Numpy Assignment

```
In [3]: import numpy as np
```

1) How to create a 3×3 Identity Matrix with Float Data Type?

```
In [4]: identity_matrix = np.identity(3, dtype=float)
        print(identity_matrix)
```

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

2) Create a 1D Array with Random Values between 0 and 1

```
In [5]: arr = np.random.rand(5)
        arr
```

```
Out[5]: array([0.13205215, 0.99347326, 0.58953362, 0.78736524, 0.0504747 ])
```

3) Create a 2D Array with Random Integer Values

```
In [6]: arr_2d = np.random.randint(1, 100, size=(3, 4))
        arr_2d
```

```
Out[6]: array([[45, 19, 12, 58],
               [11, 58, 47, 35],
               [72, 58, 57, 44]], dtype=int32)
```

4) Creating an Array Using a Custom Function

```
In [7]: def square(x):
            return x * x

        arr = np.fromfunction(square, (5,), dtype=int)
        arr
```

```
Out[7]: array([ 0,  1,  4,  9, 16])
```

5) Reshaping a 1D Array into a 2D Array

```
In [8]: arr = np.arange(12)
        reshaped_arr = arr.reshape(3, 4)
        reshaped_arr
```

```
Out[8]: array([[ 0,  1,  2,  3],
               [ 4,  5,  6,  7],
               [ 8,  9, 10, 11]])
```

6) How to Create a 3×3 Array of Ones?

```
In [9]: ones_array = np.ones((3, 3))
        ones_array
```

```
Out[9]: array([[1., 1., 1.],
               [1., 1., 1.],
               [1., 1., 1.]])
```

7) Find Common Items Between Two NumPy Arrays

```
In [10]: a = np.array([1,2,3,2,3,4,3,4,5,6])
         b = np.array([7,2,10,2,7,4,9,4,9,8])

         common_elements = np.intersect1d(a, b)
         common_elements

Out[10]: array([2, 4])
```

8) From array a remove all items present in array b

```
In [11]: a = np.array([1,2,3,4,5])
         b = np.array([5,6,7,8,9])

         result = np.setdiff1d(a, b)
         result

Out[11]: array([1, 2, 3, 4])
```

9) Limit the number of items printed in NumPy array

```
In [12]: np.set_printoptions(threshold=6)
         a = np.arange(15)
         a

Out[12]: array([ 0,  1,  2, ..., 12, 13, 14], shape=(15,))
```

10) Drop all NaN values from a 1D NumPy array

```
In [13]: arr = np.array([1,2,3,np.nan,5,6,7,np.nan])
         clean_arr = arr[~np.isnan(arr)]
         clean_arr

Out[13]: array([1., 2., 3., 5., 6., 7.])
```

11) Create 1D and 2D arrays with values 1 to 20

```
In [14]: arr_1d = np.arange(1, 21)
         arr_2d = arr_1d.reshape(4, 5)

         arr_1d, arr_2d

Out[14]: (array([ 1,  2,  3, ..., 18, 19, 20], shape=(20,)),
          array([[ 1,  2,  3,  4,  5],
                 [ 6,  7,  8,  9, 10],
                 [11, 12, 13, 14, 15],
                 [16, 17, 18, 19, 20]], shape=(4, 5)))
```

12) Properties of a 3D Array and Change Data Type

```
In [15]: arr = np.random.randint(1, 10, size=(2,3,4))

         arr.shape, arr.size, arr.ndim, arr.dtype

Out[15]: ((2, 3, 4), 24, 3, dtype('int32'))

In [16]: arr_float = arr.astype(np.float64)
         arr_float.dtype
```

```
Out[16]: dtype('float64')
```

13) . Reshape → Ravel → Verify

```
In [17]: original = np.arange(12)
         reshaped = original.reshape(3,4)
         flattened = reshaped.ravel()

         np.array_equal(original, flattened)
```

```
Out[17]: True
```

14) Element-wise Operations

```
In [18]: a = np.array([1,2,3])
         b = np.array([4,5,6])

         a + b, a - b, a * b, a / b
```

```
Out[18]: (array([5, 7, 9]),
          array([-3, -3, -3]),
          array([ 4, 10, 18]),
          array([0.25, 0.4 , 0.5 ]))
```

15) . Broadcasting Example

```
In [19]: arr_2d = np.array([[1],[2],[3]])
         arr_1d = np.array([10,20,30])

         arr_2d + arr_1d
```

```
Out[19]: array([[11, 21, 31],
                [12, 22, 32],
                [13, 23, 33]], shape=(3, 3))
```

16) Boolean Mask & Replace Values

```
In [20]: arr = np.random.randint(0, 11, size=(4,4))
         arr[arr > 5] = 5
         arr
```

```
Out[20]: array([[5, 5, 5, 5],
                [5, 4, 5, 5],
                [5, 5, 4, 2],
                [1, 5, 5, 5]], shape=(4, 4), dtype=int32)
```

