Προγραμματισμός με τη γλώσσα python

Alexandros Kanterakis kantale@ics.forth.gr (kantale@ics.forth.gr)

Διάλεξη 8η, Τρίτη 3 Δεκεμβρίου 2019

numpy

Η numpy (https://en.wikipedia.org/wiki/NumPy) είναι μία βιβλιοθήκη της python για τον χειρισμό ν-διδιάστατων αιρθμητικών πινάκων:

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

In [557]:
```
a = np.array([[1,2,3], [4,5,6]])
a
```

Out[557]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

Μπορούμε να προσπελάσουμε ένα στοιχείο του πίνακα ως εξής:

In [501]:
```
a[1,2]
```

Out[501]:
```
6
```

Ομοίως μπορούμε να αλλάξουμε ένα στοιχείο του πίνακα:

In [502]:
```
a[1,2] = 10
a
```

Out[502]:
```
array([[ 1,  2,  3],
       [ 4,  5, 10]])
```

Το μέγεθος του πίνακα ανά διάσταση:

In [4]:
```
a.shape
```

Out[4]:
```
(2, 3)
```

Το πλήθος των διαστάσεων:

In [7]:
```
a.ndim
```

Out[7]:
```
2
```

Το πλήθος των στοιχείων του πίνακα:

In [9]:
```
a.size
```

Out[9]:
```
6
```

Πίνακας μόνο με 1

```
In [15]: np.ones([3,5])
```

```
Out[15]: array([[ 1.,  1.,  1.,  1.,  1.],
                [ 1.,  1.,  1.,  1.,  1.],
                [ 1.,  1.,  1.,  1.,  1.]])
```

Πίνακας μόνο με 0

```
In [18]: np.zeros([3,5])
```

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

Ένας πίνακας άδειος ο οποίο περιέχει "σκουπίδια" (δηλαδή διάφορες τιμές από προηγούμενη χρήση της μνήμης):

```
In [571]: np.empty([3,5])
```

```
Out[571]: array([[ 0.69209415,  0.74405598,  0.58994422,  0.51903866,  0.93660742],
                 [ 0.69209415,  0.74405598,  0.58994422,  0.51903866,  0.93660742],
                 [ 0.42445219,  0.54983743,  0.16428273,  0.71051264,  0.13425249]])
```

Η `arange` είναι σαν τη range με τη διαφορά ότι όλες οι τιμές μπορούν να είναι και δεκαδικές:

```
In [21]: np.arange(1,10,0.3)
```

```
Out[21]: array([ 1. ,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  4. ,
                 4.3,  4.6,  4.9,  5.2,  5.5,  5.8,  6.1,  6.4,  6.7,  7. ,  7.3,
                 7.6,  7.9,  8.2,  8.5,  8.8,  9.1,  9.4,  9.7])
```

Μπορούμε φυσικά να κάνουμε και iterate (δηλαδή να εφαρμόσουμε τη for) σε έναν πίνακα. Κάθε φορά που το κάνουμε όμως αυτό ένα μικρό κουνελάκι στενοχωριέται πάρα πολύ:

In [23]:
```
for x in np.arange(1,10,.3):
    print(x)
```

```
1.0
1.3
1.6
1.9
2.2
2.5
2.8
3.1
3.4
3.7
4.0
4.3
4.6
4.9
5.2
5.5
5.8
6.1
6.4
6.7
7.0
7.3
7.6
7.9
8.2
8.5
8.8
9.1
9.4
9.7
```

Ο λόγος για αυτό είναι ότι η numpy περιέχει εκατοντάδες συναρτήσεις για πολύ γρήγορη διαχείρηση και μετασχηματισμό πινάκων. Οπότε πριν μπείτε στον πειρασμό να κάνετε κάποιο for, τουλάχιστον κάντε ένα search στο google αν υπάρχει κάποια συνάρτηση που το κάνει..

