stackoverlow most voted questions regading numpy

How to print the full NumPy array, without truncation?

In [1]:
```
# 1
import sys
import numpy
numpy.set_printoptions(threshold=sys.maxsize)
```

In [2]:
```
# 2
import numpy as np
np.set_printoptions(threshold=np.inf)
```

Dump a NumPy array into a csv file

In [3]:
```
import numpy
a = numpy.asarray([ [1,2,3], [4,5,6], [7,8,9] ])
numpy.savetxt("foo.csv", a, delimiter=",")
```

How can the Euclidean distance be calculated with NumPy?

In [4]:
```
from scipy.spatial import distance
a = (1, 2, 3)
b = (4, 5, 6)
dist = distance.euclidean(a, b)
dist
```

Out[4]: 5.196152422706632

How do I get indices of N maximum values in a NumPy array?

In [5]:
```
import numpy as np

arr = np.array([1, 3, 2, 4, 5])
arr
```

Out[5]: array([1, 3, 2, 4, 5])

In [6]:
```
arr.argsort()[-3:][::-1]
```

Out[6]: array([4, 3, 1], dtype=int64)

In [7]:
```
arr[arr.argsort()[-3:][::-1]]
```

Out[7]:
```
array([5, 4, 3])
```

In [8]:
```
a = np.array([9, 4, 4, 3, 3, 9, 0, 4, 6, 0])
a
```

Out[8]:
```
array([9, 4, 4, 3, 3, 9, 0, 4, 6, 0])
```

In [9]:
```
ind = np.argpartition(a, -4)[-4:]
ind
```

Out[9]:
```
array([1, 5, 8, 0], dtype=int64)
```

In [10]:
```
a[ind]
```

Out[10]:
```
array([4, 9, 6, 9])
```

Convert pandas dataframe to NumPy array

In [11]:
```
import numpy as np
import pandas as pd

index = [1, 2, 3, 4, 5, 6, 7]
a = [np.nan, np.nan, np.nan, 0.1, 0.1, 0.1, 0.1]
b = [0.2, np.nan, 0.2, 0.2, 0.2, np.nan, np.nan]
c = [np.nan, 0.5, 0.5, np.nan, 0.5, 0.5, np.nan]
df = pd.DataFrame({'A': a, 'B': b, 'C': c}, index=index)
df
```

Out[11]:

	A	B	C
1	NaN	0.2	NaN
2	NaN	NaN	0.5
3	NaN	0.2	0.5
4	0.1	0.2	NaN
5	0.1	0.2	0.5
6	0.1	NaN	0.5
7	0.1	NaN	NaN

In [12]:
```
1 df.values
```

Out[12]:
```
array([[nan, 0.2, nan],
       [nan, nan, 0.5],
       [nan, 0.2, 0.5],
       [0.1, 0.2, nan],
       [0.1, 0.2, 0.5],
       [0.1, nan, 0.5],
       [0.1, nan, nan]])
```

Is there a NumPy function to return the first index of something in an array?

In [13]:
```
1 t = np.array([1, 1, 1, 2, 2, 3, 8, 3, 8, 8])
2 np.nonzero(t == 8)
```

Out[13]:
```
(array([6, 8, 9], dtype=int64),)
```

In [14]:
```
1 np.nonzero(t == 8)[0][0]
```

Out[14]:
```
6
```

What does -1 mean in numpy reshape?

