225229110 HARI PRASATH S

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

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

```
NumPy Version :  1.19.5
blas_mkl_info:
  NOT AVAILABLE
blis_info:
  NOT AVAILABLE
openblas_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_info']
    libraries = ['openblas_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
blas_opt_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_info']
    libraries = ['openblas_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
lapack_mkl_info:
  NOT AVAILABLE
openblas_lapack_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_lapack_info']
    libraries = ['openblas_lapack_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
lapack_opt_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_lapack_info']
    libraries = ['openblas_lapack_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
None
```

In [5]:
```
x = np.zeros(10)
x
```

Out[5]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [4]:
```
y = np.array([100,20,34])

print("Size of the array: ",
      y.size)

print("Memory size of one array element in bytes: ",
      y.itemsize)

print("Memory size of numpy array in bytes:",
      x.size * x.itemsize)
```

```
Size of the array:  3
Memory size of one array element in bytes:  4
Memory size of numpy array in bytes: 80
```

In [17]:
```
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 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.]])
```

In [18]:
```
a=np.zeros(10)
a[4]=1
a
```

Out[18]:
```
array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
```

In [8]:
```
h=np.arange(50,99)
h
```

Out[8]:
```
array([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 [9]:
```
b=h[::-1]
print(b)
```

```
[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 [12]:
```
p=np.arange(16,32).reshape(4,4)
p
```

Out[12]:
```
array([[16, 17, 18, 19],
       [20, 21, 22, 23],
       [24, 25, 26, 27],
       [28, 29, 30, 31]])
```

In [15]:
```
j=[1,2,0,0,4,0]
k=np.nonzero(j)
k
```

Out[15]:
```
(array([0, 1, 4], dtype=int64),)
```

In [20]:
```
a = np.identity(4)
a
```

Out[20]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [21]:
```
x = np.random.random((4, 4, 4))
print(x)
```

```
[[[0.13716632 0.92359538 0.82977962 0.82548679]
  [0.07920824 0.31958214 0.45050708 0.38655902]
  [0.47034618 0.9211659  0.12445655 0.44765501]
  [0.4066577  0.60995903 0.69166903 0.47253099]]

 [[0.01446923 0.58048867 0.28668476 0.38561905]
  [0.07002816 0.17920379 0.42152629 0.71566335]
  [0.1231912  0.4903758  0.86100375 0.811039  ]
  [0.66843183 0.04562665 0.27163998 0.52197259]]

 [[0.18559959 0.98153949 0.72289631 0.4133188 ]
  [0.66793102 0.51555602 0.43746041 0.54069517]
  [0.62142032 0.32173918 0.15612341 0.21998008]
  [0.17298413 0.66136022 0.58397376 0.06553625]]

 [[0.65631585 0.8704227  0.96834459 0.17638882]
  [0.05401099 0.01922239 0.71833523 0.00756624]
  [0.08300706 0.82037709 0.37236757 0.36288406]
  [0.59289537 0.43696732 0.94076971 0.71981315]]]
```

In [23]:
```
x = np.random.random((11,11))
print(x)
xmin, xmax = x.min(), x.max()
print("Min and Max Values:")
print(xmin, xmax)
```

```
[[0.00907046 0.83796437 0.20961779 0.94310561 0.89646206 0.82595673
  0.94925918 0.79121633 0.15898037 0.2664514  0.77977584]
 [0.45922335 0.71177914 0.56742897 0.0741304  0.94975689 0.77516433
  0.2986263  0.18771006 0.06276037 0.32628706 0.21688139]
 [0.69563659 0.39517149 0.07661208 0.86597101 0.57189074 0.25846242
  0.40499878 0.31519229 0.79324741 0.60732225 0.48943846]
 [0.57252369 0.45277587 0.05126924 0.07710611 0.29813097 0.63122621
  0.85083295 0.02766672 0.21711599 0.60731295 0.84753023]
 [0.62158069 0.13822441 0.58918241 0.68149766 0.19779575 0.34117382
  0.10755897 0.62271586 0.40523526 0.04576722 0.20322753]
 [0.27497821 0.71127498 0.28754511 0.8017132  0.24819657 0.17611408
  0.58038794 0.18006469 0.37375415 0.95987152 0.6434434 ]
 [0.73350949 0.43771168 0.88639063 0.27963002 0.67580153 0.01060898
  0.15981955 0.71228937 0.50216427 0.81971961 0.527213  ]
 [0.46792045 0.36913064 0.1040812  0.19545835 0.70934181 0.21878491
  0.68739025 0.22351456 0.780332   0.43492937 0.97402575]
 [0.65785875 0.06327948 0.77314619 0.93118586 0.84889512 0.67325727
  0.27404627 0.36521778 0.3133446  0.33392219 0.8970376 ]
 [0.54970247 0.22624149 0.84873787 0.04322001 0.35400014 0.92612173
  0.82648303 0.58107316 0.17464679 0.14212628 0.90838636]
 [0.11630339 0.98756124 0.73017307 0.93007358 0.08226835 0.59286648
  0.93478052 0.59969261 0.25850409 0.10856304 0.73884781]]
Min and Max Values:
0.0090704609847575 0.9875612443664871
```