Εκτός από την arange υπάρχει και η linspace η οποία επίσης δημιουργεί μία αριθμητική πρόοδο από έναν αριθμό μέχρι κάποιον άλλον. Η διαφορά είναι ότι στη linspace η τρίτη παράμετρος δηλώνει πόσους αριθμούς θέλουμε να έχει η πρόοδο (ενω στην arange δηλώνει το βήμα της προόδου)

In [26]:
```
np.linspace(1,10,20)
```

Out[26]:
```
array([  1.        ,   1.47368421,   1.94736842,   2.42105263,
         2.89473684,   3.36842105,   3.84210526,   4.31578947,
         4.78947368,   5.26315789,   5.73684211,   6.21052632,
         6.68421053,   7.15789474,   7.63157895,   8.10526316,
         8.57894737,   9.05263158,   9.52631579,  10.        ])
```

Τυχαίοι αριθμοί:

In [30]:
```
np.random.random([4,5])
```

Out[30]:
```
array([[ 0.73041476,  0.85627737,  0.1132018 ,  0.74100275,  0.82242056],
       [ 0.66131635,  0.16408397,  0.00490488,  0.40217054,  0.2286295 ],
       [ 0.38351069,  0.3711942 ,  0.88431264,  0.55911999,  0.06282348],
       [ 0.53250803,  0.77532521,  0.23247395,  0.28375013,  0.36304702]])
```

Στη numpy μπορούμε να εφαρμόσουμε συναρτήσεις οι οποίες παίρνουν μία παράμετρο σε όλον τον πίνακα:

In [31]:
```
a
```

Out[31]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [32]:
```
np.sin(a)
```

Out[32]:
```
array([[ 0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ]])
```

In [33]:
```
np.exp(a)
```

Out[33]:
```
array([[   2.71828183,    7.3890561 ,   20.08553692],
       [  54.59815003,  148.4131591 ,  403.42879349]])
```

Προσοχή! η np.log εφαρμόζει τον φυσικό λογάριθμο (ln)

In [34]:
```
np.log(a) # ln e
```

Out[34]:
```
array([[ 0.        ,  0.69314718,  1.09861229],
       [ 1.38629436,  1.60943791,  1.79175947]])
```

Ενώ ο δεκαδικός λαγάριθμος είναι:

In [36]:
```
np.log10(a)
```

Out[36]:
```
array([[ 0.        ,  0.30103   ,  0.47712125],
       [ 0.60205999,  0.69897   ,  0.77815125]])
```

Σταθερές:

In [40]:
```
np.pi
```

Out[40]:
```
3.141592653589793
```

In [42]:
```
np.e
```

Out[42]:
```
2.718281828459045
```

Πράξεις στοιχείο-προς-στοιχείο (ή αλλιώς elementwise) μεταξύ πινάκων:

In [561]:
```
a=np.array([[1,2,3], [4,5,6]])
b=np.array([[4,1,0], [5,0,1]])
```

In [45]:
```
a+b
```

Out[45]:
```
array([[5, 3, 3],
       [9, 5, 7]])
```

In [46]:
```
a*b
```

Out[46]:
```
array([[ 4,  2,  0],
       [20,  0,  6]])
```

Πολλαπλασιασμός πινάκων:

In [49]:
```
np.dot(a,b.T)
```

Out[49]:
```
array([[ 6,  8],
       [21, 26]])
```

το a.T μας δίνει τον ανάστροφο πίνακα. (**ΠΡΟΣΟΧΗ!** όχι ο αντίστροφος!)

In [558]:
```
a
```

Out[558]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [559]:
```
a.T
```

Out[559]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

Συνώνυμη της a.T είναι η transpose:

In [560]:
```
a.transpose()
```

Out[560]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

Ο πολλαπλασιαμός πινάκων γίνεται με την dot:

In [53]:
```
a.dot(b.T)
```

Out[53]:
```
array([[ 6,  8],
       [21, 26]])
```

ή αλλιώς:

In [562]:
```
np.dot(a,b.T)
```

Out[562]:
```
array([[ 6,  8],
       [21, 26]])
```

Με τη reshape αλλάζουμε τις διαστάσεις ενός πίνακα:

In [54]:
```
a
```

Out[54]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [56]:
```
a.reshape(3,2)
```

Out[56]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [57]:
```
a.reshape(6,1)
```

Out[57]:
```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])
```

In [422]:
```
a.reshape(1,6)
```

Out[422]:
```
array([[1, 2, 3, 4, 5, 6]])
```

Αν κάποια παράμετρος της reshape είναι -1 τότε υπολογίζει αυτόματα την τιμή της (αν μπορεί):

In [423]:
```
np.random.random((6,2)).reshape(3,-1) # Από 6 γραμμές πάμε σε 3:
```

Out[423]:
```
array([[ 0.04903716,  0.9785132 ,  0.84331155,  0.94860724],
       [ 0.85665382,  0.20847732,  0.24546295,  0.6111018 ],
       [ 0.07518856,  0.42283486,  0.47115603,  0.14732587]])
```