In [15]:
```
1 z = np.array([[1, 2, 3, 4],
2          [5, 6, 7, 8],
3          [9, 10, 11, 12]])
4 z.shape
```

Out[15]:
```
(3, 4)
```

In [16]:
```
1 z.reshape(-1)
```

Out[16]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [17]:
```
1 z.reshape(-1).shape
```

Out[17]:
```
(12,)
```

In [18]:
```
1 z.reshape(-1,1)
```

Out[18]:
```
array([[ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11],
       [12]])
```

In [19]:
```
1 z.reshape(-1,1).shape
```

Out[19]:
```
(12, 1)
```

In [20]:
```
1 z.reshape(-1, 2)
```

Out[20]:
```
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12]])
```

In [21]:
```
1 z.reshape(-1, 2).shape
```

Out[21]:
```
(6, 2)
```

In [22]:
```
1 z.reshape(1,-1)
```

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

In [23]:
```
1 z.reshape(1,-1).shape
```

Out[23]:
```
(1, 12)
```

In [24]:
```
1 z.reshape(2, -1)
```

Out[24]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
```

In [25]:
```
1 z.reshape(2, -1).shape
```

Out[25]:
```
(2, 6)
```

In [26]:
```
z.reshape(3, -1)
```

Out[26]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [27]:
```
z.reshape(3, -1).shape
```

Out[27]:
```
(3, 4)
```

What are the advantages of NumPy over regular Python lists?

NumPy is not just more efficient; it is also more convenient. You get a lot of vector and matrix operations for free, which sometimes allow one to avoid unnecessary work. And they are also efficiently implemented.

For example, you could read your cube directly from a file into an array:

x = numpy.fromfile(file=open("data"), dtype=float).reshape((100, 100, 100))

Sum along the second dimension:

s = x.sum(axis=1)

Find which cells are above a threshold:

(x > 0.5).nonzero()

Remove every even-indexed slice along the third dimension:

x[:, :, ::2]

Iso, many useful libraries work with NumPy arrays. For example, statistical analysis and visualization libraries.

Even if you don't have performance problems, learning NumPy is worth the effort.

How to count the occurrence of certain item in an ndarray?

In [28]:
```
a = numpy.array([0, 3, 0, 1, 0, 1, 2, 1, 0, 0, 0, 0, 1, 3, 4])
a
```

Out[28]:
```
array([0, 3, 0, 1, 0, 1, 2, 1, 0, 0, 0, 0, 1, 3, 4])
```

In [29]:
```
unique, counts = numpy.unique(a, return_counts=True)
print(unique, counts)
```

```
[0 1 2 3 4] [7 4 1 2 1]
```

In [30]:
```
dict(zip(unique, counts))
```

Out[30]:
```
{0: 7, 1: 4, 2: 1, 3: 2, 4: 1}
```

In [31]:
```
import collections, numpy
a = numpy.array([0, 3, 0, 1, 0, 1, 2, 1, 0, 0, 0, 0, 1, 3, 4])
collections.Counter(a)

```

Out[31]:
```
Counter({0: 7, 3: 2, 1: 4, 2: 1, 4: 1})
```

In [32]:
```
y = np.array([1, 2, 2, 2, 2, 0, 2, 3, 3, 3, 0, 0, 2, 2, 0])
np.count_nonzero(y == 1)
```

Out[32]:
```
1
```

In [33]:
```
np.count_nonzero(y == 2)
```

Out[33]:
```
7
```

In [34]:
```
np.count_nonzero(y == 3)
```

Out[34]:
```
3
```

Most efficient way to map function over numpy array

In [35]:
```
import numpy as np
x = np.array([1, 2, 3, 4, 5])
squarer = lambda t: t ** 2
vfunc = np.vectorize(squarer)
vfunc(x)
```

Out[35]:
```
array([ 1,  4,  9, 16, 25])
```

Numpy array dimensions

In [36]:
```
import numpy as np
a = np.array([[1,2],[1,2]])
a
```

Out[36]:
```
array([[1, 2],
       [1, 2]])
```

In [37]:
```
1 a.shape
```

Out[37]:
```
(2, 2)
```

In [38]:
```
1 a.ndim  # num of dimensions/axes, *Mathematics definition of dimension*
```

Out[38]:
```
2
```

In [39]:
```
1 np.shape(a)
```

Out[39]:
```
(2, 2)
```

In [40]:
```
1 a.shape[0]
```

Out[40]:
```
2
```

In [41]:
```
1 a.shape[1]
```

Out[41]:
```
2
```

Sorting arrays in NumPy by column

In [42]:
```
1 a = np.array([[1,2,3],[4,5,6],[0,0,1]])
2 a
```

Out[42]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [0, 0, 1]])
```

