Name: Ritesh Chaudhary

Id: 2438464

<u>Workshop week-1</u>

Week1.ipynb

File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

Name: Ritesh Chaudhary Id: 2438464

```python
import numpy as np
```

```python
#No:1 Initialize an empty array with size 2X2.

a = np.empty((2, 2))
a
```

```
array([[0., 0.],
       [0., 0.]])
```

```python
#No: 2 Initialize an all one array with size 4X2.

ones_array = np.ones((4, 2))
ones_array
```

```
array([[1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.]])
```

```python
#No 4. Return a new array of zeros with same shape and type as a given array.{Hint: np.zeros like}#
zeros_array=np.zeros((3,3))
zeros_array
```

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

```python
#No. 5 Return a new array of ones with same shape and type as a given array.{Hint: np.ones like}
ones_array=np.ones_like((3,3))
ones_array
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

```python
#No. 6 For an existing list new_list = [1,2,3,4] convert to an numpy array.{Hint: np.array()}
new_list = [1,2,3,4]
new_array = np.array(new_list)
new_array
```

```
array([1, 2, 3, 4])
```

Task: 2 problem solving 2

```
#No.1 Create an array with values ranging from 10 to 49. {Hint:np.arrange()}.
arr = np.arange(10, 50)
arr
```

```
array([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])
```

```
#No.2 Create a 3X3 matrix with values ranging from 0 to 8.{Hint:look for np.reshape()}
matrix = np.arange(9).reshape(3, 3)
matrix
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

```
#No. 3 Create a 3X3 identity matrix.{Hint:np.eye()}
identity_matrix = np.eye(3)
identity_matrix
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

+ Code + Text

```
#NO .4. Create a random array of size 30 and find the mean of the array. {Hint:check for np.random.random() and array.mean() function}
random_array = np.random.random(30)
print(random_array)
mean_value = random_array.mean()
mean_value
```

```
[0.49830873 0.09152256 0.53472669 0.36155279 0.68532767 0.31032319
 0.70984366 0.96421895 0.31574207 0.11640911 0.05396604 0.64411573
 0.53256612 0.73882569 0.88187579 0.2278008  0.66900332 0.4222528
 0.02872404 0.72612887 0.96773933 0.93913857 0.52534006 0.4779541
 0.60866074 0.33389884 0.87377824 0.47085217 0.85442892 0.03650504]
0.520051020943235
```

```
#5. Create a 10X10 array with random values and find the minimum and maximum values.
random_array = np.random.random((10, 10))
print(random_array)
min_value = random_array.min()
max_value = random_array.max()
print("\nMinimum value:", min_value)
print("Maximum value:", max_value)
```

```
[[0.55031291 0.28881376 0.3633234  0.08174648 0.88997395 0.77344532
  0.5365933  0.96520872 0.49683566 0.45871717]
 [0.20104284 0.9979559  0.35702968 0.41540405 0.90959901 0.97848664
  0.92549294 0.98689176 0.25206273 0.18662721]
 [0.14372567 0.95041716 0.8935291  0.54269667 0.7756559  0.89492662
  0.00304706 0.61758619 0.29527635 0.35331481]
 [0.65572446 0.21056426 0.22468741 0.14353208 0.70812509 0.42039814
  0.03057003 0.6750431  0.97396505 0.07612359]
 [0.18774958 0.795491   0.12012911 0.59676691 0.50406443 0.54898567
  0.82662213 0.68555515 0.74198969 0.11780576]
 [0.27328001 0.95836264 0.8810205  0.26973767 0.79415122 0.4815391
  0.85908483 0.8675595  0.83318338 0.26413476]
```

+ Code + Text

```
Minimum value: 0.0030470605989471045
Maximum value: 0.9979559011186023
```

```python
#6. Create a zero array of size 10 and replace 5th element with 1.
zero_array = np.zeros(10)
zero_array[4] = 1
zero_array
print("\n The replace element is: ",zero_array)
```

