**225229140 SURIYA**

In [1]:
```
import numpy as np
```

In [2]:
```
print("NumPy Version : ",np.__version__)
print(np.show_config())
```

```
NumPy Version :  1.14.0
mkl_info:
    libraries = ['mkl_rt']
    library_dirs = ['C:/Program Files (x86)/Microsoft Visual Studio/Shared/Anac
onda3_64\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_libra
ries_2016.4.246\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTools\\compile
rs_and_libraries_2016.4.246\\windows\\mkl\\include', 'C:\\Program Files (x86)
\\IntelSWTools\\compilers_and_libraries_2016.4.246\\windows\\mkl\\lib', 'C:/Pro
gram Files (x86)/Microsoft Visual Studio/Shared/Anaconda3_64\\Library\\includ
e']
blas_mkl_info:
    libraries = ['mkl_rt']
    library_dirs = ['C:/Program Files (x86)/Microsoft Visual Studio/Shared/Anac
onda3_64\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_libra
ries_2016.4.246\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTools\\compile
rs_and_libraries_2016.4.246\\windows\\mkl\\include', 'C:\\Program Files (x86)
\\IntelSWTools\\compilers_and_libraries_2016.4.246\\windows\\mkl\\lib', 'C:/Pro
gram Files (x86)/Microsoft Visual Studio/Shared/Anaconda3_64\\Library\\includ
e']
blas_opt_info:
    libraries = ['mkl_rt']
    library_dirs = ['C:/Program Files (x86)/Microsoft Visual Studio/Shared/Anac
onda3_64\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_libra
ries_2016.4.246\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTools\\compile
rs_and_libraries_2016.4.246\\windows\\mkl\\include', 'C:\\Program Files (x86)
\\IntelSWTools\\compilers_and_libraries_2016.4.246\\windows\\mkl\\lib', 'C:/Pro
gram Files (x86)/Microsoft Visual Studio/Shared/Anaconda3_64\\Library\\includ
e']
lapack_mkl_info:
    libraries = ['mkl_rt']
    library_dirs = ['C:/Program Files (x86)/Microsoft Visual Studio/Shared/Anac
onda3_64\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_libra
ries_2016.4.246\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTools\\compile
rs_and_libraries_2016.4.246\\windows\\mkl\\include', 'C:\\Program Files (x86)
\\IntelSWTools\\compilers_and_libraries_2016.4.246\\windows\\mkl\\lib', 'C:/Pro
gram Files (x86)/Microsoft Visual Studio/Shared/Anaconda3_64\\Library\\includ
e']
lapack_opt_info:
    libraries = ['mkl_rt']
    library_dirs = ['C:/Program Files (x86)/Microsoft Visual Studio/Shared/Anac
onda3_64\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_libra
ries_2016.4.246\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTools\\compile
rs_and_libraries_2016.4.246\\windows\\mkl\\include', 'C:\\Program Files (x86)
\\IntelSWTools\\compilers_and_libraries_2016.4.246\\windows\\mkl\\lib', 'C:/Pro
```

```
gram Files (x86)/Microsoft Visual Studio/Shared/Anaconda3_64\\Library\\includ
e']
None
```

In [4]:
```
x = np.zeros(10)
x
```

Out[4]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [7]:
```
print(x.size)
print(x.itemsize)
print(x.size*x.itemsize)
```

```
10
8
80
```

In [13]:
```
np.info(np.add)
```

```
add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=N
one, 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 may be the shape of one or
    the other).
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
    Values of True indicate to calculate the ufunc at that position, values
    of False indicate to leave the value in the output alone.
**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.  Returns 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.]])
```

In [31]:
```
a=np.zeros(10)
a[4]=1
a
```

Out[31]:
```
array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
```

In [12]:
```
x= np.arange(50,99)
print(x)
```

```
[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]
```

In [14]:
```
a=x[::-1]
print(a)
```

```
[98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75
 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51
 50]
```

In [22]:
```
a =np.arange(0, 16).reshape(4,4)
a
```

Out[22]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [26]:
```
a=[1,2,0,0,4,0]
b=np.nonzero(a)
b
```

Out[26]:
```
(array([0, 1, 4], dtype=int64),)
```

In [30]:
```
a=np.identity(4)
a
```

Out[30]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [36]:
```
a= np.random.random((4, 4,4))
print(a)
```

```
[[[9.66179173e-01 7.34820728e-01 9.77507292e-01 8.07464231e-01]
  [5.26120130e-04 1.19061967e-01 5.22539115e-01 6.94896858e-01]
  [7.95679697e-01 3.03257431e-01 3.14724044e-01 9.03467353e-02]
  [5.23223091e-02 8.07786894e-01 3.20528816e-01 3.48509961e-01]]

 [[7.49918457e-01 6.30273799e-01 7.73227061e-01 5.19472398e-01]
  [5.22948229e-01 2.96233381e-01 3.97715966e-01 9.10599218e-01]
  [4.84597793e-01 3.11134342e-03 3.94563699e-01 7.80434780e-01]
  [9.01399514e-01 1.92124443e-01 9.18563782e-01 3.87852421e-01]]

 [[6.69814675e-01 1.25935990e-01 7.60335166e-01 8.29405548e-02]
  [4.30927051e-01 3.75070176e-01 2.57108848e-01 7.63657808e-01]
  [2.82289165e-01 4.30476372e-01 8.94283587e-01 6.95026980e-01]
  [5.45375065e-01 1.76764910e-01 1.79252634e-01 8.57392178e-01]]

 [[7.92813301e-01 3.41595350e-01 5.51328205e-01 5.15849853e-01]
  [3.06215073e-01 3.12590662e-01 2.72904684e-02 8.55401596e-01]
  [2.61001039e-01 2.58043047e-01 5.01076065e-01 9.64683060e-01]
  [9.86348748e-01 2.08964973e-01 4.21040961e-01 2.69087507e-01]]]
```