In [43]:
```
1 sorted(a, key=lambda a_entry: a_entry[1])
```

Out[43]:
```
[array([0, 0, 1]), array([1, 2, 3]), array([4, 5, 6])]
```

Find nearest value in numpy array

In [44]:
```
1 a = numpy.array([0, 3, 0, 1, 0, 1, 2, 1, 0, 0, 0, 0, 1, 3, 4])
2 a
```

Out[44]:
```
array([0, 3, 0, 1, 0, 1, 2, 1, 0, 0, 0, 0, 1, 3, 4])
```

In [45]:
```
1 def find_nearest(array, value):
2     array = np.asarray(array)
3     idx = (np.abs(array - value)).argmin()
4     return array[idx]
5
```

In [46]:
```
print(find_nearest(a, 0.8))
```

```
1
```

How to pretty-print a numpy.array without scientific notation and with given precision?

In [47]:
```
import numpy as np
x=np.random.random(10)
print(x)
```

```
[0.28511141 0.88137162 0.8176507  0.81117071 0.46760709 0.38769271
 0.58014007 0.35466982 0.68255276 0.83208039]
```

In [48]:
```
np.set_printoptions(precision=3)
print(x)
```

```
[0.285 0.881 0.818 0.811 0.468 0.388 0.58  0.355 0.683 0.832]
```

In [49]:
```
y=np.array([1.5e-10,1.5,1500])
print(y)
```

```
[1.5e-10 1.5e+00 1.5e+03]
```

In [50]:
```
np.set_printoptions(suppress=True)
print(y)
```

```
[   0.     1.5 1500. ]
```

In [51]:
```
x = np.random.random(10)
x
```

Out[51]:
```
array([0.587, 0.053, 0.755, 0.221, 0.395, 0.668, 0.489, 0.821, 0.333,
       0.349])
```

In [52]:
```
with np.printoptions(precision=3, suppress=True):
    print(x)
```

```
[0.587 0.053 0.755 0.221 0.395 0.668 0.489 0.821 0.333 0.349]
```

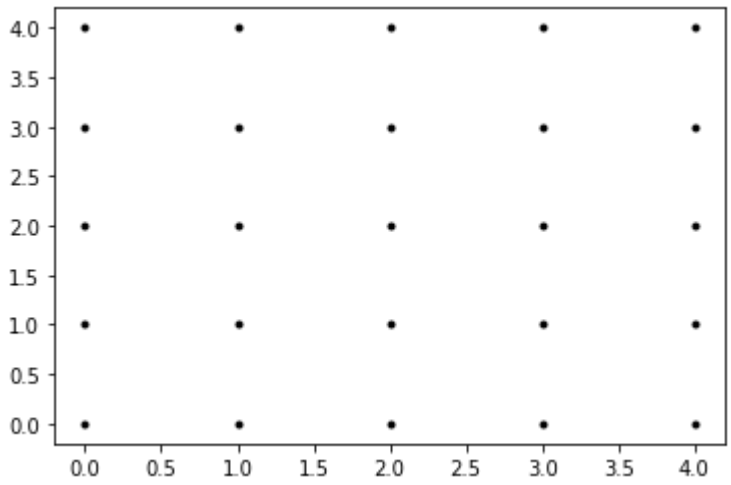
What is the purpose of meshgrid in Python / NumPy?

