Numpy

What is numpy?

Numpy is the fundamental package for scientific computing in Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebrea, basic statistical operations, random simulation and much more.

All the core of the Numpy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types

Numpy Arrays Vs Python Sequences

- NumPy arrays have a fixed size at creation, unlike Python lists(which can grow dynamically).Changing the size of an ndarray will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory.
- NumPy arrays facillitate advanced mathematical and other types of operation on large numbers of data. Typically, such opertations are executed more effciently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays, through these typicallly support python-sequence input, they convert such input to NumPy arrays prior to processing and they often output NumPy arrays.

Creating Numpy Arrays

In [ ]:
```
# np.array
import numpy as np
```

In [ ]:
```
# create numpy 1D array
a = np.array([1, 2, 3])
print(a)
```

```
[1 2 3]
```

In [ ]:
```
# type of array
print(type(a))
```

```
<class 'numpy.ndarray'>
```

In [ ]:
```
# create 2D array
b = np.array([[1, 2, 3], [4, 5, 6]])
print(b)
```

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

In [ ]:
```
# create 3D array
c = np.array([[[1, 2], [3, 4]], [[5, 6], [6, 7]]])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
[[[1 2]
  [3 4]]

 [[5 6]
  [6 7]]]
```

In [ ]:
```
# float datatype
d = np.array([1, 2, 3], dtype=float)
print(d)
```

```
[1. 2. 3.]
```

In [ ]:
```
# bool datatype
np.array([1, 2, 3], dtype=bool)
```

Out[ ]:
```
array([ True,  True,  True])
```

In [ ]:
```
# complex datatype
np.array([1, 2, 3], dtype=complex)
```

Out[ ]:
```
array([1.+0.j, 2.+0.j, 3.+0.j])
```

In [ ]:
```
# np.arange
np.arange(1, 11)
```

Out[ ]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [ ]:
```
np.arange(1, 11, 2)
```

Out[ ]:
```
array([1, 3, 5, 7, 9])
```

In [ ]:
```
# np.reshape
np.arange(1, 11).reshape(5, 2)
```

Out[ ]:
```
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10]])
```

In [ ]:
```
np.arange(1, 11).reshape(2, 5)
```

Out[ ]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

In [ ]:
```
# study error
np.arange(1, 11).reshape(5, 5)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
c:\Users\dhanr\Desktop\DS\Numpy\All_in_one_Numpy\01All_in_one_numpy.ipynb Cell 17 in <cell line: 2>()
      <a href='vscode-notebook-cell:/c%3A/Users/dhanr/Desktop/DS/Numpy/All_in_one_Numpy/01All_in_one_numpy.ipynb#X22sZmlsZQ%3D%3D?line=0'>1</a> # error
----> <a href='vscode-notebook-cell:/c%3A/Users/dhanr/Desktop/DS/Numpy/All_in_one_Numpy/01All_in_one_numpy.ipynb#X22sZmlsZQ%3D%3D?line=1'>2</a> np.arange(1, 11).reshape(5, 5)

ValueError: cannot reshape array of size 10 into shape (5,5)
```

In [ ]:
```
# np.ones -> every item is 1
# default array is float
np.ones((3, 4))
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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

In [ ]:
```
# np.zeros -> every item is 0
np.zeros((4, 2))
```

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

In [ ]:
```
# np.random
np.random.random((3, 4))
```

Out[ ]:
```
array([[0.85032095, 0.67623376, 0.74863953, 0.3381394 ],
       [0.24380273, 0.70702724, 0.22450031, 0.54841072],
       [0.17541142, 0.65340015, 0.32233203, 0.70954878]])
```

In [ ]:
```
# np.linspace
# -10 is lower range , 10 is upper range and 20 is number of item to generate
np.linspace(-10, 10, 20)
```

Out[ ]:
```
array([-10.        ,  -8.94736842,  -7.89473684,  -6.84210526,
        -5.78947368,  -4.73684211,  -3.68421053,  -2.63157895,
        -1.57894737,  -0.52631579,   0.52631579,   1.57894737,
         2.63157895,   3.68421053,   4.73684211,   5.78947368,
         6.84210526,   7.89473684,   8.94736842,  10.        ])
```

In [ ]:
```
# np.identity -> diagonal items is 1 and rest of item is 0
np.identity(3)
```

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

In [ ]:
```
np.identity(3, dtype=int)
```

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

Array Attributes

In [ ]:
```
a1 = np.arange(10)
a2 = np.arange(12, dtype=float).reshape(3, 4)
a3 = np.arange(8).reshape(2, 2, 2)
```

In [ ]:
```
print('a1: \n', a1)
print('a2: \n', a2)
print('a3: \n', a3)
```

