Numpy Basics

In [67]:

```
# 1. Write a Numpy program to get the Numpy version and show the Numpy build configuration.
import numpy as np
print(np.__version__)
print(np.show_config())
```

```
1.26.3
{
  "Compilers": {
    "c": {
      "name": "msvc",
      "linker": "link",
      "version": "19.29.30153",
      "commands": "cl"
    },
    "cython": {
      "name": "cython",
      "linker": "cython",
      "version": "3.0.7",
      "commands": "cython"
    },
    "c++": {
      "name": "msvc",
      "linker": "link",
      "version": "19.29.30153",
      "commands": "cl"
    }
  },
  "Machine Information": {
    "host": {
      "cpu": "x86_64",
      "family": "x86_64",
      "endian": "little",
      "system": "windows"
    },
    "build": {
      "cpu": "x86_64",
      "family": "x86_64",
      "endian": "little",
      "system": "windows"
    }
  },
  "Build Dependencies": {
    "blas": {
      "name": "openblas64",
      "found": true,
      "version": "0.3.23.dev",
      "detection method": "pkgconfig",
      "include directory": "/c/opt/64/include",
      "lib directory": "/c/opt/64/lib",
      "openblas configuration": "USE_64BITINT=1 DYNAMIC_ARCH=1 DYNAMIC_OLDER= NO_CBLAS= NO_LAPACK= NO_LAPACKE= NO_AFFINITY=
1 USE_OPENMP= SKYLAKEX MAX_THREADS=2",
      "pc file directory": "C:/opt/64/lib/pkgconfig"
    },
    "lapack": {
      "name": "dep2628220032720",
      "found": true,
      "version": "1.26.3",
      "detection method": "internal",
      "include directory": "unknown",
      "lib directory": "unknown",
      "openblas configuration": "unknown",
      "pc file directory": "unknown"
    }
  },
  "Python Information": {
    "path": "C:\\Users\\runneradmin\\AppData\\Local\\Temp\\cibw-run-frm2g0yv\\cp311-win_amd64\\build\\venv\\Scripts\\pytho
n.exe",
    "version": "3.11"
  },
  "SIMD Extensions": {
    "baseline": [
      "SSE",
      "SSE2",
      "SSE3"
    ],
    "found": [
      "SSSE3",
      "SSE41",
      "POPCNT",
      "SSE42",
      "AVX",
      "F16C",
      "FMA3",
      "AVX2",
      "AVX512F",
      "AVX512CD",
      "AVX512_SKX",
      "AVX512_CLX",
      "AVX512_CNL",
      "AVX512_ICL"
    ]
  }
}
None
```

```
c:\Users\presh\AppData\Local\Programs\Python\Python311\Lib\site-packages\numpy\__config__.py:149: UserWarning: Install `pyy
aml` for better output
  warnings.warn("Install `pyyaml` for better output", stacklevel=1)
```

In [68]:

```
# 2. Write a NumPy program to get help with the add function.
import numpy as np
print(np.info(np.add))
```

```
add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj])

Add arguments element-wise.
```

```
Parameters
----------
x1, x2 : array_like
    The arrays to be added.
    If ``x1.shape != x2.shape``, they must be broadcastable to a common
    shape (which becomes the shape of the output).
out : ndarray, None, or tuple of ndarray and None, optional
    A location into which the result is stored. If provided, it must have
    a shape that the inputs broadcast to. If not provided or None,
    a freshly-allocated array is returned. A tuple (possible only as a
    keyword argument) must have length equal to the number of outputs.
where : array_like, optional
    This condition is broadcast over the input. At locations where the
    condition is True, the `out` array will be set to the ufunc result.
    Elsewhere, the `out` array will retain its original value.
    Note that if an uninitialized `out` array is created via the default
    ``out=None``, locations within it where the condition is False will
    remain uninitialized.
**kwargs
    For other keyword-only arguments, see the
    :ref:`ufunc docs <ufuncs.kwargs>`.

Returns
-------
add : ndarray or scalar
    The sum of `x1` and `x2`, element-wise.
    This is a scalar if both `x1` and `x2` are scalars.

Notes
-----
Equivalent to `x1` + `x2` in terms of array broadcasting.

Examples
--------
>>> np.add(1.0, 4.0)
5.0
>>> x1 = np.arange(9.0).reshape((3, 3))
>>> x2 = np.arange(3.0)
>>> np.add(x1, x2)
array([[  0.,   2.,   4.],
       [  3.,   5.,   7.],
       [  6.,   8.,  10.]])

The ``+`` operator can be used as a shorthand for ``np.add`` on ndarrays.

>>> x1 = np.arange(9.0).reshape((3, 3))
>>> x2 = np.arange(3.0)
>>> x1 + x2
array([[ 0.,  2.,  4.],
       [ 3.,  5.,  7.],
       [ 6.,  8., 10.]])
None
```