In [53]:
```
xvalues = np.array([0, 1, 2, 3, 4]);
yvalues = np.array([0, 1, 2, 3, 4]);
```

In [54]:
```
%matplotlib inline
from matplotlib import pyplot as plt

xx, yy = np.meshgrid(xvalues, yvalues)
plt.plot(xx, yy, marker='.', color='k', linestyle='none')
```

Out[54]:
```
[<matplotlib.lines.Line2D at 0x1a6f1197b48>,
 <matplotlib.lines.Line2D at 0x1a6f11a1d88>,
 <matplotlib.lines.Line2D at 0x1a6f11a1808>,
 <matplotlib.lines.Line2D at 0x1a6f11a1f48>,
 <matplotlib.lines.Line2D at 0x1a6f11a8188>]
```



Difference between numpy.array shape (R, 1) and (R,)

In [55]:
```
# 1 dimension with 2 elements, shape = (2,).
# Note there's nothing after the comma.
z=np.array([  # start dimension
    10,       # not a dimension
    20        # not a dimension
])            # end dimension
print(z.shape)
```

```
(2,)
```

In [56]:
```
# 2 dimensions, each with 1 element, shape = (2,1)
w=np.array([  # start outer dimension
    [10],     # element is in an inner dimension
    [20]      # element is in an inner dimension
])            # end outer dimension
print(w.shape)
```

```
(2, 1)
```

What does numpy.random.seed(0) do?

In [57]:
```
# np.random.seed(0) makes the random numbers predictable
numpy.random.seed(0); numpy.random.rand(4)
```

Out[57]:
```
array([0.549, 0.715, 0.603, 0.545])
```

In [58]:
```
numpy.random.seed(0); numpy.random.rand(4)
```

Out[58]:
```
array([0.549, 0.715, 0.603, 0.545])
```

In [59]:
```
numpy.random.rand(4)
```

Out[59]:
```
array([0.424, 0.646, 0.438, 0.892])
```

In [60]:
```
numpy.random.rand(4)
```

Out[60]:
```
array([0.964, 0.383, 0.792, 0.529])
```

How do I create an empty array/matrix in NumPy?

In [61]:
```
a = numpy.zeros(shape=(5,2))
a
```

Out[61]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

In [62]:
```
a[0] = [1,2]
a[1] = [2,3]
a
```

Out[62]:
```
array([[1., 2.],
       [2., 3.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

How to convert 2D float numpy array to 2D int numpy array?

In [63]:
```
x = np.array([[1.0, 2.3], [1.3, 2.9]])
x
```

Out[63]:
```
array([[1. , 2.3],
       [1.3, 2.9]])
```

In [64]:
```
1 x.astype(int)
```

Out[64]:
```
array([[1, 2],
       [1, 2]])
```

In [65]:
```
1 x = np.array([[1.0,2.3],[1.3,2.9]])
2 x
```

Out[65]:
```
array([[1. , 2.3],
       [1.3, 2.9]])
```

In [66]:
```
1 y = np.trunc(x)
2 y
```

Out[66]:
```
array([[1., 2.],
       [1., 2.]])
```

In [67]:
```
1 z = np.ceil(x)
2 z
```

Out[67]:
```
array([[1., 3.],
       [2., 3.]])
```

In [68]:
```
1 t = np.floor(x)
2 t
```

Out[68]:
```
array([[1., 2.],
       [1., 2.]])
```

In [69]:
```
1 a = np.rint(x)
2 a
```

Out[69]:
```
array([[1., 2.],
       [1., 3.]])
```

In [70]:
```
1 a.astype(int)
```

Out[70]:
```
array([[1, 2],
       [1, 3]])
```

In [71]:
```
1  y.astype(int)
```

Out[71]:
```
array([[1, 2],
       [1, 2]])
```

In [72]:
```
1 np.int_(y)
```

Out[72]:
```
array([[1, 2],
       [1, 2]])
```

How to add an extra column to a NumPy array

In [73]:
```
import numpy as np
N = 10
a = np.random.rand(N,N)
print(a.shape)
b = np.zeros((N,N+1))
print(b.shape)
b[:,:-1] = a
print(b.shape)
```

