Import The Module

In [257…
```
import numpy as np
```

- Numpy: Stands for Numerical Python
- Is A Python Library Used for Working With Arrays
- Is Also Has Functions for Working in Domain of Linear Algebra, Fourier Transform, And Matrices
- Numpy Is Up To 50X Faster Than Traditional Python Lists
- The Array Object In Numpy Is Called "ndarray"

Create New List

In [258…
```
lst = ["osama", "mohamed", 120, 12.21, True]
print(lst)
```

```
['osama', 'mohamed', 120, 12.21, True]
```

Create 1D Array From Existing List

In [259…
```
array = np.array(lst)
print(array)
```

```
['osama' 'mohamed' '120' '12.21' 'True']
```

Get The Type Of Numpy Array

In [260…
```
print(type(array))
```

```
<class 'numpy.ndarray'>
```

Note That Will Create Array

- Array Elements Can Have Differnt Data Type
- If String Exist Will Convert All Types Of Elements In Array To String

If The Array Elemenst Are Only Numbers, and Float Number Is Exist With Integer Will Convert All Types To Floating Point Number

In [261…
```
array = np.array([100, 10.221, 20, 65])
print(array)
```

```
[100.     10.221  20.     65.   ]
```

Get The Type Of Array ELements

In [262…
```
print(array.dtype)
```

```
float64
```

Get The Shape Of The Array

- shape: get the form of array
- syntax: (rows, columns)

Get The Size Of Array

- Size: Is The Number Of Elements In Array
- Size = Rows * Columns

In [263…
```
print(array.size)
```

```
4
```

Determine The Dimensions Of Array

In [264…
```
print(array.ndim)
```

```
1
```

Create 2D Array

In [265…
```
array = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(array)
```

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

In [266…
```
print(f"Size Of Array: {array.size}")
print(f"Number Of Dimensions: {array.ndim}")
print(f"Shape Of Array: {array.shape}")
print(f"Type Of Array Elements: {array.dtype}")
print(f"Type Of The Array Object: {type(array)}")
```

```
Size Of Array: 12
Number Of Dimensions: 2
Shape Of Array: (3, 4)
Type Of Array Elements: int32
Type Of The Array Object: <class 'numpy.ndarray'>
```

Print The PI Value

In [267…
```
pi = np.pi
print(f"PI Value Is: {pi}")
```

```
PI Value Is: 3.141592653589793
```

Get The Trigonometry

In [268…
```
x = np.sin(90)
print(x)
```

```
0.8939966636005579
```

In [269…
```
x = np.cos(90)
print(x)
```

```
-0.4480736161291701
```

In [270…
```
x = np.tan(90)
print(x)
```

```
-1.995200412208242
```

In [271…
```
array = np.arange(0, 5)
print(np.sin(array))
```

```
[ 0.          0.84147098  0.90929743  0.14112001 -0.7568025 ]
```

Convert Data Types Of Array Elements

Create Array With Floating Point Number Values

In [272…
```
array = np.array([1.2, 1.5, 2.5])
print(array)
```

```
[1.2 1.5 2.5]
```

Create Array With Floating Point Number Values, Then Change Type Of Elements To Integer

In [273…
```
array = np.array([20.2, 10.21, 15.36], dtype="int64")
print(array)
```

```
[20 10 15]
```

Saving An Array To Binary File

In [274…
```
# Save Array To File With An Extension ".npy" Format
array = np.array([[1, 10, 20, 11], [14, 12, 13, 14]])
np.save("File_Name", array)
```

In [275…
```
# Then You Can Call The Array With Its Name
print(np.load("File_Name.npy"))
```

```
[[ 1 10 20 11]
 [14 12 13 14]]
```