In [69]:

```
# 3. Write a NumPy program to test whether none of the elements of a given array are zero.

import numpy as np
x = np.array([1, 2, 3, 4])
print("Original array:")
print(x)
print("Test if none of the elements of the said array is zero:")
print(np.all(x))
x = np.array([0, 1, 2, 3])
print("Original array:")
print(x)
print("Test if none of the elements of the said array is zero:")
print(np.all(x))
```

```
Original array:
[1 2 3 4]
Test if none of the elements of the said array is zero:
True
Original array:
[0 1 2 3]
Test if none of the elements of the said array is zero:
False
```

In [70]:

```
# 4. Write a NumPy program to test if any of the elements of a given array are non-zero.
import numpy as np
x = np.array([1, 0, 0, 0])
print("Original array:")
print(x)
print("Test whether any of the elements of a given array is non-zero:")
print(np.any(x))
x = np.array([0, 0, 0, 0])
print("Original array:")
print(x)
print("Test whether any of the elements of a given array is non-zero:")
print(np.any(x))
```

```
Original array:
[1 0 0 0]
Test whether any of the elements of a given array is non-zero:
True
Original array:
[0 0 0 0]
Test whether any of the elements of a given array is non-zero:
False
```

In [71]:

```
# 5. Write a NumPy program to test a given array element-wise for finiteness (not infinity or not a number).
import numpy as np
a = np.array([1, 0, np.nan, np.inf])
print("Original array")
print(a)
print("Test a given array element-wise for finiteness :")
print(np.isfinite(a))
```

```
Original array
[ 1.   0. nan inf]
Test a given array element-wise for finiteness :
```

```
Test a given array element-wise for finiteness :
[ True  True False False]
```

In [72]:

```
#  6. Write a NumPy program to test elements-wise for positive or negative infinity.
import numpy as np
a = np.array([1, 0, np.nan, np.inf])
print("Original array")
print(a)
print("Test element-wise for positive or negative infinity:")
print(np.isinf(a))
```

```
Original array
[ 1.  0. nan inf]
Test element-wise for positive or negative infinity:
[False False False  True]
```

In [73]:

```
# 7. Write a NumPy program to test element-wise for NaN of a given array.
import numpy as np
a = np.array([1, 0, np.nan, np.inf])
print("Original array")
print(a)
print("Test element-wise for NaN:")
print(np.isnan(a))
```

```
Original array
[ 1.  0. nan inf]
Test element-wise for NaN:
[False False  True False]
```

In [74]:

```
# 8. Write a NumPy program to test element-wise for complex numbers, real numbers in a given array. Also test if a given r
import numpy as np
a = np.array([1+1j, 1+0j, 4.5, 3, 2, 2j])
print("Original array")
print(a)
print("Checking for complex number:")
print(np.iscomplex(a))
print("Checking for real number:")
print(np.isreal(a))
print("Checking for scalar type:")
print(np.isscalar(3.1))
print(np.isscalar([3.1]))
```

```
Original array
[1. +1.j 1. +0.j 4.5+0.j 3. +0.j 2. +0.j 0. +2.j]
Checking for complex number:
[ True False False False False  True]
Checking for real number:
[False  True  True  True  True False]
Checking for scalar type:
True
False
```

In [75]:

```
# 9. Write a NumPy program to test whether two arrays are element-wise equal within a tolerance.
import numpy as np
print("Test if two arrays are element-wise equal within a tolerance:")
print(np.allclose([1e10,1e-7], [1.00001e10,1e-8]))
print(np.allclose([1e10,1e-8], [1.00001e10,1e-9]))
print(np.allclose([1e10,1e-8], [1.0001e10,1e-9]))
print(np.allclose([1.0, np.nan], [1.0, np.nan]))
print(np.allclose([1.0, np.nan], [1.0, np.nan], equal_nan=True))
```

```
Test if two arrays are element-wise equal within a tolerance:
False
True
False
False
True
```

In [76]:

```
# 10. Write a NumPy program to create an element-wise comparison (greater, greater_equal, less and less_equal) of two give
import numpy as np
x = np.array([3, 5])
y = np.array([2, 5])
print("Original numbers:")
print(x)
print(y)
print("Comparison - greater")
print(np.greater(x, y))
print("Comparison - greater_equal")
print(np.greater_equal(x, y))
print("Comparison - less")
print(np.less(x, y))
print("Comparison - less_equal")
print(np.less_equal(x, y))
```

```
Original numbers:
[3 5]
[2 5]
Comparison - greater
[ True False]
Comparison - greater_equal
[ True  True]
Comparison - less
[False False]
Comparison - less_equal
[False  True]
```

In [77]:

```
# 11. Write a NumPy program to create an element-wise comparison (equal, equal within a tolerance) of two given arrays.
import numpy as np
x = np.array([72, 79, 85, 90, 150, -135, 120, -10, 60, 100])
y = np.array([72, 79, 85, 90, 150, -135, 120, -10, 60, 100.000001])
print("Original numbers:")
print(x)
print(y)
print("Comparison - equal:")
print(np.equal(x, y))
print("Comparison - equal within a tolerance:")
print(np.allclose(x, y))
```

```
Original numbers:
[  72   79   85   90  150 -135  120  -10   60  100]
[  72.          79.          85.          90.         150.        -135.
  120.         -10.          60.         100.000001]
Comparison - equal:
[ True  True  True  True  True  True  True  True  True False]
Comparison - equal within a tolerance:
True
```

