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

**Input:**

print(np.\_\_version\_\_)

print(np.show\_config())

**Output:**

1.24.1

openblas64\_\_info:

libraries = ['openblas64\_', 'openblas64\_']

library\_dirs = ['openblas\\lib']

language = c

define\_macros = [('HAVE\_CBLAS', None), ('BLAS\_SYMBOL\_SUFFIX', '64\_'), ('HAVE\_BLAS\_ILP64', None)]

runtime\_library\_dirs = ['openblas\\lib']

blas\_ilp64\_opt\_info:

libraries = ['openblas64\_', 'openblas64\_']

library\_dirs = ['openblas\\lib']

language = c

define\_macros = [('HAVE\_CBLAS', None), ('BLAS\_SYMBOL\_SUFFIX', '64\_'), ('HAVE\_BLAS\_ILP64', None)]

runtime\_library\_dirs = ['openblas\\lib']

openblas64\_\_lapack\_info:

libraries = ['openblas64\_', 'openblas64\_']

library\_dirs = ['openblas\\lib']

language = c

define\_macros = [('HAVE\_CBLAS', None), ('BLAS\_SYMBOL\_SUFFIX', '64\_'), ('HAVE\_BLAS\_ILP64', None), ('HAVE\_LAPACKE', None)]

runtime\_library\_dirs = ['openblas\\lib']

lapack\_ilp64\_opt\_info:

libraries = ['openblas64\_', 'openblas64\_']

library\_dirs = ['openblas\\lib']

language = c

define\_macros = [('HAVE\_CBLAS', None), ('BLAS\_SYMBOL\_SUFFIX', '64\_'), ('HAVE\_BLAS\_ILP64', None), ('HAVE\_LAPACKE', None)]

runtime\_library\_dirs = ['openblas\\lib']

Supported SIMD extensions in this NumPy install:

baseline = SSE,SSE2,SSE3

found = SSSE3,SSE41,POPCNT,SSE42,AVX,F16C,FMA3,AVX2,AVX512F,AVX512CD,AVX512\_SKX,AVX512\_CLX,AVX512\_CNL,AVX512\_ICL

not found =

None

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

print(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.]])

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

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

l = np.array([1,2,3,4,0])

if np.all(l):

print("The given array dosen't contains zero")

else:++

print("The given array containes zero")

**Output:**

The given array containes zero

1. **Write a NumPy program to create an array of 10 zeros, 10 ones, 10 fives.**

array1 = np.zeros(10)

array2 = np.ones(10)

array3 = np.ones(10)\*5

result = np.append(np.append(array1,array2),array3)

print(result)

**Output:**

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 5. 5. 5. 5.

5. 5. 5. 5. 5. 5.]

1. **Write a NumPy program to create an array of all the even integers from 30 to 70.**

print(np.arange(30,71,2))

**Output:**

[30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70]

1. **Write a NumPy program to create a 3x3 identity matrix.**

print(np.identity(3))

**Output:**

[[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]]

1. **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.**

array = np.arange(21)

array[(array>=9)&(array<=15)] \*=-1

print(array)

**Output:**

[ 0 1 2 3 4 5 6 7 8 -9 -10 -11 -12 -13 -14 -15 16 17 18 19 20]

1. **Write a NumPy program to find the number of rows and columns of a given matrix.**

array = np.identity(3)

print(np.shape(array))

**Output:**

(3, 3)

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

matrix = np.zeros([10,10])

matrix[0:,0] = 1

matrix[0,0:] = 1

matrix[-1,0:] = 1

matrix[0:,-1] = 1

print(matrix)

**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.]]

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

matrix = np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]])

print("The sum of all elements : ",np.sum(matrix))

print("Row sum : ",np.sum(matrix,axis=1))

print("Column sum : ",np.sum(matrix,axis=0))

**Output:**

The sum of all elements : 120

Row sum : [15 40 65]

Column sum : [18 21 24 27 30]

1. **Write a NumPy program to convert a given list into an array, then again convert it into a list. Check initial list and final list are equal or not.**

l = array([1,2,3,4,5])

print(l,type(l))

l = list(l)

print(l,type(l))

**Output:**

array([1, 2, 3, 4, 5]) <class 'array'>

[1, 2, 3, 4, 5] <class 'list'>

1. **Write a NumPy program to create a 3x3x3 array filled with arbitrary values.**

matrix = np.round(np.random.random([3,3,3]))

print(matrix)

**Output:**

[[[1. 0. 0.]

[0. 0. 1.]

[1. 1. 0.]]

[[0. 1. 0.]

[0. 0. 1.]

[1. 0. 0.]]

[[0. 0. 0.]

[0. 1. 1.]

[0. 1. 1.]]]

1. **Write a NumPy program to create a 5x5 zero matrix with elements on the main diagonal equal to 1, 2, 3, 4, 5.**

print(np.diag([1,2,3,4,5]))

**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]]

1. **Write a NumPy program to extract all numbers from a given array which are less and greater than a specified number.**

array = np.array([np.round(random()\*10) for i in range(10)])

n1 = int(input("Enter lower limit : "))

n2 = int(input("Enter upper limit : "))

f1 = array[np.where(array>=n1)]

f2 = array[np.where(f1<=n2)]

print(f2)

**Output:**

Enter lower limit : 2

Enter upper limit : 10

[0. 6. 3. 6. 0. 6. 5. 5.]

1. **Write a NumPy program to compute the sum of the diagonal element of a given array.**

array = np.array([np.round(random()\*10) for i in range(10)])

array = array.reshape(5,2)

print(array)

print("The trace of the matrix : ",np.trace(array))

**Output:**

[[ 7. 10.]

[ 9. 4.]

[ 8. 3.]

[ 5. 9.]

[ 5. 3.]]

The trace of the matrix : 11.0

1. **Get the common items between two arrays.**

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

b = np.array([7,2,10,2,7,4,9,4,9,8])

Desired Output:

array([2, 4])

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

b = np.array([5,6,7,8,9,10,11,12,13])

print(np.intersect1d(a,b))

**Output:**

[5 6 7 8 9]