Create Array From The Built-in Functions On The Numpy

- zeros((rows, columns))
- ones((rows, columns))
- full((rows. columns), fill_value)
- eye(shape)
- diag(array)
- arange(start, stop, step)
- linspace(start, step, stop, float_numbers, endpoint, retstep)
- random.rand(rows, columns)
- random.uniform(start, stop, size=(rows, columns))
- random.randint(start, stop, size=(rows, columns))
- random.normal(Mean, Standard_Deviation, size=(rows, columns))
- random.randn(rows, columns)
- random.choice(array, size=num)
- random.permutation(array)
- reshape(array, (rows, .columns))
- array.T
- delete(array, object, axis)
- append(array, value, axis)
- insert(array, index, value. axis)
- vstack((array_1, array_2))
- hstack((array_1, array_2))
- split(array, num_of_subarrays)
- isinf(array)
- isnan(array)
- concatenate((array_1, array_2))
- array_equal(array_1, array_2)
- equal(array_1, array_2)
- not_equal(array_1, array_2)
- copy(array)
- diag(array, k)
- uniqu(array)
- intersect1d(array_1, array_2)
- setdiff1d(array_1, array_2)
- union1d(array_1, array_2)
- sort(array)
- argsort(array)
- argmax(array)
- argmin(array)
- add(array_1, array_2)
- subtract(array_1, array_2)
- multiply(array_1, array_2)
- divide(array_1, array_2)
- sqrt(array)
- exp(array)
- power(array, exponent)
- mean(array)
- sum(array, axis)
- dot(array_1, array_2)
- linalg.det(array)
- linalg.inv(array)
- linalg.eig(var(array))
- median(array)
- std(array)
- max(array)
- min(array)
- corrcoef(array)
- cov(array)

Create Array Of Zeros Elements

- Default Data Type Of Array Elements Are float64

In [276…
```
array = np.zeros((5, 5))
print(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.]]
```

In [277…
```
array = np.zeros((5, 5), dtype="int")
print(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]]
```

Create Array Of Ones Elements

- Default Data Type Is Float64

In [278…
```
array = np.ones((5, 5))
print(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. 1.]]
```

Create New Array With The Same Shape And Type As a Given Array Filled With a Fill_value

- Syntax: full((rows, columns), fill_value)

In [279…
```
array = np.full((5, 5), 10)
print(array)
```

```
[[10 10 10 10 10]
 [10 10 10 10 10]
 [10 10 10 10 10]
 [10 10 10 10 10]
 [10 10 10 10 10]]
```

Creat Identity Matrix

- Syntax: eye(shape)
- Create A 2D Array With Ones On The Diagonal And Zeros Elsewhere
- This Is Very Important Matrix In Linear Algebra
- Disgonal Matrix Is Always Squar Matrix

In [280…
```
array = np.eye(5)
print(array)
```

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

Create A New ndarray With The Given 1D Array as Its Diagonal Elements

- Syntax: diag(array)
- Disgonal Matrix Is Always Squar Matrix

In [281…
```
array = np.diag([1, 2, 3, 4, 5])
print(array)
```

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

Create An Array Of Sequence Of Elements That Take Start And Stop Numbers

- Syntax: arange(start, stop, step)
- Start Can Be Discard
- Stop Is Neccessary Stop Is Not Included
- Step Is Optional

In [282…
```
array = np.arange(0, 10)
print(array)
```

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

In [283…
```
array = np.arange(10)
print(array)
```

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

Create To Create An Array With Evenly Spaced Elements Over An Interval

- Syntax: linspace(start, stop, num, endpoint, retstep, dtype)

In [284…
```
array = np.linspace(0, 10, 5, dtype=int)
print(array)
```

```
[ 0  2  5  7 10]
```

Print The Step Number In The linspace Method

In [285…
```
array = np.linspace(0, 100, 6, retstep=True)
print(array)
```

```
(array([  0.,  20.,  40.,  60.,  80., 100.]), 20.0)
```

Can Make The Stop ELement Included

In [286…
```
array = np.linspace(0, 100, 6, endpoint=True)
print(array)
```

```
[  0.  20.  40.  60.  80. 100.]
```

Can Determine The Number Of Floating Point Numbers But Make The retstep=True

In [287…
```
x = np.linspace(0, 10, num=4, retstep=True)
print(x)
```

```
(array([ 0.        ,  3.33333333,  6.66666667, 10.        ]), 3.3333333333333335)
```

Linspace Can Take Negative Number

In [288…
```
x = np.linspace(-2, 2, 5)
print(x)
```

```
[-2. -1.  0.  1.  2.]
```