In [78]:

```python
# 12. Write a NumPy program to create an array with the values 1, 7, 13, 105 and determine the size of the memory occupie
import numpy as np
X = np.array([1, 7, 13, 105])
print("Original array:")
print(X)
print("Size of the memory occupied by the said array:")
print("%d bytes" % (X.size * X.itemsize))
```

```
Original array:
[  1   7  13 105]
Size of the memory occupied by the said array:
16 bytes
```

In [79]:

```python
# 13. Write a NumPy program to create an array of 10 zeros, 10 ones, and 10 fives.
import numpy as np
array = np.zeros(10)
print("An array of 10 zeros:")
print(array)
array = np.ones(10)
print("An array of 10 ones:")
print(array)
array = np.ones(10) * 5
print("An array of 10 fives:")
print(array)
```

```
An array of 10 zeros:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
An array of 10 ones:
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
An array of 10 fives:
[5. 5. 5. 5. 5. 5. 5. 5. 5. 5.]
```

In [80]:

```python
# 14. Write a NumPy program to create an array of integers from 30 to 70.
import numpy as np
array = np.arange(30, 71)
print("Array of the integers from 30 to 70")
print(array)
```

```
Array of the integers from 30 to 70
[30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70]
```

In [81]:

```python
# 15.Write a NumPy program to create an array of all even integers from 30 to 70.
import numpy as np
array = np.arange(30, 71, 2)
print("Array of all the even integers from 30 to 70")
print(array)
```

```
Array of all the even integers from 30 to 70
[30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70]
```

In [82]:

```python
# 16. Write a NumPy program to create a 3x3 identity matrix.
import numpy as np
array_2D = np.identity(3)
print('3x3 matrix:')
print(array_2D)
```

```
3x3 matrix:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [83]:

```python
# 17. Write a NumPy program to generate a random number between 0 and 1.
import numpy as np
rand_num = np.random.normal(0, 1, 1)
print("Random number between 0 and 1:")
print(rand_num)
```

```
Random number between 0 and 1:
[1.251707823079436]
```

In [84]:

```python
# 18. Write a NumPy program to generate an array of 15 random numbers from a standard normal distribution.
import numpy as np
rand_num = np.random.normal(0, 1, 15)
print("15 random numbers from a standard normal distribution:")
print(rand_num)
```

```
15 random numbers from a standard normal distribution:
[-1.02567365808793  -1.342084944306678  1.522164815397617
 -0.21452092092714   0.857198512656195  1.08328770302373
 -0.465458029172515  0.538568596236211 -0.527482249039686
  0.463672212042426 -1.798180355462621  0.591782788633801
  1.174072949514046  1.159350301663237 -0.089150847411426]
```

In [85]:

```python
# 19. Write a NumPy program to create a vector with values ranging from 15 to 55 and print all values except the first an
import numpy as np
v = np.arange(15, 55)
print("Original vector:")
print(v)
print("All values except the first and last of the said vector:")
print(v[1:-1])
```

```
Original vector:
[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 50 51 52 53 54]
All values except the first and last of the said vector:
[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 50 51 52 53]
```

In [86]:

```
# 20. Write a NumPy program to create a 3X4 array and iterate over it.
import numpy as np
a = np.arange(10, 22).reshape((3, 4))
print("Original array:")
print(a)
print("Each element of the array is:")
for x in np.nditer(a):
    print(x, end=" ")
```

```
Original array:
[[10 11 12 13]
 [14 15 16 17]
 [18 19 20 21]]
Each element of the array is:
10 11 12 13 14 15 16 17 18 19 20 21
```

In [87]:

```
# 21. Write a NumPy program to create a vector of length 10 with values evenly distributed between 5 and 50.
import numpy as np
v = np.linspace(10, 49, 5)
print("Length 10 with values evenly distributed between 5 and 50:")
print(v)
```

```
Length 10 with values evenly distributed between 5 and 50:
[10.   19.75 29.5  39.25 49.  ]
```

In [88]:

```
# 22. Write a NumPy program to create a vector with values from 0 to 20 and change the sign of the numbers in the range fr
import numpy as np
x = np.arange(21)
print("Original vector:")
print(x)
x[(x >= 9) & (x <= 15)] *= -1
print("After changing the sign of the numbers in the range from 9 to 15:")
print(x)
```

```
Original vector:
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]
After changing the sign of the numbers in the range from 9 to 15:
[  0   1   2   3   4   5   6   7   8  -9 -10 -11 -12 -13 -14 -15  16  17
  18  19  20]
```

In [89]:

```
# 23. Write a NumPy program to create a vector of length 5 filled with arbitrary integers from 0 to 10.
import numpy as np
x = np.random.randint(0, 11, 5)
print("Vector of length 5 filled with arbitrary integers from 0 to 10:")
print(x)
```

```
Vector of length 5 filled with arbitrary integers from 0 to 10:
[5 2 9 6 1]
```

In [90]:

