]
#bottom left
x[i+1,j-1]
```

In [48]:
```
# Task 15:
# Implement a function that takes a 2D array and returns a new array where
# each element is the
# average of its neighboring elements (including diagonals). If a neighboring element
# out of bounds,
# consider it as zero
array = np.arange(1,26).reshape(5,5)
x = np.zeros((len(array)+2,len(array)+2))
x[1:-1 , 1:-1] = array
print(x)
print()
print()
results = []
for i in range(1,len(x)-1):
    for j in range(1,len(x)-1):
        count = 9
        count -=1 if x[i-1,j] == 0 else 0
        count -=1 if x[i,j+1] == 0  else 0
        count -=1 if x[i,j-1] == 0  else 0
        count -=1 if x[i+1,j] == 0 else 0
        count -=1 if x[i-1,j-1] == 0 else 0
        count -=1 if x[i-1,j+1] == 0 else 0
        count -=1 if x[i+1,j+1] == 0 else 0
        count -=1 if x[i+1,j-1] == 0  else 0
        results.append(np.round((x[i,j]+x[i-1,j]+x[i+1,j]+x[i,j+1]+x[i,j-1]+x[i-1,j-1]
results = np.array(results).reshape(5,5)
print(results)
```

```
[[ 0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  2.  3.  4.  5.  0.]
 [ 0.  6.  7.  8.  9. 10.  0.]
 [ 0. 11. 12. 13. 14. 15.  0.]
 [ 0. 16. 17. 18. 19. 20.  0.]
 [ 0. 21. 22. 23. 24. 25.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.]]
```

```
[[ 4.   4.5  5.5  6.5  7. ]
 [ 6.5  7.   8.   9.   9.5]
 [11.5 12.  13.  14.  14.5]
 [16.5 17.  18.  19.  19.5]
 [19.  19.5 20.5 21.5 22. ]]
```