**1.Write a NumPy program to get the numpy version and show numpy build configuration.**

import numpy as np

print("Version:", np.\_\_version\_\_)

print("\n\nBuild Configuration")

np.show\_config()

**Output:**

**Version: 1.18.1**

**Build Configuration**

**blas\_mkl\_info:**

**NOT AVAILABLE**

**blis\_info:**

**NOT AVAILABLE**

**openblas\_info:**

**library\_dirs = ['C:\\projects\\numpy-wheels\\numpy\\build\\openblas\_info']**

**libraries = ['openblas\_info']**

**language = f77**

**define\_macros = [('HAVE\_CBLAS', None)]**

**blas\_opt\_info:**

**library\_dirs = ['C:\\projects\\numpy-wheels\\numpy\\build\\openblas\_info']**

**libraries = ['openblas\_info']**

**language = f77**

**define\_macros = [('HAVE\_CBLAS', None)]**

**lapack\_mkl\_info:**

**NOT AVAILABLE**

**openblas\_lapack\_info:**

**library\_dirs = ['C:\\projects\\numpy-wheels\\numpy\\build\\openblas\_lapack\_info']**

**libraries = ['openblas\_lapack\_info']**

**language = f77**

**define\_macros = [('HAVE\_CBLAS', None)]**

**lapack\_opt\_info:**

**library\_dirs = ['C:\\projects\\numpy-wheels\\numpy\\build\\openblas\_lapack\_info']**

**libraries = ['openblas\_lapack\_info']**

**language = f77**

**define\_macros = [('HAVE\_CBLAS', None)]**

**2. Write a NumPy program to  get help on the add function.**

import numpy as np

np.info(np.add)

**Output:**

**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.]])**

**3. Write a NumPy program to test whether none of the elements of a given array is 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))

**Output:**

**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**

**4. Write a NumPy program to test if any of the elements of a given array is non-zero.**

import numpy as np

x = np.array([1, 0, 0, 0])

print("Original array:")

print(x)

print("Test if 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 if any of the elements of a given array is non-zero:")

print(np.any(x))

**Output:**

**Original array:**

**[1 0 0 0]**

**Test if any of the elements of a given array is non-zero:**

**True**

**Original array:**

**[0 0 0 0]**

**Test if any of the elements of a given array is non-zero:**

**False**

**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))

**Output:**

**Original array**

**[ 1. 0. nan inf]**

**Test a given array element-wise for finiteness :**

**[ True True False False]**

**6. Write a NumPy program to test element-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))

**Output:**

**Original array**

**[ 1. 0. nan inf]**

**Test element-wise for positive or negative infinity:**

**[False False False True]**

**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))

**Output:**

**Original array**

**[ 1. 0. nan inf]**

**Test element-wise for NaN:**

**[False False True False]**

**8. Write a NumPy program to test element-wise for complex number, real number of a given array. Also test if a given number is a scalar type or not.**

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("3.1 is scalar:", np.isscalar(3.1))

print("[3.1] is scalar:", np.isscalar([3.1]))

**Output:**

**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:**

**3.1 is scalar: True**

**[3.1] is scalar: False**

**9. Write a NumPy program to test if 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))

**Output:**

**Test if two arrays are element-wise equal within a tolerance:**

**False**

**True**

**False**

**False**

**True**

**10. Write a NumPy program to create an element-wise comparison (greater, greater\_equal, less and less\_equal) of two given arrays.**

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))

**Output:**

**Original numbers:**

**[3 5]**

**[2 5]**

**Comparison - greater**

**[ True False]**

**Comparison - greater\_equal**

**[ True True]**

**Comparison - less**

**[False False]**

**Comparison - less\_equal**

**[False True]**

**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))

**Output:**

**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**

**12. Write a NumPy program to create an array with the values 1, 7, 13, 105 and determine the size of the memory occupied by the array.**

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))

**Output:**

**Original array:**

**[ 1 7 13 105]**

**Size of the memory occupied by the said array:**

**16 bytes**

**13. Write a NumPy program to create an array of 10 zeros,10 ones, 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)

**Output:**

**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.]**

**14. Write a NumPy program to create an array of the integers from 30 to70.**

import numpy as np

array=np.arange(30,71)

print("Array of the integers from 30 to 70")

print(array)

**Output:**

**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]**

**15. Write a NumPy program to create an array of all the 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)

**Output:**

**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]**

**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)

**Output:**

**3x3 matrix:**

**[[1. 0. 0.]**

**[0. 1. 0.]**

**[0. 0. 1.]]**

**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)

**Output:**

**Random number between 0 and 1:**

**[0.21105278]**

**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)

**Output:**

**15 random numbers from a standard normal distribution:**

**[ 0.0722008 0.8065558 -0.26524681 -1.19586017 0.06542999 1.40869047**

**0.19169348 0.64314327 0.83495768 0.18579548 0.8936144 -0.01114068**

**-0.67336777 -0.00384282 1.00666944]**

**19. Write a NumPy program to create a vector with values ranging from 15 to 55 and print all values except the first and last.**

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])

**Output:**

**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]**

**20. Write a NumPy program to create a 3X4 array using 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=" ")

**Output:**

**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**

**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(5, 50, 10)

print("Length 10 with values evenly distributed between 5 and 50:")

print(v)