```
 The replace element is:  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

```python
#7. Reverse an array arr = [1,2,0,0,4,0].
arr_rev=[1,2,0,0,4,0]
arr_rev.reverse()
print("\n The reverse of an array is: ",arr_rev)
```

```
 The reverse of an array is:  [0, 4, 0, 0, 2, 1]
```

```python
#8. Create a 2d array with 1 on border and 0 inside.
border_array = np.ones((3, 3))
border_array[1:-1, 1:-1] = 0
border_array
```

```
array([[1., 1., 1.],
       [1., 0., 1.],
       [1., 1., 1.]])
```

+ Code + Text

```
[ ] #9. Create a 8X8 matrix and fill it with a checkerboard pattern.
    checkerboard_matrix = np.zeros((8, 8))
    checkerboard_matrix[1::2, ::2] = 1
    checkerboard_matrix[2::2, 1::2] = 1
    checkerboard_matrix
    print("\n The results is: ", checkerboard_matrix)
```

```
 The results is:  [[0. 0. 0. 0. 0. 0. 0. 0.]
 [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:

```
[ ] import numpy as np
```

```
[ ] #No: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:
    #1. Add the two array.

    x = np.array([[1, 2], [3, 5]])
    y = np.array([[5, 6], [7, 8]])
    result_add = x + y
    result_add
```

```
array([[ 6,  8],
       [10, 13]])
```

```
[ ] #2. Subtract the two array.
    result_sub = x - y
    result_sub
```

```
array([[-4, -4],
       [-4, -3]])
```

```python
#3. Multiply the array with any integers of your choice.
result_mul = x * 2
result_mul
```

```
array([[ 2,  4],
       [ 6, 10]])
```

```python
#4. Find the square of each element of the array.
result_square = x ** 2
result_square
```

```
array([[ 1,  4],
       [ 9, 25]])
```

```python
#5. Find the dot product between: v(and)w ; x(and)v ; x(and)y.
v = np.array([9, 10])
w = np.array([11, 12])
dot_product_vw = np.dot(v, w)
dot_product_vw
```

```
219
```

```python
#6. Concatenate x(and)y along row and Concatenate v(and)w along column. {Hint:try np.concatenate() or np.vstack() functions.
concatenated_rows = np.concatenate((x, y), axis=0)
concatenated_rows
```

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

```python
#7. Concatenate x(and)v; if you get an error, observe and explain why did you get the error?
concatenated_rows = np.concatenate((x, v), axis=0)
concatenated_rows
```

Show hidden output

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)

+ Code + Text

Problem - 4: Matrix Operations:

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

    import numpy as np

    # 1. Prove A * A^-1 = I
    A_inv = np.linalg.inv(A)
    identity_matrix = np.dot(A, A_inv)
    print("A * A^-1 = I:\n", identity_matrix)

    # 2. Prove AB = BA
    AB = np.dot(A, B)
    BA = np.dot(B, A)
    print("AB:\n", AB)
    print("BA:\n", BA)
    print("AB == BA:", np.array_equal(AB, BA))

    # 3. Prove (AB)^T = B^T A^T
    AB_T = np.transpose(AB)
    BT_AT = np.dot(np.transpose(B), np.transpose(A))
    print("(AB)^T:\n", AB_T)
    print("B^T A^T:\n", BT_AT)
    print("(AB)^T == B^T A^T:", np.array_equal(AB_T, BT_AT))
```

```
A * A^-1 = I:
 [[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]
```

```
[5] print("BA:\n", BA)
    print("AB == BA:", np.array_equal(AB, BA))

    # 3. Prove (AB)^T = B^T A^T
    AB_T = np.transpose(AB)
    BT_AT = np.dot(np.transpose(B), np.transpose(A))
    print("(AB)^T:\n", AB_T)
    print("B^T A^T:\n", BT_AT)
    print("(AB)^T == B^T A^T:", np.array_equal(AB_T, BT_AT))
```

```
A * A^-1 = I:
 [[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]
AB:
 [[23 13]
 [51 29]]
BA:
 [[36 44]
 [13 16]]
AB == BA: False
(AB)^T:
 [[23 51]
 [13 29]]
B^T A^T:
 [[23 51]
 [13 29]]
(AB)^T == B^T A^T: True
```