In [424]:
```
np.linspace(1,10,20).reshape(-1,10)# Από μία γραμμή πάμε σε 2
```

Out[424]:
```
array([[  1.        ,   1.47368421,   1.94736842,   2.42105263,
          2.89473684,   3.36842105,   3.84210526,   4.31578947,
          4.78947368,   5.26315789],
       [  5.73684211,   6.21052632,   6.68421053,   7.15789474,
          7.63157895,   8.10526316,   8.57894737,   9.05263158,
          9.52631579,  10.        ]])
```

Προσοχή! Ο πίνακας δεν αλλάζει με την reshape. Αν θέλουμε να αλλάξουμε τον πίνακα τότε χρησιμοποιούμε τη reshape:

In [533]:
```
a = np.random.random((2,3))
a
```

Out[533]:
```
array([[ 0.86501027,  0.87671615,  0.86737332],
       [ 0.84734983,  0.14904097,  0.82635533]])
```

In [534]:
```
a.reshape(3,2)
```

Out[534]:
```
array([[ 0.86501027,  0.87671615],
       [ 0.86737332,  0.84734983],
       [ 0.14904097,  0.82635533]])
```

In [535]:
```
a # Δεν έχει αλλάξει
```

Out[535]:
```
array([[ 0.86501027,  0.87671615,  0.86737332],
       [ 0.84734983,  0.14904097,  0.82635533]])
```

In [536]:
```
a.resize(3,2)
```

In [537]:
```
a # Έχει αλλάξει
```

Out[537]:
```
array([[ 0.86501027,  0.87671615],
       [ 0.86737332,  0.84734983],
       [ 0.14904097,  0.82635533]])
```

Η min, max, sum εφαρμόζεται σε όλο τον πίνακα:

In [538]:
```
a
```

Out[538]:
```
array([[ 0.86501027,  0.87671615],
       [ 0.86737332,  0.84734983],
       [ 0.14904097,  0.82635533]])
```

In [539]:
```
a.min()
```

Out[539]:
```
0.14904097385101789
```

Αν δώσουμε την παράμετρο axis, τότε βρίσκει όλες τις τιμές ξεχωριστά για αυτή τη διάσταση:

In [540]:
```
a.min(axis=0)
```

Out[540]:
```
array([ 0.14904097,  0.82635533])
```

In [541]:
```
a.min(axis=1)
```

Out[541]:
```
array([ 0.86501027,  0.84734983,  0.14904097])
```

In [542]:
```
a
```

Out[542]:
```
array([[ 0.86501027,  0.87671615],
       [ 0.86737332,  0.84734983],
       [ 0.14904097,  0.82635533]])
```

In [543]:
```
a.sum()
```

Out[543]:
```
4.4318458782793408
```

In [544]:
```
a.sum(axis=0)
```

Out[544]:
```
array([ 1.88142456,  2.55042132])
```

In [545]:
```
a.sum(axis=1)
```

Out[545]:
```
array([ 1.74172642,  1.71472315,  0.97539631])
```

Η argmin και argmax επιστρέφει το index του μικρότερου (ή μεγαλύτερου):

In [546]:
```
a.argmin()
```

Out[546]:
```
4
```

In [547]:
```
a.argmin(axis=0)
```

Out[547]:
```
array([2, 2])
```

In [548]:
```
a.argmin(axis=1)
```

Out[548]:
```
array([0, 1, 0])
```

Όπως και με τις λίστες, έτσι και με τους πίνακες της numpy μπορούμε να βάλουμε διαστήματα στα indexes.

In [82]:
```
b = [1,2,3,4,5,6,7,8,9,10]
```

In [81]:
```
type(a)
```

Out[81]:
```
numpy.ndarray
```

In [84]:
```
type(b)
```

Out[84]:
```
list
```

In [86]:
```
b[5:]
```

Out[86]:
```
[6, 7, 8, 9, 10]
```

In [87]:
```
a
```

Out[87]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [427]:
```
a = np.random.random([10,5])
```