**Output:**

**Length 10 with values evenly distributed between 5 and 50:**

**[ 5. 10. 15. 20. 25. 30. 35. 40. 45. 50.]**

**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 from 9 to 15.**

import numpy as np

x = np.arange(20)

print("Original vector:")

print(x)

print("After changing the sign of the numbers in the range from 9 to 15:")

x[(x >= 9) & (x <= 15)] \*= -1

print(x)

**Output:**

**Original vector:**

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

**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]**

**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)

**Output:**

**Vector of length 5 filled with arbitrary integers from 0 to 10:**

**[4 7 2 8 5]**

**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)

**Output:**

**Vector-1**

**[1 8 3 5]**

**Vector-2**

**[7 5 5 9]**

**Multiply the values of two said vectors:**

**[ 7 40 15 45]**

**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)

**Output:**

**[[10 11 12 13]**

**[14 15 16 17]**

**[18 19 20 21]]**

**26. Write a NumPy program to find the number of rows and columns of 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)

**Output:**

**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)**

**27. Write a NumPy program to create a 3x3 identity matrix, i.e. diagonal elements are 1, the rest are 0.**

import numpy as np

x = np.eye(3)

print(x)

**Output:**

**[[1. 0. 0.]**

**[0. 1. 0.]**

**[0. 0. 1.]]**

**28. Write a NumPy program to create a 10x10 matrix, in which the elements on the borders will be equal to 1, and inside 0.**

import numpy as np

x = np.ones((10, 10))

x[1:-1, 1:-1] = 0

print(x)

**Output:**

**[[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.]]**

**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)

**Output:**

**[[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]]**

**30. Write a NumPy program to create an 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)

**Output:**

**[[0. 1. 0. 1.]**

**[1. 0. 1. 0.]**

**[0. 1. 0. 1.]**

**[1. 0. 1. 0.]]**

**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)

**Output:**

**[[[0.20559919 0.13438198 0.83752326]**

**[0.94072944 0.85237732 0.17716011]**

**[0.59225776 0.88193484 0.95068337]]**

**[[0.841262 0.91807646 0.51632102]**

**[0.76130197 0.26881533 0.90647864]**

**[0.80310539 0.10487766 0.59120595]]**

**[[0.2094306 0.62459476 0.75849465]**

**[0.32326467 0.528206 0.63925565]**

**[0.7454037 0.3961623 0.55419649]]]**

**32. Write a NumPy program to compute sum of all elements, sum of each column and sum of each row of an given array.**

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))

**Output:**

**Original array:**

**[[0 1]**

**[2 3]]**

**Sum of all elements:**

**6**

**Sum of each column:**

**[2 4]**

**Sum of each row:**

**[1 5]**

**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))

**Output:**

**Original vectors:**

**[4 5]**

**[ 7 10]**

**Inner product of said vectors:**

**78**

**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)

**Output:**

**Original vector:**

**[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]]**

**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))

**Output:**

**Check if 'temp\_arra.npy' exists or not?**

**True**

**36. Write a NumPy program to save two given arrays into a single file in compressed format (.npz format) and load it.**

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)

**Output:**

**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]**

**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)

**Output:**

**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.]]**

**38. Write a NumPy program to convert a given array into bytes, and load it as array.**

import numpy as np

import os

a = np.array([1, 2, 3, 4, 5, 6])

print("Original array:")

print(a)

a\_bytes = a.tostring()

a2 = np.frombuffer(a\_bytes, dtype=a.dtype)

print("After loading, content of the text file:")

print(a2)

print(np.array\_equal(a, a2))

**Output:**

**Original array:**

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

**After loading, content of the text file:**

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

**True**

**39. Write a NumPy program to convert a given array into a list and then convert it into a list again.**

import numpy as np

a = [[1, 2], [3, 4]]

x = np.array(a)

a2 = x.tolist()

print(a == a2)

**Output:**

**True**

**40. Write a NumPy program to compute the x and y coordinates for points on a sine curve and plot the points using matplotlib.**

import numpy as np

import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve

x = np.arange(0, 3 \* np.pi, 0.2)

y = np.sin(x)

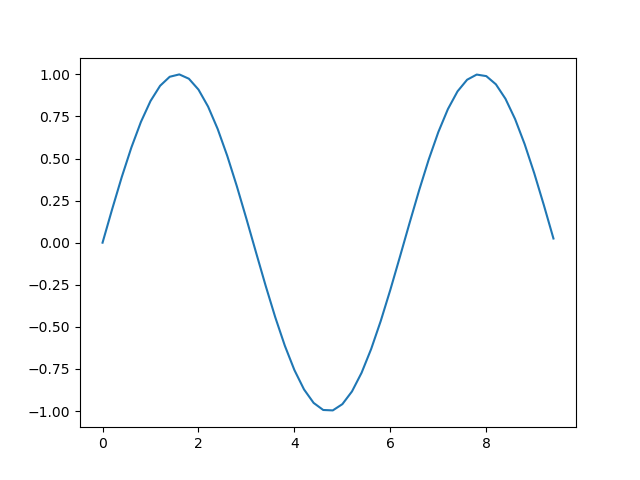
print("Plot the points using matplotlib:")

plt.plot(x, y)

plt.show()

**Output:**

**Plot the points using matplotlib:**

****

**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))

**Output:**

**numpy.float32 to python float**

**<class 'numpy.float32'>**

**<class 'float'>**