```
a1:
 [0 1 2 3 4 5 6 7 8 9]
a2:
 [[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]]
a3:
 [[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```
# ndim -> number of dimensions
print('a1: ', a1.ndim)
print('a2: ', a2.ndim)
print('a3: ', a3.ndim)
```

```
a1:  1
a2:  2
a3:  3
```

In [ ]:
```
# shape -> how many rows and columns
print('a1: ', a1.shape)
a1
# 10 rows
```

Out[ ]:
```
a1:  (10,)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [ ]:
```
print('a2: ', a2.shape)
a2
# 3 rows 2 columns
```

Out[ ]:
```
a2:  (3, 4)
array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5.,  6.,  7.],
       [ 8.,  9., 10., 11.]])
```

In [ ]:
```
print('a3: ', a3.shape)
a3
'''first 2 describe how many 2D array and second and third to describe how many rows and columns'''
```

Out[ ]:
```
a3:  (2, 2, 2)
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
```

In [ ]:
```
# size -> number of items
print(a1.size)
a1
```

Out[ ]:
```
10
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [ ]:
```
print(a2.size)
a2
```

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

In [ ]:
```
print(a3.size)
a3
```

Out[ ]:
```
8
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```
# itemsize -> every item how many size occupy in memory
# int32 -> 4  and int64-> 8  same as float
a1.itemsize
```

Out[ ]:
```
4
```

In [ ]:
```
a2.itemsize
```

Out[ ]:
```
8
```

In [ ]:
```
a3.itemsize
```

Out[ ]:
```
4
```

In [ ]:
```
# dtype
print(a1.dtype)
print(a2.dtype)
print(a3.dtype)
```

```
int32
float64
int32
```

Changing Datatype

In [ ]:
```
# astype
print(a2.astype(np.int32))
a2.dtype
```

Out[ ]:
```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
dtype('float64')
```

Array Operations

In [ ]:
```
a1 = np.arange(12).reshape(3, 4)
a2 = np.arange(12, 24).reshape(3, 4)

print('a1: \n', a1)
print('a2: \n', a2)
```

```
a1:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
a2:
 [[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

In [ ]:
```
# scaler operations
# --> arithmetic <--

# every item multiply by 2
a1 * 2
```

Out[ ]:
```
array([[ 0,  2,  4,  6],
       [ 8, 10, 12, 14],
       [16, 18, 20, 22]])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```
# square of every item
a1 ** 2
```

Out[ ]:
```
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]], dtype=int32)
```

In [ ]:
```
# relational

# check every item
a2 > 5
```

Out[ ]:
```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

In [ ]:
```
a2 > 15
```

Out[ ]:
```
array([[False, False, False, False],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

In [ ]:
```
a2 == 15
```

Out[ ]:
```
array([[False, False, False,  True],
       [False, False, False, False],
       [False, False, False, False]])
```

In [ ]:
```
# vector operation
# shape must be equal -> a1.shape = a2.shape
a1 + a2
```

Out[ ]:
```
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])
```

In [ ]:
```
a2 ** a1
```

Out[ ]:
```
array([[         1,         13,        196,       3375],
       [     65536,    1419857,   34012224,  893871739],
       [-169803776, -288903179, -154967040, 1328067399]], dtype=int32)
```

Array Functions

In [ ]:
```
a1 = np.random.random((3, 3))
a1 = np.round(a1*100)
a1
```

Out[ ]:
```
array([[51., 36., 50.],
       [10.,  5., 27.],
       [ 8., 41., 88.]])
```

In [ ]:
```
# max/min/sum/prod
print('Max: ', np.max(a1))  # max number in array
print('Min: ', np.min(a1))  # min number in array
print('Sum: ', np.sum(a1))  # sum of all number of array
print('Product: ', np.prod(a1))  # product of all number of array
```

```
Max:  88.0
Min:  5.0
Sum:  316.0
Product:  3577115520000.0
```

In [ ]:
```
# 0 --> col and 1 --> row
# maximum item of every row
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
print(a1)
np.max(a1, axis=1)
```

```
[[51. 36. 50.]
 [10.  5. 27.]
 [ 8. 41. 88.]]
array([51., 27., 88.])
```

Out[ ]:

In [ ]:
```
# minimum item of every row
print(a1)
np.min(a1, axis=1)
```

```
[[51. 36. 50.]
 [10.  5. 27.]
 [ 8. 41. 88.]]
array([36.,  5.,  8.])
```

Out[ ]:

In [ ]:
```
# product of row
np.prod(a1, axis=1)
```

```
array([91800.,  1350., 28864.])
```

Out[ ]:

In [ ]:
```
# minimum row of every columns
print(a1)
np.max(a1, axis=0)
```

```
[[51. 36. 50.]
 [10.  5. 27.]
 [ 8. 41. 88.]]
array([51., 41., 88.])
```

Out[ ]:

In [ ]:
```
# mean/ median/ std/ var
np.mean(a1)
```

```
35.111111111111114
```

Out[ ]:

In [ ]:
```
# row and column of mean
np.mean(a1, axis=0)
```

```
array([23.        , 27.33333333, 55.        ])
```

Out[ ]:

In [ ]:
```
# median -> you can find row and colomn vise
np.median(a1)
```

```
36.0
```

Out[ ]:

In [ ]:
```
np.median(a1, axis=0)
```

```
array([10., 36., 50.])
```

Out[ ]:

In [ ]:
```
# standard deviation
np.std(a1)
```

```
25.044158531346383
```

Out[ ]:

In [ ]:
```
# variance
np.var(a1)
```

```
627.2098765432098
```

Out[ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```
# trignometric function
np.sin(a1)
# same for cos , tan ...
```

Out[ ]:
```
array([[ 0.67022918, -0.99177885, -0.26237485],
       [-0.54402111, -0.95892427,  0.95637593],
       [ 0.98935825, -0.15862267,  0.0353983 ]])
```

In [ ]:
```
# dot product

a2 = np.arange(12).reshape(3, 4)
a3 = np.arange(12, 24).reshape(4, 3)

print('a2: \n', a2)
print('a3: \n', a3)
```

```
a2:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
a3:
 [[12 13 14]
 [15 16 17]
 [18 19 20]
 [21 22 23]]
```

In [ ]:
```
''' (row1, col1) (row2, col2) -> row1 = col2 and col1 = row2 if this condition true then we apply dot product'''
np.dot(a2, a3)
```

Out[ ]:
```
array([[114, 120, 126],
       [378, 400, 422],
       [642, 680, 718]])
```

In [ ]:
```
# log and exponents
np.log(a1)
```

Out[ ]:
```
array([[3.93182563, 3.58351894, 3.91202301],
       [2.30258509, 1.60943791, 3.29583687],
       [2.07944154, 3.71357207, 4.47733681]])
```

In [ ]:
```
np.exp(a1)
```

Out[ ]:
```
array([[1.40934908e+22, 4.31123155e+15, 5.18470553e+21],
       [2.20264658e+04, 1.48413159e+02, 5.32048241e+11],
       [2.98095799e+03, 6.39843494e+17, 1.65163625e+38]])
```

In [ ]:
```
# round/ floor / ceil

np.random.random((2, 3))*100
# all are float
```

Out[ ]:
```
array([[59.04094484, 32.17821075, 11.55951977],
       [ 4.7255809 , 47.88422335,  0.38452897]])
```

In [ ]:
```
np.round(np.random.random((2, 3))*100)
# nearest integer -> roundoff
```

Out[ ]:
```
array([[91., 10., 61.],
       [66.,  5., 33.]])
```

In [ ]:
```
'''lets example - number is 6.9 then floor convert into 6 '''
np.floor(np.random.random((2, 3))*100)
```

Out[ ]:
```
array([[90., 41., 14.],
       [97.,  8., 79.]])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:
```
'''lets example - number is 6.1 then ceil convert into 7'''
np.ceil(np.random.random((2, 3))*100)
```

Out[ ]:
```
array([[81., 51., 58.],
       [79., 81., 99.]])
```

Indexing and Slicing

In [ ]:
```
a1 = np.arange(10)
a2 = np.arange(12).reshape(3, 4)
a3 = np.arange(8).reshape(2, 2, 2)

print('a1: \n', a1)
print('a2: \n', a2)
print('a3: \n', a3)
```

```
a1:
 [0 1 2 3 4 5 6 7 8 9]
a2:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
a3:
 [[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

In [ ]:
```
# indexing
print(a1)
print('last index number: ', a1[-1])
```

```
[0 1 2 3 4 5 6 7 8 9]
last index number:  9
```

In [ ]:
```
print(a1)
print('first number of array : ', a1[0])
```

```
[0 1 2 3 4 5 6 7 8 9]
first number of array :  0
```

In [ ]:
```
# 2D
print(a2)
print('extracting 6 :', a2[1, 2])   #   a2[row number, column number]
```

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

In [ ]:
```
# 3D
print(a3)
print('extracting 5 :', a3[1, 0, 1]) # a3[2D array number, row , column]
```