Generate Random Floats From 0 To 1 From A Uniform Distribution

- Syntax: random.rand(rows, columns)
- Array Elements Are Interchangeable

In [289…
```
x = np.random.rand(3, 3)
print(x)
```

```
[[0.53222441 0.29579631 0.21714007]
 [0.73599611 0.5373174  0.93455911]
 [0.83750433 0.64050864 0.48521168]]
```

Array Elelements Are Interchangeable

In [290…
```
x = np.random.rand(3, 3)
print(x)
```

```
[[0.82559121 0.07670088 0.65138593]
 [0.6141081  0.03285109 0.0340378 ]
 [0.46957064 0.98754744 0.98107267]]
```

Generate A Random Floats From A Uniform Distribution

- Syntax: random.randint(start, stop, (rows, columns))
- This Must Determine The Start And Stop Numbers

x = np.random.uniform(0, 10, (5, 5)) print(x)

Generate Random Floats From 0 to 1

- Syntax: random.random(rows, columns)

In [291…
```
x = np.random.random((3, 3))
print(x)
```

```
[[0.92451795 0.88040046 0.44825995]
 [0.67115666 0.36016399 0.68579463]
 [0.30390124 0.93611185 0.41043922]]
```

Generate Random Integer Numbers

- Syntax: random.randint(start, stop, (rows, columns))

In [292…
```
x = np.random.randint(0, 5, (5, 5))
print(x)
```

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

Generate Only one Integer Number

- Syntax: random.randint(start, stop)

In [293…
```
x = np.random.randint(1, 11)
print(x)
```

```
2
```

In [294…
```
x = np.random.randint(1, 11)
print(x)
```

```
2
```

Generate 1D Array

- Syntax: random.randint(num)

In [295…
```
x = np.random.randint((1, 10))
print(x)
```

```
[0 9]
```

In [296…
```
print(x.size)
```

```
2
```

In [297…
```
x = np.random.randint((1, 10))
print(x)
```

```
[0 4]
```

In [298…
```
print(x.size)
```

```
2
```

Generate random Integer Number

- Syntax: random.randint(start, stop)

In [299…
```
x = np.random.randint(5, 10)
```

```
print(x)
```

```
6
```

Generate 2D Array With Random Integer Numbers

- Syntax: random.randint(start, stop, size(rows, columns))
- Size Refers To The Shape

In [300…
```
x = np.random.randint(0, 5, size=(3, 3))
print(x)
```

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

In [301…
```
x = np.random.randint(0, 5, size=(3, 3))
print(x)
```

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

Generate Random Integer Number With A Range

- Syntax: random.randint(start, stop, size)

In [302…
```
x = np.random.randint(0, 5, 10)
print(x)
```

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

Get The Normal Data Distribution

- Syntax: random.normal(Mean, Standard_Deviation)

In [303…
```
x = np.random.normal(0, 5, size=(5, 5))
print(x)
```

```
[[ 3.49119927  6.44884944  0.80494    -8.15059313  4.77451805]
 [-1.49762422  2.40043143  7.5378271  -2.86527926  4.36247434]
 [ 2.13241443 -5.8642209  -0.73042751 -2.96664283 -2.28328948]
 [ 6.86040112 -5.3912155   4.28134293  6.72989067  3.17880234]
 [ 7.63349497  1.83075177  1.18961284 -1.11948065 13.92851253]]
```

In [304…
```
x = np.random.normal(0, 5, size=(5, 5))
print(x)
```

```
[[ 1.65812551  9.38846975  7.55907658 10.87987171  1.30288571]
 [ 4.63335499  2.41956173 -3.73232232  1.28571986 -3.5451619 ]
 [-6.16150384  2.91990492 -6.75175558  7.89395677  0.64691019]
 [-3.55423303 -4.30518357  1.76983554 13.51985421 10.22294928]
 [ 8.58957126  1.4312731   6.32300691 -1.32395393 -4.37989077]]
```

Get The Standard Normal Distribution

- Syntax: random.randn(3, 3)

In [305…
```
x = np.random.randn(3, 3)
print(x)
```

```
[[ 1.11385997 -0.35686965 -0.5890456 ]
 [-0.6810654  -0.75063017 -1.05170806]
 [ 0.38650518 -0.77848301 -0.89780905]]
```

Select Random Element From Array

- Syntax: random.choice(array, size)

In [306…
```
array = np.arange(20)
print(array)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

In [307…
```
x = np.random.choice(array, size=3)
print(x)
```

