Transposing NumPy Arrays

- Transposition is a special form of data reorganization.
- It shows the transposed view of the underlying data without modifying any content.
- Transposing of NumPy arrays canbe done using:

1. np.transpose()
2. .T attribute

A. Using transpose()

- The numpy ndarary **transpose()** function is used to transpose a numpy array.

> **Syntax:**

> new = arr.transpose()

```
#arr is a numpy array
```

> It returns a view of the array with the axes transposed.

- This function transposes the axis by transposing the index value of the row and column values of the array.
- On transpose, the first column becomes the first row, the second column becomes the second row, and the third column becomes the third row.

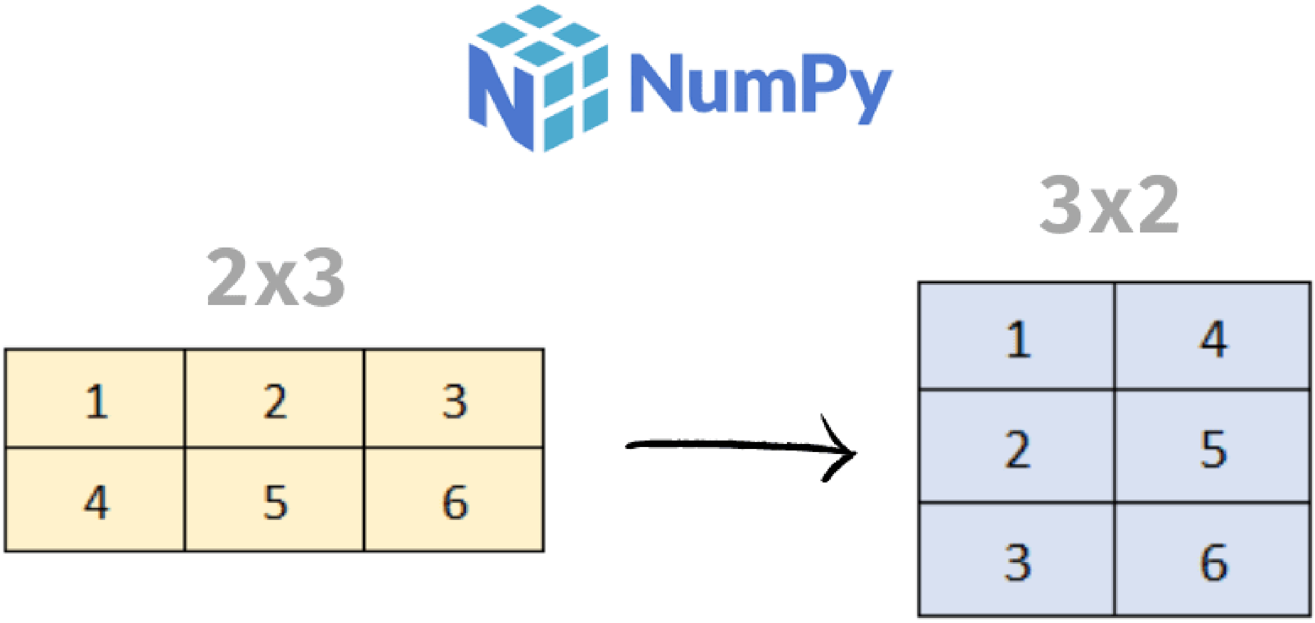
> For example, a numpy array of shape (2, 3) becomes a numpy array of shape (3, 2) after the operation wherein the first row becomes the first column and the second row becomes the second column.

B. Using .T attribute

- .T attribute performs the transpose same as that of transpose().

> **Syntax:**

> new = arr.T



Transpose array

Numpy Transpose 1d array

- For 1d arrays, the transpose operation has no effect on the array. As a transposed vector it is simply the same vector.
- Both the arrays (original and transposed) have the same shape.

```
import numpy as np

# create a 1d numpy array
arr = np.array([1, 2, 3, 4])
print(arr)
print("\n")

# transpose the array
arr_t = arr.transpose()
print(arr_t)
print("\n")

arr_t1 = arr.T
print(arr_t1)
```

```
[1 2 3 4]


[1 2 3 4]


[1 2 3 4]
```