```
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
extracting 5 : 5
```

In [ ]:
```
# slicing
a1
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
[2 3 4]
```

In [ ]:
```
# extracting row
print(a2, '\n\n extracting first row:')
a2[0, :]
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

 extracting first column:
```

Out[ ]:
```
array([0, 1, 2, 3])
```

In [ ]:
```
# extracting column
print(a2, '\n\n extracting first column:')
a2[:, 0]
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

 extracting first column:
```

Out[ ]:
```
array([0, 4, 8])
```

In [ ]:
```
# extracting 5 6 and 9 10
a2[1:3, 1:3]    #a2[start row: end row, start col : end col]
```

Out[ ]:
```
array([[ 5,  6],
       [ 9, 10]])
```

In [ ]:
```
# extracting corners
a2[::2, ::3]

# a2[start row : end row : skip row, start col, end col: skip col]
```

Out[ ]:
```
array([[ 0,  3],
       [ 8, 11]])
```

In [ ]:
```
# extracting [1, 3],[9, 11]
a2[::2, 1::2]
```

Out[ ]:
```
array([[ 1,  3],
       [ 9, 11]])
```

In [ ]:
```
# extracting [4, 7]
a2[1, ::3]
```

Out[ ]:
```
array([4, 7])
```

In [ ]:
```
# 3D array
a3 = np.arange(27).reshape(3, 3, 3)
a3
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

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

In [ ]:
```
# extract first and last array in 3D
a3[::2]
```

Out[ ]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

       [[18, 19, 20],
        [21, 22, 23],
        [24, 25, 26]]])
```

In [ ]:
```
# first 2D array of 2nd row
a3[0, 1, :]
# a3[start number of 2D array : end no.of 2D array,star row:end row, start col:end col]
```

Out[ ]:
```
array([3, 4, 5])
```

In [ ]:
```
# extracting [10, 13, 16]
a3[1, :, 1]
```

Out[ ]:
```
array([10, 13, 16])
```

In [ ]:
```
# extracting ([[22, 23],[25, 26]])
a3[2, 1:, 1:]
```

Out[ ]:
```
array([[22, 23],
       [25, 26]])
```

In [ ]:
```
#  extracting ([[0,2], [18, 20]])
a3[::2, 0, ::2]
```

Out[ ]:
```
array([[ 0,  2],
       [18, 20]])
```

Iterating

In [ ]:
```
a1
```

Out[ ]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [ ]:
```
for i in a1:
    print(i)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
0
1
2
3
4
5
6
7
8
9
```

In [ ]:
```python
# for 2D
for i in a2:
    print(i) #every time 1 row printing
```

```
[0 1 2 3]
[4 5 6 7]
[ 8  9 10 11]
```

In [ ]:
```python
# for 3D
for i in a3:
    print(i)  # every time one 2D array printing
```

```
[[0 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]]
```

In [ ]:
```python
# print all the number of 3D array
for i in np.nditer(a3):
    print(i)
```

```
0
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
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Reshaping

In [ ]:
```
a3 = np.arange(27).reshape(3, 3, 3)
a3
```

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

In [ ]:
```
# transpose
print(a2)
np.transpose(a2)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

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

In [ ]:
```
# another syntax of transpose
a2.T
```

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

In [ ]:
```
# ravel -> convert multidimensional array into 1D
a3.ravel()
```

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

Stacking

In [ ]:
```
# horizontal stacking
a4 = np.arange(12).reshape(3, 4)
a5 = np.arange(12, 24).reshape(3, 4)
print('a4 :\n', a4)
print('a5 :\n', a5)
```

```
a4 :
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
a5 :
 [[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

In [ ]:
```
np.hstack((a4, a5))
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[ ]:
```
array([[ 0,  1,  2,  3, 12, 13, 14, 15],
       [ 4,  5,  6,  7, 16, 17, 18, 19],
       [ 8,  9, 10, 11, 20, 21, 22, 23]])
```

In [ ]:
```
np.hstack((a4, a5, a5, a4))
```

Out[ ]:
```
array([[ 0,  1,  2,  3, 12, 13, 14, 15, 12, 13, 14, 15,  0,  1,  2,  3],
       [ 4,  5,  6,  7, 16, 17, 18, 19, 16, 17, 18, 19,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 20, 21, 22, 23, 20, 21, 22, 23,  8,  9, 10, 11]])
```

In [ ]:
```
# vertical stack
np.vstack((a4, a5))
```

Out[ ]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

Splitting

In [ ]:
```
# horizontal splitting
np.hsplit(a4, 2) # 2 means  split 2D array in 2 equal parts
```

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

In [ ]:
```
# vertical splitting
np.vsplit(a5, 3)
```

Out[ ]:
```
[array([[12, 13, 14, 15]]),
 array([[16, 17, 18, 19]]),
 array([[20, 21, 22, 23]])]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js