importing the library

//SABIN CHAULAGAIN //2358554

[41]
```
import pandas as pd
import numpy as np
import time
```

1. Initialize an empty array with size 2X2

[42]
```
empty_array = np.empty((2, 2))
print(empty_array)
```

```
[[1. 0.]
 [0. 1.]]
```

2. Initialize an all one array with size 4X2

[43]
```
ones_array = np.ones((4, 2))
print(ones_array)
```

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

3. Return a new array of given shape and type, filled with fill value

[44]
```
filled_array = np.full((3, 3), 7)
print(filled_array)
```

```
[[7 7 7]
 [7 7 7]
 [7 7 7]]
```

4. Return a new array of zeros with same shape and type as a given array

[45]
```
sample_array = np.array([[1, 2, 3], [4, 5, 6]])
zeros_like_array = np.zeros_like(sample_array)
print(zeros_like_array)
```

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

5. Return a new array of ones with same shape and type as a given array

[46]
```
ones_like_array = np.ones_like(sample_array)
print(ones_like_array)
```

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

6. Convert an existing list to a NumPy array

[47]
```
new_list = [1, 2, 3, 4]
numpy_array = np.array(new_list)
print(numpy_array)
```

```
[1 2 3 4]
```

Problem 2

Create an array with values ranging from 10 to 49. {Hint:np.arrange()}.

[48]
```
array_range = np.arange(10, 50)
print(array_range)
```

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]
```

Create a 3X3 matrix with values ranging from 0 to 8. {Hint:look for np.reshape()}

[49]
```
matrix_3x3 = np.arange(9).reshape(3, 3)
print(matrix_3x3)
```

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

Create a 3X3 matrix with values ranging from 0 to 8. {Hint:look for np.reshape()}

[49]
```
matrix_3x3 = np.arange(9).reshape(3, 3)
print(matrix_3x3)
```

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

Create a 3X3 identity matrix.{Hint:np.eye()}

[50]
```
identity_matrix = np.eye(3)
print(identity_matrix)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Create a random array of size 30 and find the mean of the array. {Hint:check for np.random.random() and array.mean() function}

[51]
```
random_array = np.random.random(30)
mean_value = random_array.mean()
print(mean_value)
```

```
0.43103519239177623
```

Create a 10X10 array with random values and find the minimum and maximum values.

[52]
```
random_matrix = np.random.random((10, 10))
min_value = random_matrix.min()
max_value = random_matrix.max()
print("Minimum value:", min_value)
print("Maximum value:", max_value)
```

```
[52] random_matrix = np.random.random((10, 10))
     min_value = random_matrix.min()
     max_value = random_matrix.max()
     print("Minimum value:", min_value)
     print("Maximum value:", max_value)
```

```
Minimum value: 0.0012446062076497677
Maximum value: 0.9922441312969429
```

Create a zero array of size 10 and replace 5th element with 1.

```
[53] zero_array = np.zeros(10)
     zero_array[4] = 1
     print(zero_array)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

Reverse an array arr = [1,2,0,0,4,0].

```
[54] arr = np.array([1, 2, 0, 0, 4, 0])
     reversed_arr = arr[::-1]
     print(reversed_arr)
```

```
[0 4 0 0 2 1]
```

Create a 2d array with 1 on border and 0 inside.

```
[15] border_array = np.ones((5, 5))
     border_array[1:-1, 1:-1] = 0
     print(border_array)
```

```
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
```

Create a 8X8 matrix and fill it with a checkerboard pattern.

[55]
```
checkerboard = np.zeros((8, 8), dtype=int)
checkerboard[1::2, ::2] = 1
checkerboard[::2, 1::2] = 1
print(checkerboard)
```

```
[[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]]
```

Problem - 3: Array Operations: For the following arrays: x = np.array([[1,2],[3,5]]) and y = np.array([[5,6],[7,8]]); v = np.array([9,10]) and w = np.array([11,12]); Complete all the task using numpy:

[56]
```
x = np.array([[1, 2], [3, 5]])
y = np.array([[5, 6], [7, 8]])
v = np.array([9, 10])
w = np.array([11, 12])
```

Add the two array.

[57]
```
addition = x + y
print(addition)
```

```
[[ 6  8]
 [10 13]]
```

Subtract the two array.

```
[58] subtraction = x-y
     print(subtraction)
```

```
[[-4 -4]
 [-4 -3]]
```

Multiply the array with any integers of your choice.