```
[ 7  4 13]
```

In [308…
```
x = np.random.choice(array, size=3)
print(x)
```

```
[ 6  9 11]
```

Select Only One Element If Not Determine The Size

In [309…
```
x = np.random.choice(array)
print(x)
```

```
2
```

Creating A Random Permutation Of An Array

- The numbers Can Be Interchangeable
- Syntax: random.permutation(array)

In [310…
```
array = np.arange(10)
print(array)
print('='*44)
x = np.random.permutation(array)
print(x)
```

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

In [311…
```
x = np.random.permutation(array)
print(x)
```

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

In [312…
```
x = np.random.permutation(array)
print(x)
```

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

Reshape The Array

In [313…
```
array = np.arange(25)
print(array)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24]
```

In [314…
```
x = np.reshape(array, (5, 5))
print(x)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

In [315…
```
array = np.arange(25).reshape(5, 5)
print(array)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

Transpose The Array

- Convert The Row To Column
- Convert The Column To Row
- Syntax: array.T

array = np.arange(15).reshape(3, 5) print(array)

Get The Shape Of The Array Before Transpose

print(f"The Shape Of The Array Is: {array.shape}")

In [316…
```
x = array.T
print(x)
# Get The Shape Of The Array After Transpose
```

```
print(f"The Shape Of The Array Is: {x.shape}")
```

```
[[ 0  5 10 15 20]
 [ 1  6 11 16 21]
 [ 2  7 12 17 22]
 [ 3  8 13 18 23]
 [ 4  9 14 19 24]]
The Shape Of The Array Is: (5, 5)
```

Delete Value On Array

- Syntax: delete(array, index, axis)
- axis = 1, Refers To The Columns
- axis = 0, Refers To The Rows

In [317…
```
array = np.arange(20).reshape(4, 5)
print(array)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

In [318…
```
# Delete The First Row
x = np.delete(array, 0, axis=0)
print(x)
```

```
[[ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

In [319…
```
# Delete The First Column
x = np.delete(array, 0, axis=1)
print(x)
```

```
[[ 1  2  3  4]
 [ 6  7  8  9]
 [11 12 13 14]
 [16 17 18 19]]
```

In [320…
```
# Remove The First & Third Column
x = np.delete(array, [0, 2], axis=1)
print(x)
```

```
[[ 1  3  4]
 [ 6  8  9]
 [11 13 14]
 [16 18 19]]
```

Add Element To Array

- Syntax: append(array, value)
- Append Used To Add Elements At The End Of The List

In [321…
```
array = np.arange(0, 5)
print(array)
```

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

In [322…
```
x = np.append(array, 10)
print(x)
```

```
[ 0  1  2  3  4 10]
```

Append in 2D Array

- Note That: New Values = New Values

In [323…
```
array = np.arange(9).reshape(3, 3)
print(array)
```

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

In [324…
```
print(f"The Shape Of The Array: {array.shape}")
```

```
The Shape Of The Array: (3, 3)
```

In [325…
```
x = np.append(array, [[22], [33], [44]], axis=1)
print(x)
```

```
[[ 0  1  2 22]
 [ 3  4  5 33]
 [ 6  7  8 44]]
```

Insert In 1D Array

- Syntax: insert(array, index, value)

In [326…
```
array = np.arange(10)
print(array)
```

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

In [327…
```
x = np.insert(array, 5, [5, 5, 5, 5])
print(x)
```

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

Insert In 2D Array

- Syntax: insert(array, index, value, axis)

In [328…
```
array = np.arange(20).reshape(4, 5)
print(array)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

In [329…
```
values = np.array([0, 0, 0, 0, 0], dtype="int")
# Insert Values On The First Row Of Array
x = np.insert(array, 0, values, axis=0)
print(x)
```

```
[[ 0  0  0  0  0]
 [ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

Add Array To Each Other vertically

- Synatx: vstack((array_1, array_2))
- Means vertical Stack

In [330…
```
# Create 1D Array
array_1 = np.arange(5)
# Create 2D Array
array_2 = np.arange(25).reshape(5, 5)