```
# 24. Write a NumPy program to multiply the values of two given vectors.
import numpy as np
x = np.array([1, 8, 3, 5])
print("Vector-1")
print(x)
y = np.random.randint(0, 11, 4)
print("Vector-2")
print(y)
result = x * y
print("Multiply the values of two said vectors:")
print(result)
```

```
Vector-1
[1 8 3 5]
Vector-2
[8 1 0 8]
Multiply the values of two said vectors:
[ 8  8  0 40]
```

In [91]:

```
# 25. Write a NumPy program to create a 3x4 matrix filled with values from 10 to 21.
import numpy as np
m = np.arange(10, 22).reshape((3, 4))
print(m)
```

```
[[10 11 12 13]
 [14 15 16 17]
 [18 19 20 21]]
```

In [92]:

```
# 26. Write a NumPy program to find the number of rows and columns in a given matrix.
import numpy as np
m = np.arange(10, 22).reshape((3, 4))
print("Original matrix:")
print(m)
print("Number of rows and columns of the said matrix:")
print(m.shape)
```

```
Original matrix:
[[10 11 12 13]
 [14 15 16 17]
 [18 19 20 21]]
Number of rows and columns of the said matrix:
(3, 4)
```

In [93]:

```
# 27. Write a NumPy program to create a 3x3 identity matrix, i.e. the diagonal elements are 1, the rest are 0.
import numpy as np
x = np.eye(3)
print(x)
```

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

In [94]:

```
# 28. Write a NumPy program to create a 10x10 matrix, in which the elements on the borders will be equal to 1, and inside
import numpy as np
x = np.ones((10, 10))
```

```
x[1:-1, 1:-1] = 0
print(x)
```

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

In [95]:

```
# 29. Write a NumPy program to create a 5x5 zero matrix with elements on the main diagonal equal to 1, 2, 3, 4, 5.
import numpy as np
x = np.diag([1, 2, 3, 4, 5])
print(x)
```

```
[[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]]
```

In [96]:

```
# 30. Write a NumPy program to create a 4x4 matrix in which 0 and 1 are staggered, with zeros on the main diagonal.
import numpy as np
x = np.zeros((4, 4))
x[::2, 1::2] = 1
x[1::2, ::2] = 1
print(x)
```

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

In [97]:

```
# 31. Write a NumPy program to create a 3x3x3 array filled with arbitrary values.
import numpy as np
x = np.random.random((3, 3, 3))
print(x)
```

```
[[[0.755523598795747 0.83889445819586  0.035537026671508]
  [0.121443769800747 0.891008055063653 0.515092234242717]
  [0.924419011628661 0.581242629710508 0.773256387306404]]

 [[0.422676894344956 0.825464751264126 0.385404561727487]
  [0.354345962251899 0.428078522457806 0.395065181553887]
  [0.844629714147486 0.373823809162965 0.371580976185914]]

 [[0.415942958007548 0.424544476363223 0.662804842833816]
  [0.189775925616323 0.722478651112437 0.253660972902271]
  [0.962674138524461 0.132661245819338 0.806796697367721]]]
```

In [98]:

```
# 32. Write a NumPy program to compute the sum of all elements, the sum of each column and the sum of each row in a given
import numpy as np
x = np.array([[0, 1], [2, 3]])
print("Original array:")
print(x)
print("Sum of all elements:")
print(np.sum(x))
print("Sum of each column:")
print(np.sum(x, axis=0))
print("Sum of each row:")
print(np.sum(x, axis=1))
```

```
Original array:
[[0 1]
 [2 3]]
Sum of all elements:
6
Sum of each column:
[2 4]
Sum of each row:
[1 5]
```

In [99]:

```
# 33. Write a NumPy program to compute the inner product of two given vectors.
import numpy as np
x = np.array([4, 5])
y = np.array([7, 10])
print("Original vectors:")
print(x)
print(y)
print("Inner product of said vectors:")
print(np.dot(x, y))
```

```
Original vectors:
[4 5]
[ 7 10]
Inner product of said vectors:
78
```

In [100…

```
# 34. Write a NumPy program to add a vector to each row of a given matrix.
import numpy as np
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
v = np.array([1, 1, 0])
print("Original vector:")
print(v)
print("Original matrix:")
print(m)
result = np.empty_like(m)
for i in range(4):
    result[i, :] = m[i, :] + v
print("\nAfter adding the vector v to each row of the matrix m:")
print(result)
```

```
Original vector:
[1 1 0]
```

```
[1 1 0]
Original matrix:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

After adding the vector v to each row of the matrix m:
[[ 2  3  3]
 [ 5  6  6]
 [ 8  9  9]
 [11 12 12]]
```

In [101…

```
# 35. Write a NumPy program to save a given array to a binary file.
import numpy as np
import os
a = np.arange(20)
np.save('temp_arra.npy', a)
print("Check if 'temp_arra.npy' exists or not?")
if os.path.exists('temp_arra.npy'):
    x2 = np.load('temp_arra.npy')
    print(np.array_equal(a, x2))
```