```
[59] multiplication = x * 2
     print(multiplication)
```

```
[[ 2  4]
 [ 6 10]]
```

4. Find the square of each element of the array.

```
[60] square_x = np.square(x)
     print(square_x)
```

```
[[ 1  4]
 [ 9 25]]
```

5. Find the dot product between: v(and)w ; x(and)v ; x(and)y.

```
[61] dot_vw = np.dot(v, w)
     dot_xv = np.dot(x, v)
     dot_xy = np.dot(x, y)
     print("Dot product of v and w:", dot_vw)
     print("Dot product of x and v:", dot_xv)
     print("Dot product of x and y:", dot_xy)
```

```
Dot product of v and w: 219
```

```
Dot product of v and w: 219
Dot product of x and v: [29 77]
Dot product of x and y: [[19 22]
 [50 58]]
```

Concatenate x(and)y along row and Concatenate v(and)w along column. {Hint:try np.concatenate() or np.vstack() functions.

```
[62] concat_xy_row = np.concatenate((x, y), axis=0)
     concat_vw_col = np.vstack((v, w))
     print("Concatenated x and y along row:")
     print(concat_xy_row)
     print("Concatenated v and w along column:")
     print(concat_vw_col)
```

```
Concatenated x and y along row:
[[1 2]
 [3 5]
 [5 6]
 [7 8]]
Concatenated v and w along column:
[[ 9 10]
 [11 12]]
```

Concatenate x(and)v; if you get an error, observe and explain why did you get the error?

```
[63] try:
         concat_xv = np.concatenate((x, v), axis=0)
     except ValueError as e:
         concat_xv = str(e)
         print("Error:", concat_xv)
```

```
Error: all the input arrays must have same number of dimensions, but the array at index 0 has 2 dimension(s) and the array at index 1 has 1 dimension(s)
```

Explanation of the error:

The error occurs because `x` is a 2x2 matrix, and `v` is a 1D array with shape (2,).

In order to concatenate them, `v` must be reshaped to a 2D array, e.g., `v.reshape(1, -1)`.

+ Code + Text

Problem - 4: Matrix Operations: • For the following arrays: A = np.array([[3,4],[7,8]]) and B = np.array([[5,3],[2,1]]); Prove following with Numpy:

```
[64] A = np.array([[3, 4], [7, 8]])
     B = np.array([[5, 3], [2, 1]])
```

1. Prove A.A−1 = I.

```
[65] A_inv = np.linalg.inv(A)
     identity_matrix = np.dot(A, A_inv)

     identity_matrix = np.round(identity_matrix, decimals=5)
     print(identity_matrix)
```

```
[[1. 0.]
 [0. 1.]]
```

Prove AB /= BA.

```
[66] AB = np.dot(A, B)
     BA = np.dot(B, A)
     are_not_equal = not np.array_equal(AB, BA)
     print(are_not_equal)
```

```
True
```

Prove (AB)

T = BTAT .

[67]
```
AB_T = np.transpose(AB)
BT_AT = np.dot(np.transpose(B), np.transpose(A))
proof_transpose = np.array_equal(AB_T, BT_AT)
print(proof_transpose)
```

```
True
```

Solve the following system of Linear equation using Inverse Methods.

2x − 3y + z = −1 x − y + 2z = −3 3x + y − z = 9

[68]
```
A_matrix = np.array([[2, -3, 1], [1, -1, 2], [3, 1, -1]])
B_matrix = np.array([-1, -3, 9])

A_inv_matrix = np.linalg.inv(A_matrix)
X_solution = np.dot(A_inv_matrix, B_matrix)

X_solution_direct = np.linalg.solve(A_matrix, B_matrix)

print("A * A^(-1) = Identity Matrix:\n", identity_matrix)
print("AB ≠ BA:", are_not_equal)
print("(AB)^T = B^T * A^T:", proof_transpose)
print("Solution using Inverse Method:", X_solution)
print("Solution using np.linalg.solve:", X_solution_direct)
```

```
A * A^(-1) = Identity Matrix:
 [[1. 0.]
 [0. 1.]]
AB ≠ BA: True
(AB)^T = B^T * A^T: True
Solution using Inverse Method: [ 2.  1. -2.]
Solution using np.linalg.solve: [ 2.  1. -2.]
```

```
[0. 1.]]
AB ≠ BA: True
(AB)^T = B^T * A^T: True
Solution using Inverse Method: [ 2.  1. -2.]
Solution using np.linalg.solve: [ 2.  1. -2.]
```

10.2 Experiment: How Fast is Numpy? In this exercise, you will compare the performance and implementation of operations using plain Python lists (arrays) and NumPy arrays. Follow the instructions:

1. Element-wise Addition: • Using Python Lists, perform element-wise addition of two lists of size 1, 000, 000. Measure and Print the time taken for this operation.