# Add Array_1 To Array_2 vertically
v = np.vstack((array_1, array_2))
print(v)
```

```
[[ 0  1  2  3  4]
 [ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

In [331…
```
# Add Array_2 To Array_1 Vertically
v = np.vstack((array_2, array_1))
print(v)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [ 0  1  2  3  4]]
```

Add Arrays to Each Other horizontally

- Syntax: hstack((array_1, array_2))
- Means Horizontal Stack

In [332…
```
array_1 = np.arange(5).reshape(5, 1)
```

```
array_2 = np.arange(25).reshape(5, 5)
```

In [333…
```
# Add Array_1 To Array_2 Horizontally
v = np.hstack((array_1, array_2))
print(v)
```

```
[[ 0  0  1  2  3  4]
 [ 1  5  6  7  8  9]
 [ 2 10 11 12 13 14]
 [ 3 15 16 17 18 19]
 [ 4 20 21 22 23 24]]
```

Split The Array Into Subarrays

- Syntax: split(array, num_of_subarrays)

In [334…
```
array = np.arange(1, 10).reshape(3, 3)
print(array)
```

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

Split The Array Into num Of Subarrays

In [335…
```
x = np.split(array, 3)
print(x)
```

```
[array([[1, 2, 3]]), array([[4, 5, 6]]), array([[7, 8, 9]])]
```

Check For Positive Or Negative Infinity

- Return True If Exist Infinity Value
- Return False If Not Exist Infinity Value

In [336…
```
array = np.array([2, 3, 5, np.infty, -np.infty, np.nan])
print(array)
```

```
[  2.   3.   5.  inf -inf  nan]
```

In [337…
```
is_inf_array = np.isinf(array)
print(is_inf_array)
```

```
[False False False  True  True False]
```

Check Each Element In The Array Is Nan Or Not

- Return True If Exist Nan Element
- Return False If Not Exist Nan Element

In [338…
```
array = np.array([1, 12, np.nan, 5])
is_nan_element = np.isnan(array)
print(is_nan_element)
```

```
[False False  True False]
```

Concatenating Arrays

- Syntax: concatenate((array_1, array_2))

In [339…
```
array_1 = np.arange(10)
array_2 = np.arange(10, 20)
concatenated_arrays = np.concatenate((array_1, array_2))
print(concatenated_arrays)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

Check If The Array Elements Are Equal To Each Other Or Not

- Check Each Element On The Array
- Return True If The Two Arrays Are Equal To Each Other
- Return False If The Two Arrays Are Not Equal To Each Other
- Syntax: array_equal(array_1, array_2)

In [340…
```
array_1 = np.arange(10)
```

```
array_2 = np.arange(10, 20)
# Check The Arrays
check_arrays = np.array_equal(array_1, array_2)
print(check_arrays)
```

```
False
```

In [341…
```
check_arrays = np.array_equal([1, 2, 3], [1, 2, 3])
print(check_arrays)
```

```
True
```

In [342…
```
check_arrays = np.array_equal([1, 2, 3], [3, 2, 1])
print(check_arrays)
```

```
False
```

Check The Array Element Are Equal To Each Other Or Not

- Check Each Element On The Array
- Return The Result On Array Of Boolean Values
- Return True If The Elements Are Equal
- Return False If The Array Elements Are Not Equal
- Synatx: equal(array_1, array_2)

In [343…
```
check = np.equal([1, 2, 3], [1, 2, 3])
print(check)
```

```
[ True  True  True]
```

In [344…
```
check = np.equal([1, 2], [2, 1])
print(check)
```

```
[False False]
```

In [345…
```
check = np.equal([1, 2, 3], [1, 2, 6])
print(check)
```

```
[ True  True False]
```

Check The Array Is Not Equal To Each Other Or Not

- Return True If Elements Are Not Equal
- Return False If The Elements Are Eqaul
- Syntax: not_equal(array_1, array_2)

In [346…
```
check = np.not_equal([1, 2, 3], [1, 2, 3])
print(check)
```

```
[False False False]
```

In [347…
```
check = np.not_equal([1, 2], [1, 3])
print(check)
```

```
[False  True]
```

Indexing & Slicing

- Used For Pre-processing Before Enter The Data Into A Model
- Slicing Means Taking Elements From One Given Index To Another Given Index
- Syntax Of Slicing: array[start : stop : step]

In [348…
```
array = np.arange(20).reshape(4, 5)
print(array)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