[6]

```python
import time

# Create two lists of size 1,000,000
list1 = [i for i in range(1000000)]
list2 = [i for i in range(1000000)]

# Measure the time taken for element-wise addition
start_time = time.time()
result_list = [list1[i] + list2[i] for i in range(1000000)]
end_time = time.time()

print("Time taken for element-wise addition using Python lists:", end_time - start_time, "seconds")
```

```
Time taken for element-wise addition using Python lists: 0.19272375106811523 seconds
```

[7]

```python
import numpy as np
import time

array1 = np.arange(1000000)
array2 = np.arange(1000000)

start_time = time.time()
result_array = array1 + array2
end_time = time.time()

print("Time taken for element-wise addition using NumPy arrays:", end_time - start_time, "seconds")
```

```
Time taken for element-wise addition using NumPy arrays: 0.007726430892944336 seconds
```

3m 11s completed at 8:42 PM

Time taken for element-wise addition using NumPy arrays: 0.007726430892944336 seconds

[8]

```python
import time

list1 = [i for i in range(1000000)]
list2 = [i for i in range(1000000)]

start_time = time.time()
result_list = [list1[i] * list2[i] for i in range(1000000)]
end_time = time.time()

print("Time taken for element-wise multiplication using Python lists:", end_time - start_time, "seconds")
```

```
Time taken for element-wise multiplication using Python lists: 0.2915968894958496 seconds
```

[9]

```python
import numpy as np
import time

array1 = np.arange(1000000)
array2 = np.arange(1000000)

start_time = time.time()
result_array = array1 * array2
end_time = time.time()

print("Time taken for element-wise multiplication using NumPy arrays:", end_time - start_time, "seconds")
```

```
Time taken for element-wise multiplication using NumPy arrays: 0.005314826965332031 seconds
```

3m 11s completed at 8:42 PM

[10]
```python
import time

list1 = [i for i in range(1000000)]
list2 = [i for i in range(1000000)]

start_time = time.time()
dot_product_list = sum(list1[i] * list2[i] for i in range(1000000))
end_time = time.time()

print("Dot product using Python lists:", dot_product_list)
print("Time taken using Python lists:", end_time - start_time, "seconds")
```

```
Dot product using Python lists: 333332833333500000
Time taken using Python lists: 0.30770206451416016 seconds
```

```python
import time

list1 = [i for i in range(1000000)]
list2 = [i for i in range(1000000)]

start_time = time.time()
dot_product_list = sum(list1[i] * list2[i] for i in range(1000000))
end_time = time.time()

print("Dot product using Python lists:", dot_product_list)
print("Time taken using Python lists:", end_time - start_time, "seconds")
```

```
Dot product using Python lists: 333332833333500000
Time taken using Python lists: 0.30916738510131836 seconds
```


```python
import time

list1 = [i for i in range(1000000)]
list2 = [i for i in range(1000000)]

start_time = time.time()
dot_product_list = sum(list1[i] * list2[i] for i in range(1000000))
end_time = time.time()

print("Dot product using Python lists:", dot_product_list)
print("Time taken using Python lists:", end_time - start_time, "seconds")
```

```
Dot product using Python lists: 333332833333500000
Time taken using Python lists: 0.30916738510131836 seconds
```


[16]
```python
import time

matrix1 = [[i for i in range(1000)] for j in range(1000)]
matrix2 = [[i for i in range(1000)] for j in range(1000)]

start_time = time.time()
result_matrix = [[sum(a * b for a, b in zip(matrix1_row, matrix2_col)) for matrix2_col in zip(*matrix2)] for matrix1_row in matrix1]
end_time = time.time()
print("Time taken for matrix multiplication using Python lists:", end_time - start_time, "seconds")
```

```
Time taken for matrix multiplication using Python lists: 191.85745310783386 seconds
```

[14]

```python
import numpy as np
import time

array1 = np.arange(1000000).reshape(1000, 1000)
array2 = np.arange(1000000).reshape(1000, 1000)

start_time = time.time()
result_array = np.dot(array1, array2)
end_time = time.time()

print("Time taken for matrix multiplication using NumPy arrays:", end_time - start_time, "seconds")
```

```
Time taken for matrix multiplication using NumPy arrays: 1.6247961521148682 seconds
```