```
Check if 'temp_arra.npy' exists or not?
True
```

In [102…

```
# 36. Write a NumPy program to save a given array to a binary file.
import numpy as np
import os
x = np.arange(10)
y = np.arange(11, 20)
print("Original arrays:")
print(x)
print(y)
np.savez('temp_arra.npz', x=x, y=y)
print("Load arrays from the 'temp_arra.npz' file:")
with np.load('temp_arra.npz') as data:
    x2 = data['x']
    y2 = data['y']
    print(x2)
    print(y2)
```

```
Original arrays:
[0 1 2 3 4 5 6 7 8 9]
[11 12 13 14 15 16 17 18 19]
Load arrays from the 'temp_arra.npz' file:
[0 1 2 3 4 5 6 7 8 9]
[11 12 13 14 15 16 17 18 19]
```

In [103…

```
# 37. Write a NumPy program to save a given array to a text file and load it.
import numpy as np
import os
x = np.arange(12).reshape(4, 3)
print("Original array:")
print(x)
header = 'col1 col2 col3'
np.savetxt('temp.txt', x, fmt="%d", header=header)
print("After loading, content of the text file:")
result = np.loadtxt('temp.txt')
print(result)
```

```
Original array:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
After loading, content of the text file:
[[ 0.  1.  2.]
 [ 3.  4.  5.]
 [ 6.  7.  8.]
 [ 9. 10. 11.]]
```

In [104…

```
# 38. Write a NumPy program to convert a given array into bytes, and load it as an array.
import numpy as np
import os
a = np.array([1, 2, 3, 4, 5, 6])
print("Original array:")
print(a)
a_bytes = a.tobytes()
a2 = np.frombuffer(a_bytes, dtype=a.dtype)
print("After loading, content of the text file:")
print(a2)
print(np.array_equal(a, a2))
```

```
Original array:
[1 2 3 4 5 6]
After loading, content of the text file:
[1 2 3 4 5 6]
True
```

In [105…

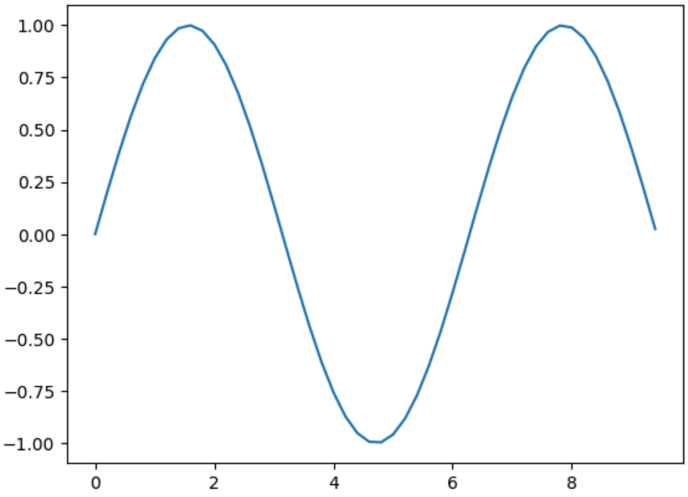
```
# 39. Write a NumPy program to convert a given list into an array, then again convert it into a list. Check initial list
import numpy as np
a = [[1, 2], [3, 4]]
x = np.array(a)
a2 = x.tolist()
print(a == a2)
```

```
True
```

In [106…

```
# 40. Write a NumPy program to compute the x and y coordinates for points on a sine curve and plot the points using matpl
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 3 * np.pi, 0.2)
y = np.sin(x)
print("Plot the points using matplotlib:")
plt.plot(x, y)
plt.show()
```

```
Plot the points using matplotlib:
```



In [107…

```python
# 41. Write a NumPy program to convert numpy dtypes to native Python types
import numpy as np
print("numpy.float32 to python float")
x = np.float32(0)
print(type(x))
pyval = x.item()
print(type(pyval))
```

```
numpy.float32 to python float
<class 'numpy.float32'>
<class 'float'>
```

In [108…

```python
# 42. Write a NumPy program to add elements to a matrix. If an element in the matrix is 0, we will not add the element be
import numpy as np
def sum_matrix_Elements(m):
    arra = np.array(m)
    element_sum = 0
    for p in range(len(arra)):
        for q in range(len(arra[p])):
            if arra[p][q] == 0 and p < len(arra)-1:
                arra[p+1][q] = 0

            element_sum += arra[p][q]
    return element_sum

m = [[1, 1, 0, 2],
     [0, 3, 0, 3],
     [1, 0, 4, 4]]
print("Original matrix:")
print(m)
print("Sum:")
print(sum_matrix_Elements(m))
```

```
Original matrix:
[[1, 1, 0, 2], [0, 3, 0, 3], [1, 0, 4, 4]]
Sum:
14
```

In [109…

```python
# 43. Write a NumPy program to find missing data in a given array.
import numpy as np
nums = np.array([[3, 2, np.nan, 1],
                 [10, 12, 10, 9],
                 [5, np.nan, 1, np.nan]])

print("Original array:")
print(nums)
print("\nFind the missing data of the said array:")
print(np.isnan(nums))
```

```
Original array:
[[ 3.   2. nan  1.]
 [10. 12. 10.   9.]
 [ 5. nan  1. nan]]

Find the missing data of the said array:
[[False False  True False]
 [False False False False]
 [False  True False  True]]
```

In [110…