Indexing

- Return The Value Of Specific Index
- Syntax: array[row, column]

In [349…
```
print(array[1, 2])
```

```
7
```

In [350…
```
print(array[0, 0])
```

```
0
```

Get The Last Index On Array On The Raw And The Column

In [351…
```
print(array[-1, -1])
```

```
19
```

Another Way Of Indexing

In [352…
```
print(array[1] [1])
```

```
6
```

Update The Array Element

In [353…
```
array[0] [0] = 10
print(array)
```

```
[[10  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

Slicing

In [354…
```
# Print The First two Rows
x = array[0:2, :]
print(x)
```

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

In [355…
```
# Print The Last Row
x = array[-1:, :]
print(x)
```

```
[[15 16 17 18 19]]
```

In [356…
```
# Get Only The Last Element On Array
x = array[-1:, -1:]
print(x)
```

```
[[19]]
```

Copy Array In Numpy

- Syntax: copy(array)

In [357…
```
array = np.arange(10)
x = np.copy(array)
print(x)
```

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

Copy Array In Python

In [358…
```
array = np.arange(10)
x = array[:].copy()
print(x)
```

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

Extract The Diagonal Elements

- Syntax: diag(array, k)

In [359…
```
array = np.arange(20).reshape(4, 5)
print(array)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

In [360…
```
x = np.diag(array)
print(x)
```

```
[ 0  6 12 18]
```

Get The Elements That Exist On The Top Of The Diagonal Radius

- Syntax: diag(array, k=1)

In [361…
```
x = np.diag(array, k=1)
print(x)
```

```
[ 1  7 13 19]
```

Get The Elements That Exist On The Bottom Of The Diagonal Radius

- Syntax: diag(array, k=-1)

In [362…
```
x = np.diag(array, k=-1)
print(x)
```

```
[ 5 11 17]
```

Get The Elements On Array, Without Repeat Elements

- Syntax: unique(array)

In [363…
```
# Get The Elements On Array, Without Repeat Elements
x = np.array([[1, 2, 3], [1, 2, 4,], [4, 5, 6]])
print(np.unique(x))
```

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

Slicing Using Comparision Operators

In [364…
```
x = np.arange(25).reshape(5, 5)
print(x)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

Print All Values That Bigger Than 10 As Boolean Values

In [365…
```
print(x < 10)
```

```
[[ True  True  True  True  True]
 [ True  True  True  True  True]
 [False False False False False]
 [False False False False False]
 [False False False False False]]
```

Print All Values That Smaller Than 10

In [366…
```
print(x[x < 10])
```

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

Print Elements That Bigger Than 10, And Smaller Than 17

In [367…
```
print(x[ (x > 10) & ( x < 17 ) ])
```

```
[11 12 13 14 15 16]
```

Change The Values Of Elements That Meet The Condition

In [368…
```
change_values = x[ (x > 10) & ( x < 17 ) ] = -1
```

In [369…
```
print(change_values)
```

```
-1
```

In [370…
```
print(x)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 -1 -1 -1 -1]
 [-1 -1 17 18 19]
 [20 21 22 23 24]]
```

Get The Intersection Elements

- Get The Elements That Exist On The First Array And Exist On The Second Array
- In Another Way Finding The Common Values That Exist On The Two Arrays
- Syntax: intersect1d(array_1, array_2)

In [371…
```
x = np.arange(1, 6)
y = np.arange(4, 9)
# Get The Intersection Elements
z = np.intersect1d(x, y)
print(z)
```

```
[4 5]
```

Get The Difference Elements That Exist On Two Arrays

- Get The ELements That Exist On The First Array And Not Exist On The Second Array
- The Arangement Of Arrays Is important
- Syntax: setdiff1d(array_1, array_2)

In [372…
```
x = np.arange(1, 6)
y = np.arange(4, 9)
# Get The Difference Elements That Exist On Two Arrays
z = np.setdiff1d(x, y)
print(z)
```

```
[1 2 3]
```

In [373…
```
# Get The Difference Elements That Exist On Two Arrays
z = np.setdiff1d(y, x)
print(z)
```