In [33]:
```
import numpy as np
x = np.random.random((11,11))
print("Original Array:")
print(x)
xmin, xmax = x.min(), x.max()
print("Minimum and Maximum Values:")
print(xmin, xmax)
```

```
Original Array:
[[0.87555451 0.60137551 0.45365217 0.44800782 0.58138672 0.96644278
  0.52696277 0.59746654 0.96069671 0.71070092 0.40434957]
 [0.52528127 0.62570195 0.08425524 0.40035349 0.97204162 0.04523673
  0.0660037  0.33527902 0.09832397 0.89220058 0.13237339]
 [0.58952682 0.18629054 0.26244926 0.19909539 0.00466641 0.25718057
  0.30071935 0.1780194  0.89842448 0.89778946 0.06421278]
 [0.26163271 0.87379607 0.06320932 0.11760714 0.03863923 0.08628992
  0.39328147 0.09925832 0.11550968 0.33580223 0.74259736]
 [0.30668695 0.31398256 0.7721377  0.51309658 0.8820339  0.92961297
  0.25477386 0.97894894 0.88156532 0.28241693 0.10739306]
 [0.28549482 0.90011919 0.78530531 0.49563873 0.48313811 0.50211862
  0.53170904 0.81336775 0.14074121 0.70403135 0.36850412]
 [0.05110994 0.57200262 0.51213884 0.85942199 0.92715799 0.45532615
  0.93895964 0.35202469 0.95173639 0.28982106 0.43655354]
 [0.13822069 0.18091168 0.68469505 0.74456197 0.70288675 0.28702376
  0.06327344 0.60431914 0.45517531 0.16478321 0.60389888]
 [0.53093006 0.93283296 0.92271352 0.93929699 0.41478339 0.09135719
  0.26022861 0.13080874 0.00966676 0.48239894 0.70637948]
 [0.98747222 0.76052418 0.91136356 0.27719576 0.85429252 0.90522727
  0.89282408 0.43810559 0.69475425 0.99308799 0.77557443]
 [0.75312826 0.94933701 0.38547735 0.62344577 0.26534001 0.37212311
  0.54311181 0.59006998 0.3600903  0.27301294 0.32860881]]
Minimum and Maximum Values:
0.004666408412099754 0.9930879858101272
```

In [39]:
```
a=np.random.random(15)
print(a)
print("\n mean:",np.mean(a))
```

```
[0.26673171 0.89604931 0.75950324 0.11276573 0.80604531 0.1507828
 0.87644931 0.85465131 0.99025772 0.30013528 0.89512941 0.55932548
 0.75182882 0.21162648 0.41001114]

 mean: 0.5894195358829677
```

In [42]:
```
x = np.zeros((8,8),dtype=int)
x[1::2,::2] = 1
x[::2,1::2] = 1

print(x)
```

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

In [48]:
```
print(np.unravel_index(80,(8,7,6)))
```

```
(1, 6, 2)
```

In [46]:
```
import numpy as np
x = np.random.random((5,3))
print("First array:")
print(x)
y = np.random.random((3,2))
print("Second array:")
print(y)
z = np.dot(x, y)
print("Dot product of two arrays:")
print(z)
```

```
First array:
[[0.39753979 0.50029304 0.63275172]
 [0.8144938  0.9034187  0.85775742]
 [0.62822082 0.51778442 0.92174893]
 [0.32903635 0.01827436 0.67756536]
 [0.45941068 0.14779558 0.02877488]]
Second array:
[[0.2800663  0.91047254]
 [0.6251956  0.97950055]
 [0.07969724 0.44446184]]
Dot product of two arrays:
[[0.47454708 1.13322037]
 [0.86128657 2.00771379]
 [0.57312088 1.48883016]
 [0.15757714 0.61863025]
 [0.22335988 0.57583599]]
```

In [ ]:

In [ ]:

In [54]:
```
import numpy as np
x = np.array([0, 1, 2, 3, 4])
y = np.array([0, 2, 4])
print(np.intersect1d(x, y))









from datetime import datetime, timedelta

present = datetime.now()
yesterday = present - timedelta(1)
tomorrow = present + timedelta(1)

print("Yesterday was :")
print(yesterday.strftime('%d-%m-%Y'))
print("Today is :")
print(present.strftime('%d-%m-%Y'))
print("Tomorrow would be :")
print(tomorrow.strftime('%d-%m-%Y'))
```

```
[0 2 4]
Yesterday was :
20-12-2022
Today is :
21-12-2022
Tomorrow would be :
22-12-2022
```

In [55]:
```
import numpy as np
x = np.zeros((6,6))
print("Original array:")
print(x)
print("Row values ranging from 0 to 5.")
x += np.arange(6)
print(x)
```

```
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.]
 [0. 0. 0. 0. 0. 0.]]
Row values ranging from 0 to 5.
[[0. 1. 2. 3. 4. 5.]
 [0. 1. 2. 3. 4. 5.]
 [0. 1. 2. 3. 4. 5.]
 [0. 1. 2. 3. 4. 5.]
 [0. 1. 2. 3. 4. 5.]
 [0. 1. 2. 3. 4. 5.]]
```

In [60]:
```
import numpy as np
x = np.random.randint(0,3,5)
print("First array:")
print(x)
y = np.random.randint(0,3,5)
print("Second array:")
print(y)
print("Test above two arrays are equal or not!")
array_equal = np.allclose(x, y)
print(array_equal)
```

```
First array:
[2 2 1 2 1]
Second array:
[0 0 1 1 1]
Test above two arrays are equal or not!
False
```

In [ ]:
```
```