- You can see that transposing a 1d array doesn’t change anything. We can further confirm this by looking at the shape of the two arrays:

```
 # print the shape of the two arrays
print("Original array shape: ", arr.shape)
print("Shape after transpose: ", arr_t.shape)
print("Shape after transpose: ", arr_t1.shape)
```

```
     Original array shape:  (4,)
     Shape after transpose:  (4,)
     Shape after transpose:  (4,)
```

```
a_1d = np.arange(3)
print(a_1d)
print("\n")

print(a_1d.transpose())
print("\n")

#OR

print(a_1d.T)
print("\n")

#OR

print(np.transpose(a_1d))
print("\n")
```

```
     [0 1 2]


     [0 1 2]


     [0 1 2]


     [0 1 2]
```

Numpy Transpose 2d array

- For a 2d array, the transpose operation means to swap out the rows and columns of the array.

Example 1:

```
import numpy as np

# create a 1d numpy array
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
```

```
print(arr)
print("\n")

# transpose the array
arr_t = arr.transpose()
print(arr_t)
print("\n")

#OR

arr_t1 = arr.T
print(arr_t1)
```

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


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


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

- Here, we transpose a 2×3 array. Note that after the transpose, the first row in the original array [1, 2, 3] becomes the first column in the transposed array and similarly the second row [4, 5, 6] becomes the second column in the transposed array.

```
# print the shape of the two arrays
print("Original array shape: ", arr.shape)
print("Shape after transpose: ", arr_t.shape)
print("Shape after transpose: ", arr_t1.shape)
```

```
Original array shape:  (2, 3)
Shape after transpose:  (3, 2)
Shape after transpose:  (3, 2)
```

Example 2:

```
arr1 = np.arange(15).reshape ((3,5))

print(arr1)
print("\n")

t = arr1.transpose()
print(t)
print("\n")

#OR

t1 = arr1.transpose()
print(t1)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]


[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]]


[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]]
```

```
import numpy as np

a = np.arange(6).reshape(2, 3)
print(a_2d)
print("\n")

b = a.transpose()
print(b)
print("\n")

#OR

c = a.T
print(c)
```

```
[[0 1 2]
 [3 4 5]]


[[0 3]
 [1 4]
 [2 5]]


[[0 3]
 [1 4]
 [2 5]]
```

Numpy Transpose 3d array

- For a 3d array, the previous dimensions are reordered, that is, the first dimension before the second dimension is converted to the first dimension becomes the second dimension, and the last axis remains unchanged.

```
import numpy as np
arr = np . arange (16). reshape ((2,2,4))
print(arr)



print("\n")
```

```
t1 = arr.transpose()
print(t1)

print("Original array shape: ", arr.shape)
print("Shape after transpose: ", t1.shape)
```

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

 [[ 8  9 10 11]
  [12 13 14 15]]]


[[[ 0  8]
  [ 4 12]]

 [[ 1  9]
  [ 5 13]]

 [[ 2 10]
  [ 6 14]]

 [[ 3 11]
  [ 7 15]]]
Original array shape:  (2, 2, 4)
Shape after transpose:  (4, 2, 2)
```

```
a_3d = np.arange(24).reshape(2, 3, 4)
print(a_3d)
print("\n")

x = a_3d.transpose()
print(x)