In [428]:
```
a
```

Out[428]:
```
array([[ 0.69209415,  0.74405598,  0.58994422,  0.51903866,  0.93660742],
       [ 0.42445219,  0.54983743,  0.16428273,  0.71051264,  0.13425249],
       [ 0.42144851,  0.69445979,  0.40491544,  0.46205062,  0.76795599],
       [ 0.37588999,  0.63994809,  0.28475934,  0.21667052,  0.07422019],
       [ 0.01741184,  0.28740475,  0.56879267,  0.63534581,  0.0612609 ],
       [ 0.9298351 ,  0.35012857,  0.51996718,  0.44845842,  0.92092282],
       [ 0.12143151,  0.94833192,  0.2439955 ,  0.44217524,  0.45783427],
       [ 0.83292434,  0.90528182,  0.26152684,  0.46834753,  0.30246709],
       [ 0.03192285,  0.70764385,  0.94167941,  0.02025402,  0.69930778],
       [ 0.59585316,  0.7778326 ,  0.09498829,  0.45896575,  0.16663657]])
```

Μπορούμε όμως να βάλουμε διαφορετικά διαστήματα σε κάθε διάσταση. Π.χ.: Οι γραμμές 2,3,4 και όλες οι στήλες:

In [429]:
```
a[1:4,:]
```

Out[429]:
```
array([[ 0.42445219,  0.54983743,  0.16428273,  0.71051264,  0.13425249],
       [ 0.42144851,  0.69445979,  0.40491544,  0.46205062,  0.76795599],
       [ 0.37588999,  0.63994809,  0.28475934,  0.21667052,  0.07422019]])
```

Οι γραμμές 2,3,4 και οι στήλες 1,2:

In [430]:
```
a[1:4, 0:2]
```

Out[430]:
```
array([[ 0.42445219,  0.54983743],
       [ 0.42144851,  0.69445979],
       [ 0.37588999,  0.63994809]])
```

Οι γραμμές από 2 μέχρι τη 5 (χωρίς τη 5) με βήμα 2 και οι στήλες 1,2:

In [431]:
```
a[1:4:2, 0:2]
```

Out[431]:
```
array([[ 0.42445219,  0.54983743],
       [ 0.37588999,  0.63994809]])
```

Οι γραμμές 5,4,3 και οι στήλες 1,2:

In [432]:
```
a[4:1:-1, 0:2]
```

Out[432]:
```
array([[ 0.01741184,  0.28740475],
       [ 0.37588999,  0.63994809],
       [ 0.42144851,  0.69445979]])
```

Μπορούμε επίσης να δηλώσουμε συγκεκριμένες γραμμές (ή στήλες), αντί για διαστήματα:

In [433]:
```
a
```

Out[433]:
```
array([[ 0.69209415,  0.74405598,  0.58994422,  0.51903866,  0.93660742],
       [ 0.42445219,  0.54983743,  0.16428273,  0.71051264,  0.13425249],
       [ 0.42144851,  0.69445979,  0.40491544,  0.46205062,  0.76795599],
       [ 0.37588999,  0.63994809,  0.28475934,  0.21667052,  0.07422019],
       [ 0.01741184,  0.28740475,  0.56879267,  0.63534581,  0.0612609 ],
       [ 0.9298351 ,  0.35012857,  0.51996718,  0.44845842,  0.92092282],
       [ 0.12143151,  0.94833192,  0.2439955 ,  0.44217524,  0.45783427],
       [ 0.83292434,  0.90528182,  0.26152684,  0.46834753,  0.30246709],
       [ 0.03192285,  0.70764385,  0.94167941,  0.02025402,  0.69930778],
       [ 0.59585316,  0.7778326 ,  0.09498829,  0.45896575,  0.16663657]])
```

In [434]:
```
a[[1, 5, 9],:]
```

Out[434]:
```
array([[ 0.42445219,  0.54983743,  0.16428273,  0.71051264,  0.13425249],
       [ 0.9298351 ,  0.35012857,  0.51996718,  0.44845842,  0.92092282],
       [ 0.59585316,  0.7778326 ,  0.09498829,  0.45896575,  0.16663657]])
```

Αυτό είναι ισοδύναμο με:

In [435]:
```
a[[1, 5, 9],]
```

Out[435]:
```
array([[ 0.42445219,  0.54983743,  0.16428273,  0.71051264,  0.13425249],
       [ 0.9298351 ,  0.35012857,  0.51996718,  0.44845842,  0.92092282],
       [ 0.59585316,  0.7778326 ,  0.09498829,  0.45896575,  0.16663657]])
```

In [436]:
```
a[[1, 5, 9],4:1:-1]
```

Out[436]:
```
array([[ 0.13425249,  0.71051264,  0.16428273],
       [ 0.92092282,  0.44845842,  0.51996718],
       [ 0.16663657,  0.45896575,  0.09498829]])
```

