TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A REPORT ON

Usage of NumPy (Lab-7)

SUBMITTED BY:

SOVIT KHAREL (081BEL081)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

1. NumPy Built-in Functions with Examples

numpy.floor

Rounds down elements of an array to the nearest integer.

Example:

```
import numpy as np
arr = np.array([1.7, 2.9, -3.1])
print(np.floor(arr))
```

Output:

```
[ 1.  2. -4.]
```

numpy.ceil

Rounds up elements of an array to the nearest integer.

Example:

```
arr = np.array([1.2, 2.8, -3.5])
print(np.ceil(arr))
```

Output:

```
[ 2.  3. -3.]
```

numpy.round

Rounds elements of an array to the nearest integer or specified decimals.

Example:

```
arr = np.array([1.234, 2.678, 3.456])
print(np.round(arr, 2))
```

Output:

```
[1.23 2.68 3.46]
```

numpy.clip

Limits values in an array between given minimum and maximum.

Example:

```
arr = np.array([1, 5, 8, 10])
print(np.clip(arr, 3, 7))
```

Output:

```
[3 5 7 7]
```

numpy.linspace

Creates evenly spaced numbers over a specified interval.

Example:

```
print(np.linspace(0, 1, 5))
```

Output:

```
[0.   0.25 0.5  0.75 1.  ]
```

numpy.logspace

Creates numbers spaced evenly on a log scale.

Example:

```
print(np.logspace(1, 3, 3))
```

Output:

```
[  10.  100. 1000.]
```

numpy.identity

Creates an identity matrix of given size.

Example:

```
print(np.identity(3))
```

Output:

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

numpy.eye

Creates a 2D array with ones on the diagonal and zeros elsewhere.

Example:

```
print(np.eye(3, 4))
```

Output:

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

numpy.all

Checks if all elements of an array are True.

Example:

```
arr = np.array([True, True, False])
print(np.all(arr))
```

Output:

```
False
```

numpy.any

Checks if any element of an array is True.

Example:

```
arr = np.array([0, 0, 1])
print(np.any(arr))
```

Output:

```
True
```

numpy.argmax

Returns the index of the maximum element.

Example:

```
arr = np.array([1, 3, 7, 2])
print(np.argmax(arr))
```

Output:

```
2
```

numpy.argmin

Returns the index of the minimum element.

Example:

```
arr = np.array([1, 3, 7, 2])
print(np.argmin(arr))
```

Output:

```
0
```

numpy.where

Returns indices where condition is True.

Example:

```
arr = np.array([10, 20, 30, 40])
print(np.where(arr > 20))
```

Output:

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

numpy.sort

Sorts an array.

Example:

```
arr = np.array([3, 1, 2])
print(np.sort(arr))
```

Output:

```
[1 2 3]
```

numpy.unique

Finds unique elements in an array.

Example:

```
arr = np.array([1, 2, 2, 3, 3, 3])
print(np.unique(arr))
```

Output:

```
[1 2 3]
```

numpy.tile

Repeats an array a specified number of times.

Example:

```
arr = np.array([1, 2])
print(np.tile(arr, 3))
```

Output:

```
[1 2 1 2 1 2]
```

numpy.repeat

Repeats each element of an array a specified number of times.

Example:

```
arr = np.array([1, 2, 3])
print(np.repeat(arr, 2))
```

Output:

```
[1 2 1 2 1 2]
```

numpy.reshape

Changes the shape of an array.

Example:

```
arr = np.arange(6)
print(np.reshape(arr, (2, 3)))
```

Output:

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

numpy.ravel

Flattens an array into 1D.

Example:

```
arr = np.array([[1, 2], [3, 4]])
print(np.ravel(arr))
```

Output:

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

numpy.hstack

Stacks arrays horizontally.

Example:

```
a = np.array([1, 2])
b = np.array([3, 4])
print(np.hstack((a, b)))
```

Output:

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

numpy.vstack

Stacks arrays vertically.

Example:

```
a = np.array([1, 2])
b = np.array([3, 4])
print(np.vstack((a, b)))
```

Output:

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

numpy.dstack

Stacks arrays along the depth (third axis).

Example:

```
a = np.array([1, 2])
b = np.array([3, 4])
print(np.dstack((a, b)))
```

Output:

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

numpy.split

Splits an array into multiple sub-arrays.

Example:

```
arr = np.array([1, 2, 3, 4, 5, 6])
print(np.split(arr, 3))
```

Output:

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

numpy.array_split

Splits array into unequal parts if needed.