```
[6 7 8]
```

Get The Union ELements

- Get The Elements That On First Array And Elements That Exist On The Second Array
- Syntax: union1d(array_1, array_2)

In [374…
```
x = np.arange(1, 6)
y = np.arange(4, 9)
# Get The Difference Elements That Exist On Two Arrays
z = np.union1d(x, y)
print(z)
```

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

Sorting Arrays

In [375…
```
x = np.array([1, 3, 6, 2, 9, 10])
print(x)
```

```
[ 1  3  6  2  9 10]
```

Sorting The Array Elements Ascending

- Sort The Elements From Smaller value To Big Value

In [376…
```
sorted_array = np.sort(x)
print(sorted_array)
```

```
[ 1  2  3  6  9 10]
```

Sort The Array Elements Descending

In [377…
```
sorted_array = np.sort(x)[::-1]
print(sorted_array)
```

```
[10  9  6  3  2  1]
```

Write The Index Of The Elements On The Array Using argsort() Method

- Sorts The Array Elements In Ascending Order And Returns Indicies Of The Sorted Elements

In [378…
```
x = np.array([1, 0, 3, 5, 4])
print(x)
```
```
[1 0 3 5 4]
```

In [379…
```
indicies_of_sort = np.argsort(x)
print(indicies_of_sort)
```
```
[1 0 2 4 3]
```

Create Random Array For Sort It

In [381…
```
array = np.random.randint(1, 21, size=(5, 4))
print(array)
```
```
[[10  7 13  8]
 [ 4  8 20 17]
 [ 3 12 17 14]
 [10 18  1 10]
 [17 13 13 14]]
```

Sort The ELements That Exist On The First Column

In [383…
```
sorted_array = array[array[:, 0].argsort()]
print(sorted_array)
```
```
[[ 3 12 17 14]
 [ 4  8 20 17]
 [10  7 13  8]
 [10 18  1 10]
 [17 13 13 14]]
```

Sort Elements On Array

- axis=0: Refers To The Column
- axis=1: Refers To The Row

In [396…
```
sorted_array = np.sort(array, axis=1)
print(sorted_array)
```
```
[[ 7  8 10 13]
 [ 4  8 17 20]
 [ 3 12 14 17]
 [ 1 10 10 18]
 [13 13 14 17]]
```

In [397…
```
sorted_array = np.sort(array, axis=0)
print(sorted_array)
```
```
[[ 3  7  1  8]
 [ 4  8 13 10]
 [10 12 13 14]
 [10 13 17 14]
 [17 18 20 17]]
```

Searching

In [402…
```
array = np.arange(1, 26).reshape(5, 5)
print(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]]
```

Get The Index Of Maximum Value

In [403…
```
max_index = np.argmax(array)
print(f"The Index Of Maximum Value Is: {max_index}")
```
```
The Index Of Maximum Value Is: 24
```

Get The Index Of Minimum Value

In [405…
```
min_index = np.argmin(array)
print(f"The Index Of Minimum Value Is: {min_index}")
```

```
The Index Of Minimum Value Is: 0
```

In [406…
```
# Get The Index Of Minimum Value
print(np.argmin([1, 2, 3, 0]))
```

```
3
```

In [407…
```
# Get The Index Of maximum Value
print(np.argmax([0, 1, 6, 10, 3, 2]))
```

```
3
```

Mathematical Operators

Add Two Arrays

- Add Array One To The Array Two

In [411…
```
added_array = np.add([1, 2, 3], [4, 5, 6])
print(added_array)
```

```
[5 7 9]
```

Subtract two Arrays

- Sub Array One From The Array two

In [413…
```
sub_array = np.subtract([10, 9, 8], [7, 6 ,5])
print(sub_array)
```

```
[3 3 3]
```

Multiply Arrays

- multiply Array One To The Array Two

In [415…
```
mul_array = np.multiply([10, 9, 8], [7, 6 ,5])
print(mul_array)
```

```
[70 54 40]
```

Divide Arrays

- Divide Array_1 By The Array_2

In [419…
```
div_arrays = np.divide([10, 8, 6], [2, 2, 2])
print(div_arrays)
```

```
[5. 4. 3.]
```

Get The Square Root To The Array Elements

In [420…
```
s = np.sqrt([16, 9, 1])
print(s)
```