```python
# 44. Write a NumPy program to check whether two arrays are equal (element wise) or not.
import numpy as np
nums1 = np.array([0.5, 1.5, 0.2])
nums2 = np.array([0.4999999999, 1.500000000, 0.2])
np.set_printoptions(precision=15)
print("Original arrays:")
print(nums1)
print(nums2)
print("\nTest said two arrays are equal (element wise) or not:?")
print(nums1 == nums2)
nums1 = np.array([0.5, 1.5, 0.23])
nums2 = np.array([0.4999999999, 1.5000000001, 0.23])
print("\nOriginal arrays:")
np.set_printoptions(precision=15)
print(nums1)
print(nums2)
print("\nTest said two arrays are equal (element wise) or not:?")
print(np.equal(nums1, nums2))
```

```
Original arrays:
[0.5 1.5 0.2]
[0.4999999999 1.5          0.2         ]
```

```
Test said two arrays are equal (element wise) or not:?
[False  True  True]

Original arrays:
[0.5  1.5  0.23]
[0.4999999999 1.5000000001 0.23        ]

Test said two arrays are equal (element wise) or not:?
[False False  True]
```

In [111…

```
# 45. Write a NumPy program to create a one-dimensional array of single, two and three-digit numbers.
import numpy as np
nums = np.arange(1, 21)
print("One-dimensional array of single digit numbers:")
print(nums)
nums = np.arange(10, 21)
print("\nOne-dimensional array of two digit numbers:")
print(nums)
nums = np.arange(100, 201)
print("\nOne-dimensional array of three digit numbers:")
print(nums)
```

```
One-dimensional array of single digit numbers:
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]

One-dimensional array of two digit numbers:
[10 11 12 13 14 15 16 17 18 19 20]

One-dimensional array of three digit numbers:
[100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171
 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189
 190 191 192 193 194 195 196 197 198 199 200]
```

In [112…

```
# 46. Write a NumPy program to create a two-dimensional array of a specified format.
import numpy as np
print("Create an array of shape (15,10):")
print("Command-1")
print(np.arange(1, 151).reshape(15, 10))
print("\nCommand-2")
print(np.arange(1, 151).reshape(-1, 10))
print("\nCommand-3")
print(np.arange(1, 151).reshape(15, -1))
```

```
Create an array of shape (15,10):
Command-1
[[  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  37  38  39  40]
 [ 41  42  43  44  45  46  47  48  49  50]
 [ 51  52  53  54  55  56  57  58  59  60]
 [ 61  62  63  64  65  66  67  68  69  70]
 [ 71  72  73  74  75  76  77  78  79  80]
 [ 81  82  83  84  85  86  87  88  89  90]
 [ 91  92  93  94  95  96  97  98  99 100]
 [101 102 103 104 105 106 107 108 109 110]
 [111 112 113 114 115 116 117 118 119 120]
 [121 122 123 124 125 126 127 128 129 130]
 [131 132 133 134 135 136 137 138 139 140]
 [141 142 143 144 145 146 147 148 149 150]]

Command-2
[[  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  37  38  39  40]
 [ 41  42  43  44  45  46  47  48  49  50]
 [ 51  52  53  54  55  56  57  58  59  60]
 [ 61  62  63  64  65  66  67  68  69  70]
 [ 71  72  73  74  75  76  77  78  79  80]
 [ 81  82  83  84  85  86  87  88  89  90]
 [ 91  92  93  94  95  96  97  98  99 100]
 [101 102 103 104 105 106 107 108 109 110]
 [111 112 113 114 115 116 117 118 119 120]
 [121 122 123 124 125 126 127 128 129 130]
 [131 132 133 134 135 136 137 138 139 140]
 [141 142 143 144 145 146 147 148 149 150]]

Command-3
[[  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  37  38  39  40]
 [ 41  42  43  44  45  46  47  48  49  50]
 [ 51  52  53  54  55  56  57  58  59  60]
 [ 61  62  63  64  65  66  67  68  69  70]
 [ 71  72  73  74  75  76  77  78  79  80]
 [ 81  82  83  84  85  86  87  88  89  90]
 [ 91  92  93  94  95  96  97  98  99 100]
 [101 102 103 104 105 106 107 108 109 110]
 [111 112 113 114 115 116 117 118 119 120]
 [121 122 123 124 125 126 127 128 129 130]
 [131 132 133 134 135 136 137 138 139 140]
 [141 142 143 144 145 146 147 148 149 150]]
```

In [113…

```
# 47. Write a NumPy program to create a one-dimensional array of forty pseudo-randomly generated values. Select random nu
import numpy as np
np.random.seed(10)
print(np.random.rand(40))
```