Example:

```
arr = np.array([1, 2, 3, 4, 5])
print(np.array_split(arr, 3))
```

Output:

```
[array([1, 2]), array([3, 4]), array([5])]
```

numpy.mean

Computes the mean of elements.

Example:

```
arr = np.array([1, 2, 3, 4])
print(np.mean(arr))
```

Output:

```
2.5
```

numpy.median

Computes the median of elements.

Example:

```
arr = np.array([1, 3, 2, 4])
print(np.median(arr))
```

Output:

```
2.5
```

numpy.std

Computes standard deviation of elements.

Example:

```
arr = np.array([1, 2, 3, 4])
print(np.std(arr))
```

Output:

```
1.118033988749895
```

numpy.var

Computes variance of elements.

Example:

```
arr = np.array([1, 2, 3, 4])
print(np.var(arr))
```

Output:

```
1.25
```

2.Executed examples in:

https://numpy.org/doc/stable/user/quickstart.html

Basics:

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize\
...
KeyboardInterrupt
>>> a.itemsize
8
>>> a.size
15
>>>
>>> type(a)
<class 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<class 'numpy.ndarray'>
>>>
```

Array Creation:

```
>>> a = np.array([2, 3, 4])
>>>
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
>>>
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
>>>
```

```
>>> b = np.array([(1.5, 2, 3), (4, 5, 6)])
>>> b
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
>>> c = np.array([[1, 2], [3, 4]], dtype=complex)
>>> c
array([[1.+0.j, 2.+0.j],
       [3.+0.j, 4.+0.j]])
>>> np.zeros((3, 4))
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
>>> np.ones((2, 3, 4), dtype=np.int16)
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]], dtype=int16)
>>> np.empty((2, 3))
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
```

Printing Arrays:

```
>>> a = np.arange(6)
>>> print(a)
[0 1 2 3 4 5]
>>> b = np.arange(12).reshape(4, 3)
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>> c = np.arange(24).reshape(2, 3, 4)
>>> print(c)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

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

Basic Operations:

```
>>> a = np.array([20, 30, 40, 50])
>>> b = np.arange(4)
>>> b
array([0, 1, 2, 3])
>>> c = a - b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10 * np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a < 35
array([ True,  True, False, False])
>>>
```

Universal Functions:

```
>>> B = np.arange(3)
>>> B
array([0, 1, 2])
>>> np.exp(B)
array([1.        , 2.71828183, 7.3890561 ])
>>> np.sqrt(B)
array([0.        , 1.        , 1.41421356])
>>> C = np.array([2., -1., 4.])
>>> np.add(B, C)
array([2., 0., 6.])
>>>
```

Indexing, Slicing and Iterating:

```
>>> a = np.arange(10)**3
>>>
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
np.int64(8)
>>> a[2:5]
array([ 8, 27, 64])
>>> a[:6:2] = 1000
>>> a
array([1000,    1, 1000,   27, 1000,  125,  216,  343,  512,  729])
>>> a[::-1]
array([ 729,  512,  343,  216,  125, 1000,   27, 1000,    1, 1000])
>>> for i in a:
...
...         print(i**(1 / 3.))
...
9.999999999999998
1.0
9.999999999999998
3.0
9.999999999999998
5.0
5.999999999999999
6.999999999999999
7.999999999999999
8.999999999999998
>>>
```

Stacking together different arrays:

```
>>> a = np.floor(10 * rg.random((2, 2)))
>>> a
array([[5., 9.],
       [1., 9.]])
>>> b = np.floor(10 * rg.random((2, 2)))
>>>
>>> b
array([[3., 4.],
       [8., 4.]])
>>> np.vstack((a, b))
array([[5., 9.],
       [1., 9.],
       [3., 4.],
       [8., 4.]])
>>>
>>> np.hstack((a, b))
array([[5., 9., 3., 4.],
       [1., 9., 8., 4.]])
>>>
>>>
```

```
>>> a = np.floor(10 * rg.random((2, 2)))
>>> a
array([[5., 9.],
       [1., 9.]])
>>> b = np.floor(10 * rg.random((2, 2)))
>>>
>>> b
array([[3., 4.],
       [8., 4.]])
>>> np.vstack((a, b))
array([[5., 9.],
       [1., 9.],
       [3., 4.],
       [8., 4.]])
>>>
>>> np.hstack((a, b))
array([[5., 9., 3., 4.],
       [1., 9., 8., 4.]])
>>>
>>> from numpy import newaxis
>>> np.column_stack((a, b))
array([[5., 9., 3., 4.],
       [1., 9., 8., 4.]])
>>> a = np.array([4., 2.])
>>> b = np.array([3., 8.])
>>> np.column_stack((a, b)
... )
array([[4., 3.],
       [2., 8.]])
>>> np.hstack((a, b))
array([4., 2., 3., 8.])
>>> a[:, newaxis]
array([[4.],
       [2.]])
>>> np.column_stack((a[:, newaxis], b[:, newaxis]))
array([[4., 3.],
       [2., 8.]])
>>> np.hstack((a[:, newaxis], b[:, newaxis]))
array([[4., 3.],
       [2., 8.]])
>>>
```