In [24]:
```
a=np.random.random(15)
print(a)
print("\n mean:",np.mean(a))
```

```
[0.13201937 0.94770199 0.32453783 0.28938331 0.31095064 0.7445627
 0.34131984 0.44520357 0.54161526 0.99310908 0.67943838 0.09918674
 0.57028623 0.22474087 0.60213832]

 mean: 0.4830796091176841
```

In [28]:
```
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 [30]:
```
a=np.unravel_index(80,(8,7,6))
a
```

Out[30]:
```
(1, 6, 2)
```

In [29]:
```
import numpy as np
x = np.random.random((5,3))
print(x)
y = np.random.random((3,2))
print(y)
z = np.dot(x, y)
print("Dot product of two arrays:")
print(z)
```

```
[[0.85306129 0.2462653  0.63366187]
 [0.4444575  0.8613673  0.45061972]
 [0.3683228  0.77383511 0.76214189]
 [0.66253724 0.15761999 0.06808154]
 [0.88452301 0.30095107 0.22000934]]
[[0.77031093 0.83980991]
 [0.26710172 0.36577469]
 [0.38947468 0.39121459]]
Dot product of two arrays:
[[0.96969558 1.0543847 ]
 [0.74794814 0.86461518]
 [0.78725074 0.89053147]
 [0.57897629 0.64069324]
 [0.84743036 0.93898234]]
```

In [33]:
```
import numpy as np
ar1 = np.array([0, 1, 2, 3, 4])
ar2 = np.array([1, 3, 4])
print(np.intersect1d(ar1, ar2))
```

```
[1 3 4]
```

In [32]:
```
from datetime import datetime, timedelta

presentday = datetime.now()
yesterday = presentday - timedelta(1)
tomorrow = presentday + timedelta(1)
print("Yesterday = ", yesterday.strftime('%d-%m-%Y'))
print("Today = ", presentday.strftime('%d-%m-%Y'))
print("Tomorrow = ", tomorrow.strftime('%d-%m-%Y'))
```

```
Yesterday =  20-12-2022
Today =  21-12-2022
Tomorrow =  22-12-2022
```

In [37]:
```
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 [41]:
```
import numpy as np
x = np.random.randint(0,3,7)
print("First array:")
print(x)
y = np.random.randint(0,3,7)
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 0 2 1 0 2]
Second array:
[0 0 2 0 2 2 0]
Test above two arrays are equal or not!
False
```

In [42]:
```
import numpy as np
x = np.random.random(15)
print("Original array:")
print(x)
x[x.argmax()] = -1
print("Maximum value replaced by -1:")
print(x)
```

```
Original array:
[0.10999081 0.40815504 0.90370426 0.52103033 0.64809248 0.21204026
 0.21881699 0.55006131 0.38773033 0.5084895  0.51000147 0.21740286
 0.49371835 0.83098848 0.00112524]
Maximum value replaced by -1:
[ 0.10999081  0.40815504 -1.          0.52103033  0.64809248  0.21204026
  0.21881699  0.55006131  0.38773033  0.5084895   0.51000147  0.21740286
  0.49371835  0.83098848  0.00112524]
```

In [ ]:
```
```