```
(10, 10)
(10, 11)
(10, 11)
```

What is the difference between Numpy's array() and asarray() functions?

In [74]:
```
A = numpy.matrix(numpy.ones((3,3)))
A
```

Out[74]:
```
matrix([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]])
```

In [75]:
```
numpy.array(A)[2]=2
A
```

Out[75]:
```
matrix([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]])
```

In [76]:
```
numpy.asarray(A)[2]=2
A
```

Out[76]:
```
matrix([[1., 1., 1.],
        [1., 1., 1.],
        [2., 2., 2.]])
```

Converting NumPy array into Python List structure?

In [77]:
```
np.array([[1,2,3],[4,5,6]]).tolist()
```

Out[77]:
```
[[1, 2, 3], [4, 5, 6]]
```

Converting between datetime, Timestamp and datetime64

In [78]:
```
from datetime import datetime
import numpy as np
dt = datetime.utcnow()
dt
```

Out[78]:
```
datetime.datetime(2021, 2, 12, 6, 9, 50, 275075)
```

In [79]:
```
dt64 = np.datetime64(dt)
dt64
```

Out[79]:
```
numpy.datetime64('2021-02-12T06:09:50.275075')
```

In [80]:
```
ts = (dt64 - np.datetime64('1970-01-01T00:00:00Z')) / np.timedelta64(1, 's')
ts
```

```
D:\anaconda\lib\site-packages\ipykernel_launcher.py:1: DeprecationWarning: pars
ing timezone aware datetimes is deprecated; this will raise an error in the fut
ure
  """Entry point for launching an IPython kernel.
```

Out[80]:
```
1613110190.275075
```

In [81]:
```
datetime.utcfromtimestamp(ts)
```

Out[81]:
```
datetime.datetime(2021, 2, 12, 6, 9, 50, 275075)
```

What is the difference between flatten and ravel functions in numpy?

In [82]:
```
import numpy
a = numpy.array([[1,2],[3,4]])

r = numpy.ravel(a)
f = numpy.ndarray.flatten(a)

print(id(a))
print(id(r))
print(id(f))

print(r)
print(f)

print("\nbase r:", r.base)
print("\nbase f:", f.base)

```

```
1816516103424
1816516007056
1816516006576
[1 2 3 4]
[1 2 3 4]

base r: [[1 2]
 [3 4]]

base f: None
```

What does axis in pandas mean?

axis=0 means each row as a bulk

axis=1 means each column as a bulk

How do I check which version of NumPy I'm using?

In [83]:
```
numpy.version.version
```

Out[83]:
```
'1.16.5'
```

Concatenating two one-dimensional NumPy arrays

In [84]:
```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
print(a)
print("\n")
print(b)
print("\n")
# Appending below last row
print(np.concatenate((a, b), axis=0))

print("\n")
# Appending after last column
print(np.concatenate((a, b.T), axis=1))    # Notice the transpose
print("\n")
# Flattening the final array
print(np.concatenate((a, b), axis=None))

```

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


[[5 6]]


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


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


[1 2 3 4 5 6]
```

Most efficient way to reverse a numpy array

In [85]:
```
a
```

Out[85]:
```
array([[1, 2],
       [3, 4]])
```

In [86]:
```
a[::-1]
```

Out[86]:
```
array([[3, 4],
       [1, 2]])
```

What is the difference between ndarray and array in numpy?

numpy.array is a function that returns a numpy.ndarray. There is no object type numpy.array.