```
[0.771320643266746 0.020751949359402 0.633648234926275 0.748803882538612
 0.49850701230259  0.224796645530848 0.198062864759624 0.760530712198959
 0.169110836562535 0.08833981417401  0.685359818367797 0.953393346194937
 0.003948266327914 0.512192263385777 0.812620961652114 0.612526066829388
 0.7217553174318   0.291876068170633 0.917774122512943 0.714575783397691
 0.542544368011261 0.142170047601527 0.373340760051469 0.674133615066345
 0.441833174422996 0.434013993333294 0.617766978469317 0.513138242554391
 0.650397181931467 0.601038953404544 0.805223196832746 0.521647152393634
 0.908648880808668 0.319236088988545 0.090459349270907 0.300700056636203
 0.11398436186355  0.828681326307677 0.04689631938925  0.626287148311393]
```

In [114…

```
# 48. Write a NumPy program to create a two-dimensional array with shape (8,5) of random numbers. Select random numbers fr
import numpy as np
np.random.seed(20)
cbrt = np.cbrt(7)
nd1 = 200
print(cbrt * np.random.randn(10, 4) + nd1)
```

```
[[201.69082669736346 200.3746763082702  200.68394275021677
  195.51750123129818]
 [197.92478991563303 201.07066048488275 201.79714021459858
  198.12823310277952]
 [200.96238963339914 200.77744291067876 200.61875865227867
  199.05613893807728]
 [198.48492638364303 198.38860811141893 197.55239946005003
  200.47003620983824]
 [199.9154583876559  202.99877319355295 202.01069856971475
  200.77735483308504]
 [199.67739161408255 193.89831807071494 202.14273592752286
  202.54951299405823]
 [199.53450968631876 199.75126019761723 199.791457270905
  202.97687756587686]
 [200.24634412660365 196.0460693352411  198.3061125324022
  197.88701546225133]
 [201.7845091237894  203.94032834458278 198.21152802602944
  196.9144607143198 ]
 [201.0082480992413  197.03285103581035 200.6305276348363
  197.82590294139914]]
```

In [115…

```
# 49. Write a NumPy program to generate a uniform, non-uniform random sample from a given 1-D array with and without repla
import numpy as np
print("Generate a uniform random sample with replacement:")
print(np.random.choice(7, 5))
print("\nGenerate a uniform random sample without replacement:")
print(np.random.choice(7, 5, replace=False))
print("\nGenerate a non-uniform random sample with replacement:")
print(np.random.choice(7, 5, p=[0.1, 0.2, 0, 0.2, 0.4, 0, 0.1]))
print("\nGenerate a uniform random sample without replacement:")
print(np.random.choice(7, 5, replace=False, p=[0.1, 0.2, 0, 0.2, 0.4, 0, 0.1]))
```

```
Generate a uniform random sample with replacement:
[2 2 1 5 2]

Generate a uniform random sample without replacement:
[6 3 5 4 2]

Generate a non-uniform random sample with replacement:
[6 1 6 0 1]

Generate a uniform random sample without replacement:
[1 6 4 3 0]
```

In [116…

```
# 50. Write a NumPy program to create a 4x4 array with random values. Create an array from the said array swapping first a
import numpy as np
nums = np.arange(16, dtype='int').reshape(-1, 4)
print("Original array:")
print(nums)
nums[[0,-1],:] = nums[[-1,0],:]
print("\nNew array after swapping first and last rows of the said array:")
print(nums)
```

```
Original array:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

New array after swapping first and last rows of the said array:
[[12 13 14 15]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [ 0  1  2  3]]
```

In [117…

```
# 51. Write a NumPy program to create a new array of given shape (5,6) and type, filled with zeros.
import numpy as np
nums = np.zeros(shape=(5, 6), dtype='int')
print("Original array:")
print(nums)
nums[::2, ::2] = 3
nums[1::2, ::2] = 7
print("\nNew array:")
print(nums)
```

```
Original 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]]

New array:
[[3 0 3 0 3 0]
 [7 0 7 0 7 0]
 [3 0 3 0 3 0]
 [7 0 7 0 7 0]
 [3 0 3 0 3 0]]
```

In [118…

```
# 52. Write a NumPy program to sort a given array by row and column in ascending order.
import numpy as np
nums = np.array([[5.54, 3.38, 7.99],
              [3.54, 4.38, 6.99],
              [1.54, 2.39, 9.29]])

print("Original array:")
print(nums)
print("\nSort the said array by row in ascending order:")
print(np.sort(nums))
print("\nSort the said array by column in ascending order:")
print(np.sort(nums, axis=0))
```

```
Original array:
[[5.54 3.38 7.99]
 [3.54 4.38 6.99]
 [1.54 2.39 9.29]]

Sort the said array by row in ascending order:
[[3.38 5.54 7.99]
 [3.54 4.38 6.99]
 [1.54 2.39 9.29]]

Sort the said array by column in ascending order:
[[1.54 2.39 6.99]
 [3.54 3.38 7.99]
 [5.54 4.38 9.29]]
```

In [119…

```
# 53. Write a NumPy program to extract all numbers from a given array less and greater than a specified number.
import numpy as np
nums = np.array([[5.54, 3.38, 7.99],
              [3.54, 4.38, 6.99],
              [1.54, 2.39, 9.29]])
print("Original array:")
print(nums)
n = 5
print("\nElements of the said array greater than", n)
print(nums[nums > n])
n = 6
print("\nElements of the said array less than", n)
print(nums[nums < n])
```