17) Indexing & Slicing

```
In [21]: arr = np.random.randint(1, 20, size=(4,4))

         arr[1], arr[:, -1], arr[:2, :2]
```

```
Out[21]: (array([ 2, 11,  7, 10], dtype=int32),
          array([ 5, 10,  9,  8], dtype=int32),
          array([[11,  4],
                 [ 2, 11]], dtype=int32))
```

18) NumPy in EDA, AI, ML, DL (Simple Example)

```
In [22]: data = np.array([10, 20, 30, 40, 50])
```

```
np.mean(data), np.std(data)
```

```
Out[22]: (np.float64(30.0), np.float64(14.142135623730951))
```

19) Eigenvalues and Eigenvectors

```
In [23]: matrix = np.random.rand(4,4)
         np.linalg.eig(matrix)
```

```
Out[23]: EigResult(eigenvalues=array([ 2.0065717 +0.j        ,  0.40887828+0.j
         ,
                -0.11984977+0.21445674j, -0.11984977-0.21445674j]), eigenvectors=arra
         y([[ 0.43568489+0.j        , -0.81259584+0.j        ,
                 -0.03925217-0.04939078j, -0.03925217+0.04939078j],
                [ 0.40794899+0.j        ,  0.36646568+0.j        ,
                  0.38469545+0.16896826j,  0.38469545-0.16896826j],
                [ 0.41435002+0.j        ,  0.43985056+0.j        ,
                  0.49922467-0.20231488j,  0.49922467+0.20231488j],
                [ 0.68707377+0.j        , -0.10918972+0.j        ,
                 -0.72754544+0.j        , -0.72754544-0.j        ]], shape=(4, 4)))
```

20) 3D Reshape and Flatten

```
In [24]: arr = np.arange(27)
         arr_3d = arr.reshape(3,3,3)
         np.array_equal(arr, arr_3d.flatten())
```

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

21) Matrix Multiplication

```
In [25]: A = np.array([[1, 2], [3, 4]])

         B = np.array([[5, 6],[7, 8]])

         A * B
```

```
Out[25]: array([[ 5, 12],
                [21, 32]])
```

22) Broadcasting & newaxis

```
In [26]: a = np.random.rand(2,1,4)
         b = np.random.rand(4,1)

         a + b.T
```

```
Out[26]: array([[[0.93754585, 1.29309612, 0.99170089, 1.03857263]],

                [[1.51946114, 1.7683418 , 0.47549441, 0.5353723 ]]],
               shape=(2, 1, 4))
```

23) Conditional Replace with Square

```
In [27]: arr = np.random.rand(4,4)
         arr[arr < 0.5] **= 2
         arr
```

```
Out[27]: array([[3.55417288e-02, 1.68402045e-03, 1.97707390e-02, 2.66992301e-03],
               [9.72939723e-01, 5.10381346e-02, 2.09000945e-05, 1.04599106e-01],
               [2.11709060e-01, 1.80874618e-01, 5.87756599e-01, 1.68517603e-01],
               [5.93104498e-01, 1.70542708e-02, 5.91218080e-02, 5.24443409e-01]],
              shape=(4, 4))
```

24) Slicing Operations

```
In [28]: arr = np.arange(1, 26).reshape(5, 5)
         arr
```

```
Out[28]: array([[ 1,  2,  3,  4,  5],
               [ 6,  7,  8,  9, 10],
               [11, 12, 13, 14, 15],
               [16, 17, 18, 19, 20],
               [21, 22, 23, 24, 25]], shape=(5, 5))
```

```
In [29]: arr[2] = 0
         arr
```

```
Out[29]: array([[ 1,  2,  3,  4,  5],
               [ 6,  7,  8,  9, 10],
               [ 0,  0,  0,  0,  0],
               [16, 17, 18, 19, 20],
               [21, 22, 23, 24, 25]], shape=(5, 5))
```

```
In [32]: np.flipud(arr)
```

```
Out[32]: array([[21, 22, 23, 24, 25],
               [16, 17, 18, 19, 20],
               [ 0,  0,  0,  0,  0],
               [ 6,  7,  8,  9, 10],
               [ 1,  2,  3,  4,  5]], shape=(5, 5))
```

```
In [31]: np.fliplr(arr)
```

```
Out[31]: array([[ 5,  4,  3,  2,  1],
               [10,  9,  8,  7,  6],
               [ 0,  0,  0,  0,  0],
               [20, 19, 18, 17, 16],
               [25, 24, 23, 22, 21]], shape=(5, 5))
```