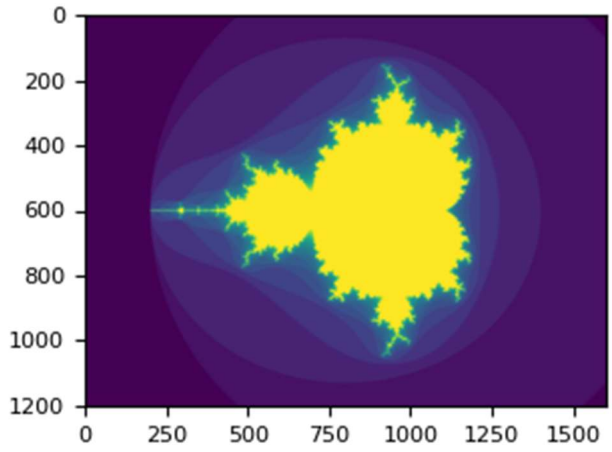
Splitting one array into several smaller ones:

```
>>> a = np.floor(10 * rg.random((2, 12)))
>>> a
array([[5., 0., 7., 5., 3., 7., 3., 4., 1., 4., 2., 2.],
       [7., 2., 4., 9., 9., 7., 5., 2., 1., 9., 5., 1.]])
>>> np.hsplit(a, 3)
[array([[5., 0., 7., 5.],
       [7., 2., 4., 9.]]), array([[3., 7., 3., 4.],
       [9., 7., 5., 2.]]), array([[1., 4., 2., 2.],
       [1., 9., 5., 1.]])]
>>> np.hsplit(a, (3, 4))
[array([[5., 0., 7.],
       [7., 2., 4.]]), array([[5.],
       [9.]]), array([[3., 7., 3., 4., 1., 4., 2., 2.],
       [9., 7., 5., 2., 1., 9., 5., 1.]])]
>>>
```

Indexing with Boolean array:

```
>>> a = np.arange(12).reshape(3, 4)
>>>
>>> b = a > 4
>>>
>>> b
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True]])
>>> a[b]
array([ 5,  6,  7,  8,  9, 10, 11])
>>> a[b] = 0
>>> a
array([[0, 1, 2, 3],
       [4, 0, 0, 0],
       [0, 0, 0, 0]])
>>>
```

```
>>>
>>> def mandelbrot(h, w, maxit=20, r=2):
...         x = np.linspace(-2.5, 1.5, 4*h+1)
...         y = np.linspace(-1.5, 1.5, 3*w+1)
...         A, B = np.meshgrid(x, y)
...         C = A + B*1j
...         z = np.zeros_like(C)
...         divtime = maxit + np.zeros(z.shape, dtype=int)
...         for i in range(maxit):
...                 z = z**2 + C
...                 diverge = abs(z) > r
...                 div_now = diverge & (divtime == maxit)
...                 divtime[div_now] = i
...                 z[diverge] = r
...         return divtime
...
>>> plt.clf()
>>> plt.imshow(mandelbrot(400, 400))
<matplotlib.image.AxesImage object at 0x0000020FD37112B0>
>>>
```



3. Executed examples in:

https://numpy.org/devdocs/user/absolute_beginners.html

Array fundamentals:

```
>>> a = np.array([1, 2, 3, 4, 5, 6])
>>> a
array([1, 2, 3, 4, 5, 6])
```

```
>>> a[0]
1
```

```
>>> a[0] = 10
>>> a
array([10,  2,  3,  4,  5,  6])
```

```
>>> a[:3]
array([10, 2, 3])
```

```
>>> b = a[3:]
>>> b
array([4, 5, 6])
>>> b[0] = 40
>>> a
array([ 10,  2,  3, 40,  5,  6])
```

```
>>> a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
>>> a
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

```
>>> a[1, 3]
8
```

Array Attributes:

```
>>> a.ndim
2
```

```
>>> a.shape
(3, 4)
>>> len(a.shape) == a.ndim
True
```

```
>>> a.size
12
>>> import math
>>> a.size == math.prod(a.shape)
True
```

```
>>> a.dtype
dtype('int64')  # "int" for integer, "64" for 64-bit
```

Adding, removing, and sorting elements:

```
>>> arr = np.array([2, 1, 5, 3, 7, 4, 6, 8])
```

```
>>> np.sort(arr)
array([1, 2, 3, 4, 5, 6, 7, 8])
```

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

```
>>> np.concatenate((a, b))
array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
>>> x = np.array([[1, 2], [3, 4]])
>>> y = np.array([[5, 6]])
```

```
>>> np.concatenate((x, y), axis=0)
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Reshaping an array:

```
>>> a = np.arange(6)
>>> print(a)
[0 1 2 3 4 5]
```

```
>>> b = a.reshape(3, 2)
>>> print(b)
[[0 1]
 [2 3]
 [4 5]]
```

```
>>> np.reshape(a, shape=(1, 6), order='C')
array([[0, 1, 2, 3, 4, 5]])
```

Converting 1D array into 2D:

```
>>> a = np.array([1, 2, 3, 4, 5, 6])
>>> a.shape
(6,)
```

```
>>> a2 = a[np.newaxis, :]
>>> a2.shape
(1, 6)
```

```
>>> a2 = a[np.newaxis, :]
>>> a2.shape
(1, 6)
```

```
>>> row_vector = a[np.newaxis, :]
>>> row_vector.shape
(1, 6)
```

```
>>> col_vector = a[:, np.newaxis]
>>> col_vector.shape
(6, 1)
```

```
>>> a = np.array([1, 2, 3, 4, 5, 6])
>>> a.shape
(6,)
```

```
>>> b = np.expand_dims(a, axis=1)
>>> b.shape
(6, 1)
```

Indexing and Slicing:

```
>>> data = np.array([1, 2, 3])

>>> data[1]
2
>>> data[0:2]
array([1, 2])
>>> data[1:]
array([2, 3])
>>> data[-2:]
array([2, 3])
```

```
>>> a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

```
>>> print(a[a < 5])
[1 2 3 4]
```

```
>>> five_up = (a >= 5)
>>> print(a[five_up])
[ 5  6  7  8  9 10 11 12]
```

```
>>> divisible_by_2 = a[a%2==0]
>>> print(divisible_by_2)
[ 2  4  6  8 10 12]
```

```
>>> five_up = (a > 5) | (a == 5)
>>> print(five_up)
[[False False False False]
 [ True  True  True  True]
 [ True  True  True True]]
```

```
>>> b = np.nonzero(a < 5)
>>> print(b)
(array([0, 0, 0, 0]), array([0, 1, 2, 3]))
```

```
>>> list_of_coordinates= list(zip(b[0], b[1]))

>>> for coord in list_of_coordinates:
...     print(coord)
(np.int64(0), np.int64(0))
(np.int64(0), np.int64(1))
(np.int64(0), np.int64(2))
(np.int64(0), np.int64(3))
```

Creating an array from an existing data:

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

```
>>> arr1 = a[3:8]
>>> arr1
array([4, 5, 6, 7, 8])
```

```
>>> a1 = np.array([[1, 1],
...                [2, 2]])

>>> a2 = np.array([[3, 3],
...                [4, 4]])
```

```
>>> np.hstack((a1, a2))
array([[1, 1, 3, 3],
       [2, 2, 4, 4]])
```

```
>>> x = np.arange(1, 25).reshape(2, 12)
>>> x
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]])
```

```
>>> np.hsplit(x, 3)
  [array([[ 1,  2,  3,  4],
         [13, 14, 15, 16]]), array([[ 5,  6,  7,  8],
         [17, 18, 19, 20]]), array([[ 9, 10, 11, 12],
         [21, 22, 23, 24]])]
```

```
>>> np.hsplit(x, (3, 4))
  [array([[ 1,  2,  3],
         [13, 14, 15]]), array([[ 4],
         [16]]), array([[ 5,  6,  7,  8,  9, 10, 11, 12],
         [17, 18, 19, 20, 21, 22, 23, 24]])]
```

```
>>> b1 = a[0, :]
>>> b1
array([1, 2, 3, 4])
>>> b1[0] = 99
>>> b1
array([99,  2,  3,  4])
>>> a
array([[99,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