Converting numpy dtypes to native python types

In [87]:
```
import numpy as np

# for example, numpy.float32 -> python float
val = np.float32(0)
pyval = val.item()
print(type(pyval))         # <class 'float'>

# and similar...
type(np.float64(0).item()) # <class 'float'>
type(np.uint32(0).item())  # <class 'long'>
type(np.int16(0).item())   # <class 'int'>
type(np.cfloat(0).item())  # <class 'complex'>
type(np.datetime64(0, 'D').item())  # <class 'datetime.date'>
type(np.datetime64('2001-01-01 00:00:00').item())  # <class 'datetime.dateti
type(np.timedelta64(0, 'D').item()) # <class 'datetime.timedelta'>
```

```
<class 'float'>
```

Out[87]:
```
datetime.timedelta
```

Better way to shuffle two numpy arrays in unison

In [88]:
```
a = numpy.array([[[  0.,   1.,   2.],
                  [  3.,   4.,   5.]],

                 [[  6.,   7.,   8.],
                  [  9.,  10.,  11.]],

                 [[ 12.,  13.,  14.],
                  [ 15.,  16.,  17.]]])

b = numpy.array([[ 0.,  1.],
                 [ 2.,  3.],
                 [ 4.,  5.]])
```

In [89]:
```
#We can now construct a single array containing all the data:
c = numpy.c_[a.reshape(len(a), -1), b.reshape(len(b), -1)]
c
```

Out[89]:
```
array([[ 0.,  1.,  2.,  3.,  4.,  5.,  0.,  1.],
       [ 6.,  7.,  8.,  9., 10., 11.,  2.,  3.],
       [12., 13., 14., 15., 16., 17.,  4.,  5.]])
```

In [90]:
```
a2 = c[:, :a.size//len(a)].reshape(a.shape)
a2
```

Out[90]:
```
array([[[ 0.,  1.,  2.],
        [ 3.,  4.,  5.]],

       [[ 6.,  7.,  8.],
        [ 9., 10., 11.]],

       [[12., 13., 14.],
        [15., 16., 17.]]])
```

In [91]:
```
b2 = c[:, a.size//len(a):].reshape(b.shape)
b2
```

Out[91]:
```
array([[0., 1.],
       [2., 3.],
       [4., 5.]])
```

NumPy array initialization (fill with identical values)

In [92]:
```
np.full((3, 5), 7)
```

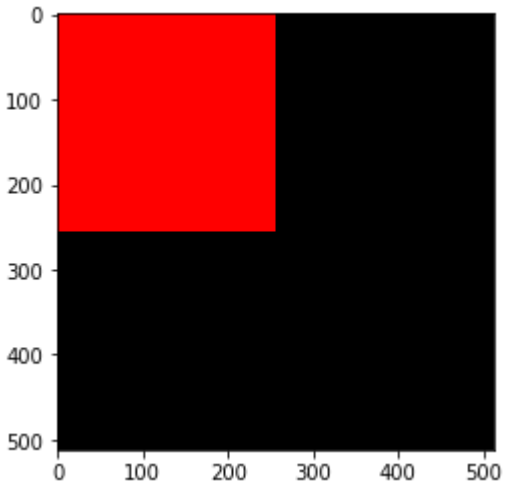
Out[92]:
```
array([[7, 7, 7, 7, 7],
       [7, 7, 7, 7, 7],
       [7, 7, 7, 7, 7]])
```

How do I convert a numpy array to (and display) an image?

In [93]:
```
from PIL import Image
import numpy as np

w, h = 512, 512
data = np.zeros((h, w, 3), dtype=np.uint8)
data[0:256, 0:256] = [255, 0, 0] # red patch in upper left
img = Image.fromarray(data, 'RGB')
# img.save('my.png')
img.show()
```

In [94]:
```
from matplotlib import pyplot as plt
plt.imshow(data, interpolation='nearest')
plt.show()
```



dropping infinite values from dataframes in pandas?

In [95]:
```
df = pd.DataFrame([1, 2, np.inf, -np.inf])
df
```

Out[95]:

	0
0	1.0
1	2.0
2	inf
3	-inf

In [96]:
```
1 df.replace([np.inf, -np.inf], np.nan)
```

Out[96]:

	0
0	1.0
1	2.0
2	NaN
3	NaN

In [ ]:
```
1
```