In [437]:
```
a[[1, 5, 9],[2,3,4]]
```

Out[437]:
```
array([ 0.16428273,  0.44845842,  0.16663657])
```

Αυτό είναι ισοδύναμο με:

In [439]:
```
b = [[1, 5, 9],[2,3,4]]
```

In [440]:
```
a[b]
```

Out[440]:
```
array([ 0.16428273,  0.44845842,  0.16663657])
```

Οι τιμές στα indexes μπορούν να επαναλαμβάνονται:

In [442]:
```
a[[0, 0, 1],:]
```

Out[442]:
```
array([[ 0.69209415,  0.74405598,  0.58994422,  0.51903866,  0.93660742],
       [ 0.69209415,  0.74405598,  0.58994422,  0.51903866,  0.93660742],
       [ 0.42445219,  0.54983743,  0.16428273,  0.71051264,  0.13425249]])
```

Επίσης τα indexes μπορούν να είναι και numpy arrays:

In [443]:
```
a = np.random.random([10, 3])
a
```

Out[443]:
```
array([[ 0.72032351,  0.1700159 ,  0.86081639],
       [ 0.61504558,  0.09700244,  0.58100856],
       [ 0.69110959,  0.06757325,  0.04092057],
       [ 0.51988531,  0.53892816,  0.2990308 ],
       [ 0.15582942,  0.46055668,  0.83132364],
       [ 0.79502634,  0.29743753,  0.76092162],
       [ 0.93368062,  0.7454287 ,  0.76971832],
       [ 0.26996306,  0.05723047,  0.26819277],
       [ 0.72463148,  0.70074029,  0.03486837],
       [ 0.46151354,  0.38307966,  0.37576748]])
```

In [444]:
```
a[np.array([0,0])] # Η πρώτη γραμμή δύο φορές
```

Out[444]:
```
array([[ 0.72032351,  0.1700159 ,  0.86081639],
       [ 0.72032351,  0.1700159 ,  0.86081639]])
```

In [445]:
```
a[np.array([0,0]),:] # Το ίδιο με παραπάνω
```

Out[445]:
```
array([[ 0.72032351,  0.1700159 ,  0.86081639],
       [ 0.72032351,  0.1700159 ,  0.86081639]])
```

In [446]:
```
a[:,np.array([0,0])] # Η πρώτη στήλη δύο φορές:
```

Out[446]:
```
array([[ 0.72032351,  0.72032351],
       [ 0.61504558,  0.61504558],
       [ 0.69110959,  0.69110959],
       [ 0.51988531,  0.51988531],
       [ 0.15582942,  0.15582942],
       [ 0.79502634,  0.79502634],
       [ 0.93368062,  0.93368062],
       [ 0.26996306,  0.26996306],
       [ 0.72463148,  0.72463148],
       [ 0.46151354,  0.46151354]])
```

Έναν πολυδιάστατο πίνακα μπορούμε να τον κάνουμε μονοδιάστατο (ή αλλιώς vector) με τρεις τρόπους.

In [448]:
```
a
```

Out[448]:
```
array([[ 0.72032351,  0.1700159 ,  0.86081639],
       [ 0.61504558,  0.09700244,  0.58100856],
       [ 0.69110959,  0.06757325,  0.04092057],
       [ 0.51988531,  0.53892816,  0.2990308 ],
       [ 0.15582942,  0.46055668,  0.83132364],
       [ 0.79502634,  0.29743753,  0.76092162],
       [ 0.93368062,  0.7454287 ,  0.76971832],
       [ 0.26996306,  0.05723047,  0.26819277],
       [ 0.72463148,  0.70074029,  0.03486837],
       [ 0.46151354,  0.38307966,  0.37576748]])
```

Ο πρώτος τρόπος είναι με κλασσικό list comprehension. **ΠΡΟΣΟΧΗ!** αυτός είναι και ο πιο "λάθος" τρόπος. Πρώτον γιατί δημιουργεί κουνελάκια με κατάθληψη και 2ον γιατί δουλεύει μόνο με διδιάστατους πίνακες (μπορεί όμως με κάποιες αλλαγές να δουλέψει και με ν-διάστατο):