Basic Array Operations:

```
>>> data = np.array([1, 2])
>>> ones = np.ones(2, dtype=int)
>>> data + ones
array([2, 3])
```

```
>>> data - ones
array([0, 1])
>>> data * data
array([1, 4])
>>> data / data
array([1., 1.])
```

```
>>> a = np.array([1, 2, 3, 4])

>>> a.sum()
10
```

```
>>> b = np.array([[1, 1], [2, 2]])
```

```
>>> b.sum(axis=0)
array([3, 3])
```

```
>>> b.sum(axis=1)
array([2, 4])
```

Creating matrices:

```
>>> data = np.array([[1, 2], [3, 4], [5, 6]])
>>> data
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
>>> data[0, 1]
2
>>> data[1:3]
array([[3, 4],
       [5, 6]])
>>> data[0:2, 0]
array([1, 3])
```

```
>>> data.max()
6
>>> data.min()
1
>>> data.sum()
21
```

```
>>> data = np.array([[1, 2], [5, 3], [4, 6]])
>>> data
array([[1, 2],
       [5, 3],
       [4, 6]])
>>> data.max(axis=0)
array([5, 6])
>>> data.max(axis=1)
array([2, 5, 6])
```

```
>>> data = np.array([[1, 2], [3, 4]])
>>> ones = np.array([[1, 1], [1, 1]])
>>> data + ones
array([[2, 3],
       [4, 5]])
```

```
>>> data = np.array([[1, 2], [3, 4], [5, 6]])
>>> ones_row = np.array([[1, 1]])
>>> data + ones_row
array([[2, 3],
       [4, 5],
       [6, 7]])
```

```
>>> np.ones((4, 3, 2))
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]]])
```

```
>>> np.ones(3)
array([1., 1., 1.])
>>> np.zeros(3)
array([0., 0., 0.])
>>> rng = np.random.default_rng()  # the simplest way to generate random numbers
>>> rng.random(3)
array([0.63696169, 0.26978671, 0.04097352])
```

```
>>> np.ones((3, 2))
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
>>> np.zeros((3, 2))
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
>>> rng.random((3, 2))
array([[0.01652764, 0.81327024],
       [0.91275558, 0.60663578],
       [0.72949656, 0.54362499]])  # may vary
```

Generating Random Numbers:

```
>>> rng.integers(5, size=(2, 4))
array([[2, 1, 1, 0],
       [0, 0, 0, 4]])  # may vary
```

```
>>> a = np.array([11, 11, 12, 13, 14, 15, 16, 17, 12, 13, 11, 14, 18, 19, 20])
```

```
>>> unique_values = np.unique(a)
>>> print(unique_values)
[11 12 13 14 15 16 17 18 19 20]
```

```
>>> unique_values, indices_list = np.unique(a, return_index=True)
>>> print(indices_list)
[ 0  2  3  4  5  6  7 12 13 14]
```

```
>>> unique_values, occurrence_count = np.unique(a, return_counts=True)
>>> print(occurrence_count)
[3 2 2 2 1 1 1 1 1 1]
```

```
>>> a_2d = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [1, 2, 3, 4]])
```

```
>>> unique_values = np.unique(a_2d)
>>> print(unique_values)
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

```
>>> unique_rows = np.unique(a_2d, axis=0)
>>> print(unique_rows)
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
>>> unique_rows, indices, occurrence_count = np.unique(
...       a_2d, axis=0, return_counts=True, return_index=True)
>>> print(unique_rows)
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
>>> print(indices)
[0 1 2]
>>> print(occurrence_count)
[2 1 1]
```

Transposing and reshaping a matrix:

```
>>> data.reshape(2, 3)
array([[1, 2, 3],
       [4, 5, 6]])
>>> data.reshape(3, 2)
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
>>> arr = np.arange(6).reshape((2, 3))
>>> arr
array([[0, 1, 2],
       [3, 4, 5]])
```

```
>>> arr.transpose()
array([[0, 3],
       [1, 4],
       [2, 5]])
```

```
>>> arr.T
array([[0, 3],
       [1, 4],
       [2, 5]])
```

Reverse an array:

```
>>> arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
>>> reversed_arr = np.flip(arr)
```

```
>>> print('Reversed Array: ', reversed_arr)
Reversed Array:  [8 7 6 5 4 3 2 1]
```

Reshaping and flattening multidimensional arrays:

```
>>> x = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

```
>>> x.flatten()
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

```
>>> a1 = x.flatten()
>>> a1[0] = 99
>>> print(x)  # Original array
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
>>> print(a1)  # New array
[99  2  3  4  5  6  7  8  9 10 11 12]
```