```
Original array:
[[5.54 3.38 7.99]
 [3.54 4.38 6.99]
 [1.54 2.39 9.29]]

Elements of the said array greater than 5
[5.54 7.99 6.99 9.29]

Elements of the said array less than 6
[5.54 3.38 3.54 4.38 1.54 2.39]
```

In [120…

```
# 54. Write a NumPy program to replace all numbers in a given array equal, less and greater than a given number.
import numpy as np
nums = np.array([[5.54, 3.38, 7.99],
              [3.54, 8.32, 6.99],
              [1.54, 2.39, 9.29]])
print("Original array:")
print(nums)
n = 8.32
r = 18.32
print("\nReplace elements of the said array which are equal to", n, "with", r)
print(np.where(nums == n, r, nums))
print("\nReplace elements of the said array which are less than", n, "with", r)
print(np.where(nums < n, r, nums))
print("\nReplace elements of the said array which are greater than", n, "with", r)
print(np.where(nums > n, r, nums))
```

```
Original array:
[[5.54 3.38 7.99]
 [3.54 8.32 6.99]
 [1.54 2.39 9.29]]

Replace elements of the said array which are equal to 8.32 with 18.32
[[ 5.54  3.38  7.99]
 [ 3.54 18.32  6.99]
 [ 1.54  2.39  9.29]]

Replace elements of the said array which are less than 8.32 with 18.32
[[18.32 18.32 18.32]
 [18.32  8.32 18.32]
 [18.32 18.32  9.29]]

Replace elements of the said array which are greater than 8.32 with 18.32
[[ 5.54  3.38  7.99]
 [ 3.54  8.32  6.99]
 [ 1.54  2.39 18.32]]
```

In [121…

```
# 55. Write a NumPy program to create an array of equal shape and data type for a given array.
import numpy as np
nums = np.array([[5.54, 3.38, 7.99],
              [3.54, 8.32, 6.99],
              [1.54, 2.39, 9.29]])
print("Original array:")
print(nums)
print("\nNew array of equal shape and data type of the said array filled by 0:")
print(np.zeros_like(nums))
```

```
Original array:
[[5.54 3.38 7.99]
 [3.54 8.32 6.99]
 [1.54 2.39 9.29]]

New array of equal shape and data type of the said array filled by 0:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

In [122…

```
# 56. Write a NumPy program to create a three-dimensional array with the shape (3,5,4) and set it to a variable.
import numpy as np
nums = np.array([[[1, 5, 2, 1],
               [4, 3, 5, 6],
               [6, 3, 0, 6],
               [7, 3, 5, 0],
               [2, 3, 3, 5]],
              [[2, 2, 3, 1],
               [4, 0, 0, 5],
               [6, 3, 2, 1],
               [5, 1, 0, 0],
               [0, 1, 9, 1]],

              [[3, 1, 4, 2],
               [4, 1, 6, 0],
               [1, 2, 0, 6],
               [8, 3, 4, 0],
               [2, 0, 2, 8]]])

print("Array:")
print(nums)
```

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

 [[2 2 3 1]
  [4 0 0 5]
  [6 3 2 1]
  [5 1 0 0]
  [0 1 9 1]]

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

In [123…

```
# 57. Write a NumPy program to create a 4x4 array. Create an array from said array by swapping first and last, second and
import numpy as np
nums = np.arange(16, dtype='int').reshape(-1, 4)
print("Original array:")
print(nums)
new_nums = nums[:, ::-1]
print("\nNew array after swapping first and last columns of the said array:")
print(new_nums)
```

```
Original array:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

New array after swapping first and last columns of the said array:
[[ 3  2  1  0]
 [ 7  6  5  4]
 [11 10  9  8]
 [15 14 13 12]]
```

In [124…

```
# 58. Write a NumPy program to swap rows and columns of a given array in reverse order.
import numpy as np
nums = np.array([[[1, 2, 3, 4],
               [0, 1, 3, 4],
               [90, 91, 93, 94],
               [5, 0, 3, 2]]])
print("Original array:")
print(nums)
new_nums = nums[::-1, ::-1]
print("\nSwap rows and columns of the said array in reverse order:")
print(new_nums)
```

```
Original array:
[[[ 1  2  3  4]
  [ 0  1  3  4]
  [90 91 93 94]
  [ 5  0  3  2]]]

Swap rows and columns of the said array in reverse order:
[[[ 5  0  3  2]
  [90 91 93 94]
  [ 0  1  3  4]
  [ 1  2  3  4]]]
```

In [125…

```
# 59. Write a NumPy program to multiply two given arrays of the same size element-by-element.
import numpy as np
nums1 = np.array([[2, 5, 2],
              [1, 5, 5]])

nums2 = np.array([[5, 3, 4],
              [3, 2, 5]])
print("Array1:")
print(nums1)
print("Array2:")
print(nums2)
print("\nMultiply said arrays of same size element-by-element:")
print(np.multiply(nums1, nums2))
```

```
Array1:
[[2 5 2]
 [1 5 5]]
Array2:
[[5 3 4]
 [3 2 5]]

Multiply said arrays of same size element-by-element:
[[10 15  8]
```