In [450]:
```
[y for x in a for y in x]
```

Out[450]:
```
[0.72032351104388803,
 0.17001589545430396,
 0.86081639054798709,
 0.6150455802614303,
 0.09700244086120513,
 0.58100855972950105,
 0.69110959169793174,
 0.067573249596323492,
 0.040920571671677952,
 0.51988530822513479,
 0.53892816090621609,
 0.2990307992517659,
 0.15582942357273566,
 0.4605566802126575,
 0.8313236400044498,
 0.7950263426004629,
 0.29743752966900272,
 0.76092162436963118,
 0.93368061958375614,
 0.74542870354155555,
 0.76971832134271212,
 0.26996305881053206,
 0.057230468408892787,
 0.2681927734696512,
 0.72463147887657697,
 0.70074028551106737,
 0.034868370907724544,
 0.46151353728526512,
 0.38307966172904284,
 0.37576747710142233]
```

Ο 2ος τρόπος είναι με τη flat. Η οποία όμως δημιουργεί generator:

In [144]:
```
a
```

Out[144]:
```
array([[ 0.72736581,  0.54898777,  0.30900569],
       [ 0.94525329,  0.39233765,  0.81590939],
       [ 0.20146162,  0.99969513,  0.92789164],
       [ 0.53228237,  0.93805259,  0.80061147],
       [ 0.26791742,  0.5269165 ,  0.5012809 ],
       [ 0.25878137,  0.36084797,  0.95754485],
       [ 0.18318426,  0.92218919,  0.86068247],
       [ 0.84290356,  0.77998675,  0.6906613 ],
       [ 0.23294411,  0.96024721,  0.59429307],
       [ 0.04607213,  0.53834989,  0.71079965]])
```

In [453]:
```
a.flat
```

Out[453]:
```
<numpy.flatiter at 0x7f954cbcfc00>
```

Ο τρίτος τρόπος είναι με τη ravel:

In [148]:
```
a.shape
```

Out[148]:
```
(10, 3)
```

In [149]:
```
a.ravel().shape
```

Out[149]:
```
(30,)
```

Μπορούμε να ενώσουμε δύο (ή παραπάνω) πίνακες:

In [457]:
```
a = np.random.random([2,3])
b = np.random.random([2,3])
```

In [458]:
```
a
```

Out[458]:
```
array([[ 0.29052439,  0.24849151,  0.36284575],
       [ 0.92366061,  0.43703868,  0.06389083]])
```

In [459]:
```
b
```

Out[459]:
```
array([[ 0.8748667 ,  0.73579282,  0.20178447],
       [ 0.21344032,  0.98158518,  0.73810592]])
```

Η vstack ενώνει τους πίνακες vertically, δηλαδή ο ένας κάτω από τον άλλο:

In [460]:
```
np.vstack([a,b])
```

Out[460]:
```
array([[ 0.29052439,  0.24849151,  0.36284575],
       [ 0.92366061,  0.43703868,  0.06389083],
       [ 0.8748667 ,  0.73579282,  0.20178447],
       [ 0.21344032,  0.98158518,  0.73810592]])
```

In [461]:
```
np.vstack([a,b,2*a])
```

Out[461]:
```
array([[ 0.29052439,  0.24849151,  0.36284575],
       [ 0.92366061,  0.43703868,  0.06389083],
       [ 0.8748667 ,  0.73579282,  0.20178447],
       [ 0.21344032,  0.98158518,  0.73810592],
       [ 0.58104878,  0.49698301,  0.72569149],
       [ 1.84732122,  0.87407735,  0.12778165]])
```

Η hstack ενώνει τους πίνακες horizontally, δηλαδή ο ένας δίπλα από τον άλλο:

In [462]:
```
a
```

Out[462]:
```
array([[ 0.29052439,  0.24849151,  0.36284575],
       [ 0.92366061,  0.43703868,  0.06389083]])
```

In [463]:
```
b
```

Out[463]:
```
array([[ 0.8748667 ,  0.73579282,  0.20178447],
       [ 0.21344032,  0.98158518,  0.73810592]])
```

In [464]:
```
np.hstack([a,b])
```

Out[464]:
```
array([[ 0.29052439,  0.24849151,  0.36284575,  0.8748667 ,  0.73579282,
         0.20178447],
       [ 0.92366061,  0.43703868,  0.06389083,  0.21344032,  0.98158518,
         0.73810592]])
```

Προσοχή στη vstack πρέπει το πλήθος από στήλες να είναι ίδιος. Στη hstach πρέπει το πλήθος από γραμμές να είναι ο ίδιος:

In [465]:
```
np.hstack([a,b[:,:-1]])
```

Out[465]:
```
array([[ 0.29052439,  0.24849151,  0.36284575,  0.8748667 ,  0.73579282],
       [ 0.92366061,  0.43703868,  0.06389083,  0.21344032,  0.98158518]])
```

In [466]:
```
a
```

Out[466]:
```
array([[ 0.29052439,  0.24849151,  0.36284575],
       [ 0.92366061,  0.43703868,  0.06389083]])
```

In [467]:
```
b[:,:-1]
```

Out[467]:
```
array([[ 0.8748667 ,  0.73579282],
       [ 0.21344032,  0.98158518]])
```

In [468]:
```
np.vstack([a,b[:,:-1]])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-468-4935f9921709> in <module>()
----> 1 np.vstack([a,b[:,:-1]])

~/anaconda3/envs/arkalos/lib/python3.6/site-packages/numpy/core/shape_base.py
in vstack(tup)
    235
    236     """
--> 237     return _nx.concatenate([atleast_2d(_m) for _m in tup], 0)
    238
    239 def hstack(tup):

ValueError: all the input array dimensions except for the concatenation axis m
ust match exactly
```

Αντί για την hstack και τη vstack μπορείτε να χρησιμοποιήσετε τη block. Απλά φτιάχνετε λίστες με τους πίνακες που θέλετε να ενώσετε: [a,b] --> ίδιο με hstack. [[a], [b]] --> ίδιο με vstack.

In [469]:
```
a
```

Out[469]:
```
array([[ 0.29052439,  0.24849151,  0.36284575],
       [ 0.92366061,  0.43703868,  0.06389083]])
```

In [470]:
```
np.block([a,b])
```

Out[470]:
```
array([[ 0.29052439,  0.24849151,  0.36284575,  0.8748667 ,  0.73579282,
         0.20178447],
       [ 0.92366061,  0.43703868,  0.06389083,  0.21344032,  0.98158518,
         0.73810592]])
```

In [471]:
```
np.block([[a], [b]])
```

Out[471]:
```
array([[ 0.29052439,  0.24849151,  0.36284575],
       [ 0.92366061,  0.43703868,  0.06389083],
       [ 0.8748667 ,  0.73579282,  0.20178447],
       [ 0.21344032,  0.98158518,  0.73810592]])
```

Το αντίθετο με τη vstack και hstack κάνει η vsplit και hsplit:

In [504]:
```
a = np.random.random((10,4))
a
```

Out[504]:
```
array([[ 0.81246515,  0.64579695,  0.31261692,  0.72833299],
       [ 0.50548399,  0.45691983,  0.94293484,  0.10713851],
       [ 0.25177997,  0.2096684 ,  0.50523253,  0.37108323],
       [ 0.16177285,  0.31801499,  0.10796055,  0.45283983],
       [ 0.92833983,  0.40167612,  0.42314142,  0.55412818],
       [ 0.72985404,  0.64141386,  0.68094954,  0.41604735],
       [ 0.8929054 ,  0.88354153,  0.86002467,  0.54289843],
       [ 0.60979488,  0.36884681,  0.61865976,  0.74078811],
       [ 0.39804021,  0.08909003,  0.05669355,  0.16086856],
       [ 0.96258289,  0.47762343,  0.69156939,  0.96706104]])
```

In [505]:
```
a.shape
```

Out[505]:
```
(10, 4)
```

In [506]:
```
np.hsplit(a, 2) # Φτιάχνει μία λίστα με δύο πίνακες. Ο κάθε ένας είναι 10 X 2.
```

Out[506]:
```
[array([[ 0.81246515,  0.64579695],
        [ 0.50548399,  0.45691983],
        [ 0.25177997,  0.2096684 ],
        [ 0.16177285,  0.31801499],
        [ 0.92833983,  0.40167612],
        [ 0.72985404,  0.64141386],
        [ 0.8929054 ,  0.88354153],
        [ 0.60979488,  0.36884681],
        [ 0.39804021,  0.08909003],
        [ 0.96258289,  0.47762343]]), array([[ 0.31261692,  0.72833299],
        [ 0.94293484,  0.10713851],
        [ 0.50523253,  0.37108323],
        [ 0.10796055,  0.45283983],
        [ 0.42314142,  0.55412818],
        [ 0.68094954,  0.41604735],
        [ 0.86002467,  0.54289843],
        [ 0.61865976,  0.74078811],
        [ 0.05669355,  0.16086856],
        [ 0.69156939,  0.96706104]])]
```