25) Create a 4D array of shape (2, 3, 4, 5) with random integers. Use advanced slicing to extract a subarray and compute the mean along a specified axis.

```
In [33]: arr = np.random.randint(1, 10, size=(2, 3, 4, 5))
         arr
```

```
Out[33]: array([[[[1, 2, 1, 7, 4],
                  [8, 2, 4, 8, 4],
                  [6, 4, 4, 7, 1],
                  [4, 7, 8, 4, 1]],

                 [[4, 4, 9, 2, 9],
                  [4, 6, 9, 4, 8],
                  [9, 6, 2, 3, 1],
                  [8, 2, 3, 5, 1]],

                 [[2, 4, 1, 1, 4],
                  [6, 9, 6, 5, 1],
                  [6, 3, 6, 1, 1],
                  [6, 9, 1, 1, 6]]],


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

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

                 [[9, 2, 6, 8, 5],
                  [3, 6, 5, 8, 6],
                  [1, 3, 6, 4, 6],
                  [3, 4, 6, 3, 1]]]], shape=(2, 3, 4, 5), dtype=int32)
```

```
In [34]: sub_arr = arr[:, :2, :2, :3]
         sub_arr
```

```
Out[34]: array([[[[1, 2, 1],
                  [8, 2, 4]],

                 [[4, 4, 9],
                  [4, 6, 9]]],


                [[[4, 9, 8],
                  [1, 1, 4]],

                 [[8, 4, 5],
                  [2, 9, 6]]]], shape=(2, 2, 2, 3), dtype=int32)
```

```
In [35]: mean_result = np.mean(sub_arr, axis=2)
         mean_result
```

```
Out[35]: array([[[4.5, 2. , 2.5],
                 [4. , 5. , 9. ]],

                [[2.5, 5. , 6. ],
                 [5. , 6.5, 5.5]]], shape=(2, 2, 3))
```

26) iven an array of shape (10, 20), reshape it to (20, 10) and (5, 40). Discuss the impact on the array's shape, size, and dimensionality.

```
In [36]: arr = np.arange(200).reshape(10, 20)
         arr.shape
```

```
Out[36]: (10, 20)
```

```
In [37]: arr_20_10 = arr.reshape(20, 10)
         arr_20_10.shape
```

```
Out[37]: (20, 10)
```

```
In [38]: arr_5_40 = arr.reshape(5, 40)
         arr_5_40.shape
```

```
Out[38]: (5, 40)
```

Generate a large 2D array and demonstrate the use of np. reshape () and unravel () to manipulate its shape for various linear algebra operations

```
In [39]: arr = np.arange(100).reshape(10, 10)
         arr
```

```
Out[39]: array([[ 0,  1,  2, ...,  7,  8,  9],
                [10, 11, 12, ..., 17, 18, 19],
                [20, 21, 22, ..., 27, 28, 29],
                ...,
                [70, 71, 72, ..., 77, 78, 79],
                [80, 81, 82, ..., 87, 88, 89],
                [90, 91, 92, ..., 97, 98, 99]], shape=(10, 10))
```

```
In [40]: reshaped_arr = arr.reshape(20, 5)
         reshaped_arr
```

```
Out[40]: array([[ 0,  1,  2,  3,  4],
                [ 5,  6,  7,  8,  9],
                [10, 11, 12, 13, 14],
                ...,
                [85, 86, 87, 88, 89],
                [90, 91, 92, 93, 94],
                [95, 96, 97, 98, 99]], shape=(20, 5))
```

```
In [41]: np.unravel_index(37, reshaped_arr.shape)
```

```
Out[41]: (np.int64(7), np.int64(2))
```

```
In [42]: reshaped_arr[np.unravel_index(37, reshaped_arr.shape)]
```

```
Out[42]: np.int64(37)
```

28) Question: Given a 6x6 matrix, use advanced indexing and slicing to extract the upper triangular part of the matrix and set the lower triangular part to zero. Verify the result.

```
In [43]: matrix = np.arange(1, 37).reshape(6, 6)
         matrix
```

```
Out[43]: array([[ 1,  2,  3,  4,  5,  6],
                [ 7,  8,  9, 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]], shape=(6, 6))
```

```
In [44]: upper_tri = np.triu(matrix)
         upper_tri
```

```
Out[44]: array([[ 1,  2,  3,  4,  5,  6],
                [ 0,  8,  9, 10, 11, 12],
                [ 0,  0, 15, 16, 17, 18],
                [ 0,  0,  0, 22, 23, 24],
                [ 0,  0,  0,  0, 29, 30],
                [ 0,  0,  0,  0,  0, 36]], shape=(6, 6))
```

```
In [45]: np.tril(upper_tri, -1)
```

```
Out[45]: 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]], shape=(6, 6))
```

```
In [ ]:
```