```
>>> a2 = x.ravel()
>>> a2[0] = 98
>>> print(x)  # Original array
[[98  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
>>> print(a2)  # New array
[98  2  3  4  5  6  7  8  9 10 11 12]
```

Save and load NumPy objects:

```
>>> a = np.array([1, 2, 3, 4, 5, 6])
```

```
>>> np.save('filename', a)
```

```
>>> b = np.load('filename.npy')
```

```
>>> print(b)
[1 2 3 4 5 6]
```

```
>>> csv_arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
>>> np.savetxt('new_file.csv', csv_arr)
```

```
>>> np.loadtxt('new_file.csv')
array([1., 2., 3., 4., 5., 6., 7., 8.])
```

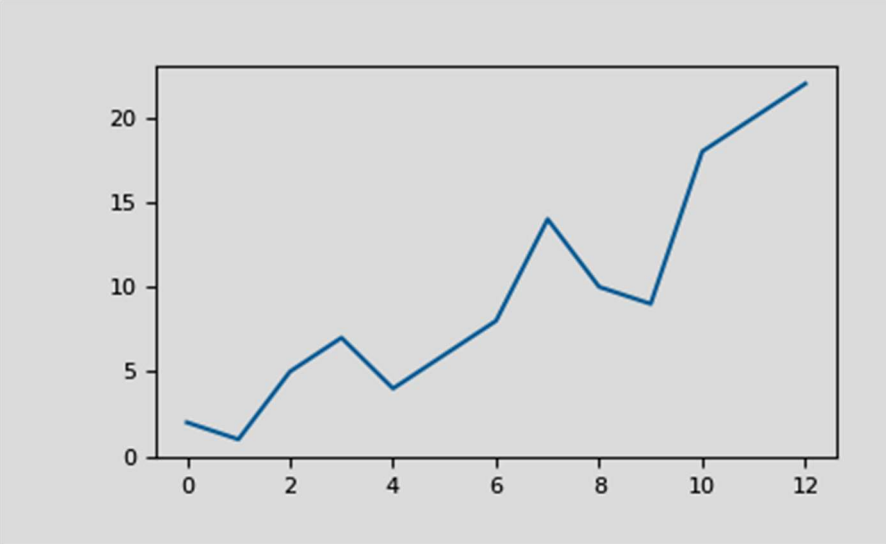
Plotting arrays with Matplotlib:

```
>>> a = np.array([2, 1, 5, 7, 4, 6, 8, 14, 10, 9, 18, 20, 22])
```

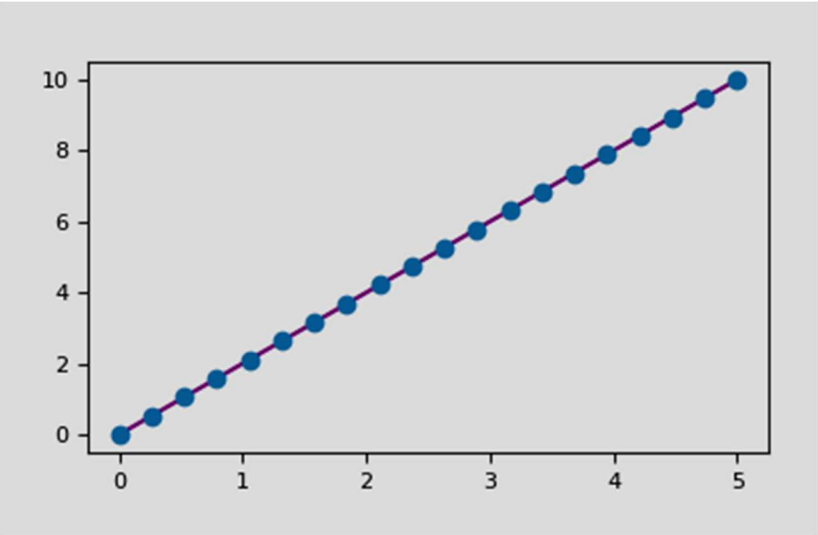
```
>>> import matplotlib.pyplot as plt
```

```
>>> plt.plot(a)
```

```
>>> plt.show()
```




```
>>> x = np.linspace(0, 5, 20)
>>> y = np.linspace(0, 10, 20)
>>> plt.plot(x, y, 'purple') # line
>>> plt.plot(x, y, 'o')      # dots
```


10
8
6
4
2
0
0
1
2
3
4
5