```python
size = 1_000_000
matrix_size = 1000

list1 = [i for i in range(size)]
list2 = [i for i in range(size)]

array1 = np.arange(size)
array2 = np.arange(size)
```

```python
# Python lists
start = time.time()
result_list = [list1[i] + list2[i] for i in range(size)]
end = time.time()
print(f"Python list addition time: {end - start:.5f} seconds")
```

```
Python list addition time: 0.08286 seconds
```

Using Numpy Arrays, Repeat the calculation and measure and print the time taken for this operation.

[71]

```
# NumPy arrays
start = time.time()
result_array = array1 + array2
end = time.time()
print(f"NumPy addition time: {end - start:.5f} seconds")
```

```
NumPy addition time: 0.00832 seconds
```

Element-wise Multiplication • Using Python Lists, perform element-wise multiplication of two lists of size 1, 000, 000. Measure and Print the time taken for this operation.

[72]

```
start = time.time()
result_list = [list1[i] * list2[i] for i in range(size)]
end = time.time()
print(f"Python list multiplication time: {end - start:.5f} seconds")
```

```
Python list multiplication time: 0.10009 seconds
```

Using Numpy Arrays, Repeat the calculation and measure and print the time taken for this operation.

[73]

```
start = time.time()
result_array = array1 * array2
end = time.time()
print(f"NumPy multiplication time: {end - start:.5f} seconds")
```

```
NumPy multiplication time: 0.00345 seconds
```

Dot Product • Using Python Lists, compute the dot product of two lists of size 1, 000, 000. Measure and Print the time taken for this operation.

[74]

```
start = time.time()
dot_product = sum(list1[i] * list2[i] for i in range(size))
end = time.time()
print(f"Python list dot product time: {end - start:.5f} seconds")
```

```
Python list dot product time: 0.17959 seconds
```

Using Numpy Arrays, Repeat the calculation and measure and print the time taken for this operation.

[75]

```
start = time.time()
dot_product_np = np.dot(array1, array2)
end = time.time()
print(f"NumPy dot product time: {end - start:.5f} seconds")
```

```
NumPy dot product time: 0.00299 seconds
```

Matrix Multiplication

• Using Python lists, perform matrix multiplication of two matrices of size 1000x1000. Mea- sure and print the time taken for this operation.

[76]

```
matrix1 = [[i for i in range(matrix_size)] for _ in range(matrix_size)]
matrix2 = [[i for i in range(matrix_size)] for _ in range(matrix_size)]

matrix1_np = np.arange(matrix_size**2).reshape(matrix_size, matrix_size)
matrix2_np = np.arange(matrix_size**2).reshape(matrix_size, matrix_size)
```

[77]

```
start = time.time()
result_matrix = [[sum(matrix1[i][k] * matrix2[k][j] for k in range(matrix_size)) for j in range(matrix_size)] for i in range(matrix_size)]
end = time.time()
print(f"Python list matrix multiplication time: {end - start:.5f} seconds")
```

```
Python list matrix multiplication time: 178.32781 seconds
```

Double-click (or enter) to edit

[76]
```
matrix1 = [[i for i in range(matrix_size)] for _ in range(matrix_size)]
matrix2 = [[i for i in range(matrix_size)] for _ in range(matrix_size)]

matrix1_np = np.arange(matrix_size**2).reshape(matrix_size, matrix_size)
matrix2_np = np.arange(matrix_size**2).reshape(matrix_size, matrix_size)
```

[77]
```
start = time.time()
result_matrix = [[sum(matrix1[i][k] * matrix2[k][j] for k in range(matrix_size)) for j in range(matrix_size)] for i in range(matrix_size)]
end = time.time()
print(f"Python list matrix multiplication time: {end - start:.5f} seconds")
```

```
Python list matrix multiplication time: 178.32781 seconds
```

Double-click (or enter) to edit

Using NumPy arrays, perform matrix multiplication of two matrices of size 1000x1000. Measure and print the time taken for this operation.

[78]
```
start = time.time()
result_matrix_np = np.dot(matrix1_np, matrix2_np)
end = time.time()
print(f"NumPy matrix multiplication time: {end - start:.5f} seconds")
```

```
NumPy matrix multiplication time: 1.80148 seconds
```