print("Original array shape: ", a_3d.shape)
print("Shape after transpose: ", x.shape)
```

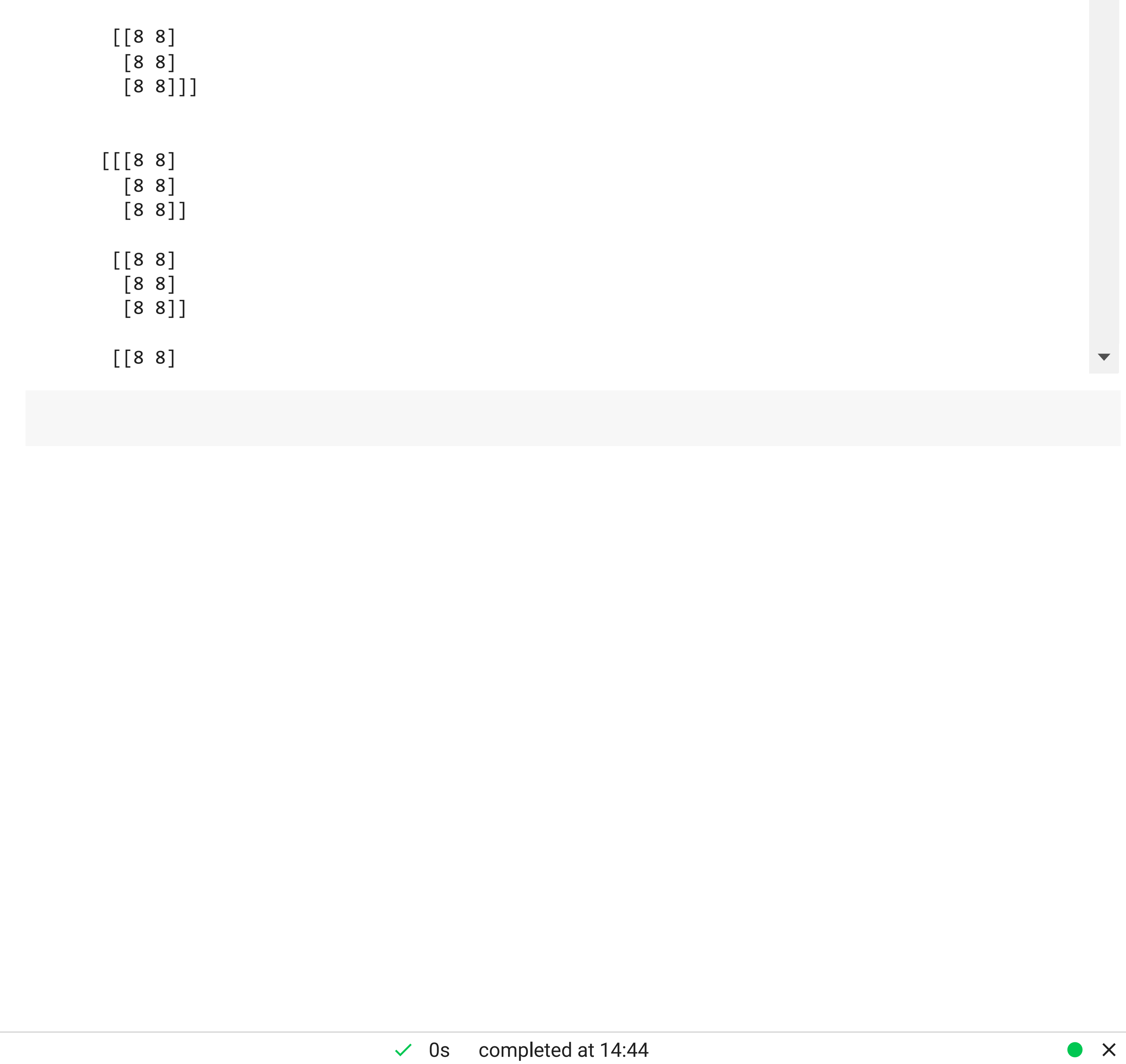
```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]


[[[ 0 12]
  [ 4 16]
  [ 8 20]]

 [[ 1 13]
  [ 5 17]
  [ 9 21]]

 [[ 2 14]
  [ 6 18]
  [10 22]]

 [[ 3 15]
  [ 7 19]
  [11 23]]]
```

```
Original array shape:  (2, 3, 4)
Shape after transpose:  (4, 3, 2)
```

Numpy Transpose 4d array

```
x = np.full((2, 3, 4, 5),8)
print(x)
print(x.ndim)
print(x.shape)

y = x.transpose()
print(y)
print(y.ndim)
print(y.shape)
```

```
[[[[8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]]

  [[8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]]

  [[8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]]]


 [[[8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]]

  [[8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]]

  [[8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]
   [8 8 8 8 8]]]]
4
(2, 3, 4, 5)
[[[[8 8]
   [8 8]
   [8 8]]

  [[8 8]
   [8 8]
   [8 8]]

  [[8 8]
   [8 8]
   [8 8]]
```



```
 [[8 8]
  [8 8]
  [8 8]]]


[[[8 8]
  [8 8]
  [8 8]]

 [[8 8]
  [8 8]
  [8 8]]

 [[8 8]
```

✓ 0s completed at 14:44