```
[4. 3. 1.]
```

Get The Exponential Value Of The Array Elements

In [422…
```
ex = np.exp([10, 12, 5])
print(ex)
```

```
[2.20264658e+04 1.62754791e+05 1.48413159e+02]
```

Make Power To The Array

In [423…
```
p = np.power([2, 3, 4, 5], 3)
print(p)
```

```
[  8  27  64 125]
```

Get The Mean Of The Array Elements

In [424…
```
array = np.arange(20).reshape(4, 5)
```

```
print(array)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

Get The Mean

- Get The Average Of The All Elements On The Array

m = np.mean(array) print(m)

Get the Mean To the Columns

- axis=0: Refers To The Columns

In [430…
```
avg = np.mean(array, axis=0)
print(avg)
```

```
[ 7.5  8.5  9.5 10.5 11.5]
```

Get The Mean To The Rows

- axis=1: Refers To The Rows

In [431…
```
average = np.mean(array, axis=1)
```

Sum All Elements On Array

In [432…
```
s = np.sum(array)
print(s)
```

```
190
```

Sum Of Columns on Array

In [436…
```
# Print The Shape Of The Array
# Shape: Refers To The Number Of Rows & Columns
print(array.shape)
```

```
(4, 5)
```

In [433…
```
s = np.sum(array, axis=0)
print(s)
```

```
[30 34 38 42 46]
```

Sum Of Rows On Array

In [434…
```
s = np.sum(array, axis=1)
print(s)
```

```
[10 35 60 85]
```

Multiply Array With A Specific Number

In [438…
```
mul = np.multiply(array, 2)
print(mul)
```

```
[[ 0  2  4  6  8]
 [10 12 14 16 18]
 [20 22 24 26 28]
 [30 32 34 36 38]]
```

Divide Array With A Specific Number

In [441…
```
div = np.divide(array, 0.5)
print(div)
```

```
[[ 0.  2.  4.  6.  8.]
 [10. 12. 14. 16. 18.]
 [20. 22. 24. 26. 28.]
 [30. 32. 34. 36. 38.]]
```

Make Dot Product to The Array

In [443…
```
x = np.arange(9).reshape(3, 3)
y = np.arange(9, 18).reshape(3, 3)
dot_product = np.dot(x, y)
```

```
print(dot_product)
```

```
[[ 42  45  48]
 [150 162 174]
 [258 279 300]]
```

Multiply Two Matrices

- Make Matrix Maultiplication

In [444…
```
mul = np.matmul(x, y)
print(mul)
```

```
[[ 42  45  48]
 [150 162 174]
 [258 279 300]]
```

Get The Determinant

- linalg: Refers To The Linear Algebra

In [448…
```
x = np.arange(16).reshape(4, 4)
det = np.linalg.det(x)
print(det)
```

```
-2.9582283945787796e-30
```

Get The Matrix Inversion

In [449…
```
inv = np.linalg.inv(x)
print(inv)
```

```
[[ 9.00719925e+14 -4.50359963e+14 -1.80143985e+15  1.35107989e+15]
 [-2.40191980e+15  2.70215978e+15  1.80143985e+15 -2.10167983e+15]
 [ 2.10167983e+15 -4.05323966e+15  1.80143985e+15  1.50119988e+14]
 [-6.00479950e+14  1.80143985e+15 -1.80143985e+15  6.00479950e+14]]
```

Get The Variance Of Array

In [450…
```
print(np.var(x))
```

```
21.25
```

Get The Median of Array

In [451…
```
print(np.median(x))
```

```
7.5
```

Get The Standard Deviation

In [452…
```
print(np.std(x))
```

```
4.6097722286464435
```

In [453…
```
# Get The minimum Value
print(np.min(x))
```

```
0
```

In [454…
```
# Get The Maximum Value
print(np.max(x))
```

```
15
```

Get The Correlation Matrix

In [456…
```
print(np.corrcoef(x))
```

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

Get The Covariance Matrix

In [457…
```
print(np.cov(x))
```

```
[[1.66666667 1.66666667 1.66666667 1.66666667]
 [1.66666667 1.66666667 1.66666667 1.66666667]
 [1.66666667 1.66666667 1.66666667 1.66666667]
 [1.66666667 1.66666667 1.66666667 1.66666667]]
```