Σε ένα array μπορούμε να κάνουμε λογικές πράξεις:

In [507]:
```
a
```

Out[507]:
```
array([[ 0.81246515,  0.64579695,  0.31261692,  0.72833299],
       [ 0.50548399,  0.45691983,  0.94293484,  0.10713851],
       [ 0.25177997,  0.2096684 ,  0.50523253,  0.37108323],
       [ 0.16177285,  0.31801499,  0.10796055,  0.45283983],
       [ 0.92833983,  0.40167612,  0.42314142,  0.55412818],
       [ 0.72985404,  0.64141386,  0.68094954,  0.41604735],
       [ 0.8929054 ,  0.88354153,  0.86002467,  0.54289843],
       [ 0.60979488,  0.36884681,  0.61865976,  0.74078811],
       [ 0.39804021,  0.08909003,  0.05669355,  0.16086856],
       [ 0.96258289,  0.47762343,  0.69156939,  0.96706104]])
```

In [508]:
```
a>0.5
```

Out[508]:
```
array([[ True,  True, False,  True],
       [ True, False,  True, False],
       [False, False,  True, False],
       [False, False, False, False],
       [ True, False, False,  True],
       [ True,  True,  True, False],
       [ True,  True,  True,  True],
       [ True, False,  True,  True],
       [False, False, False, False],
       [ True, False,  True,  True]], dtype=bool)
```

Το ακόμα πιο ενδιαφέρον είναι ότι μορούμε να βάλουμε έναν πίνακα από boolean τιμές (True, False) στο index ενός άλλου πίνακα! Το αποτέλεσεμα είναι ένας νέος πίνακας ο οποίος περιέχει μόνο τα στοιχεία που το index τους ήταν True:

In [490]:
```
b = np.array([5,3,1])
b
```

Out[490]:
```
array([5, 3, 1])
```

In [491]:
```
b[[True, False,True]]
```

Out[491]:
```
array([5, 1])
```

In [492]:
```
b>2
```

Out[492]:
```
array([ True,  True, False], dtype=bool)
```

Συνεπώς, μπορώ να χρησιμοποιήσω boolean πράξεις τους πίνακα ως index στον ίδιο τον πίνακα!

In [493]:
```
b[b>2]
```

Out[493]:
```
array([5, 3])
```

Π.χ. πάρε όλα τα στοιχεία του a τα οποία είναι > 0.5:

In [509]:
```
a[a>0.5]
```

Out[509]:
```
array([ 0.81246515,  0.64579695,  0.72833299,  0.50548399,  0.94293484,
        0.50523253,  0.92833983,  0.55412818,  0.72985404,  0.64141386,
        0.68094954,  0.8929054 ,  0.88354153,  0.86002467,  0.54289843,
        0.60979488,  0.61865976,  0.74078811,  0.96258289,  0.69156939,
        0.96706104])
```

In [510]:
```
a
```

Out[510]:
```
array([[ 0.81246515,  0.64579695,  0.31261692,  0.72833299],
       [ 0.50548399,  0.45691983,  0.94293484,  0.10713851],
       [ 0.25177997,  0.2096684 ,  0.50523253,  0.37108323],
       [ 0.16177285,  0.31801499,  0.10796055,  0.45283983],
       [ 0.92833983,  0.40167612,  0.42314142,  0.55412818],
       [ 0.72985404,  0.64141386,  0.68094954,  0.41604735],
       [ 0.8929054 ,  0.88354153,  0.86002467,  0.54289843],
       [ 0.60979488,  0.36884681,  0.61865976,  0.74078811],
       [ 0.39804021,  0.08909003,  0.05669355,  0.16086856],
       [ 0.96258289,  0.47762343,  0.69156939,  0.96706104]])
```

Είναι σημαντικό να τονίσουμε ότι αυτή την "άλγεβρα" την υποστηρίζουν και η R, Matlab, Octave. Επίσης μπορούμε να κάνουμε ανάθεση σε αυτά τα στοιχεία. Π.χ. πάρε όλα τα στοιχεία του a που είναι >0.5 και κάνε τα 10:

In [511]:
```
a[a>0.5] = 10
a
```

Out[511]:
```
array([[ 10.        ,  10.        ,   0.31261692,  10.        ],
       [ 10.        ,   0.45691983,  10.        ,   0.10713851],
       [  0.25177997,   0.2096684 ,  10.        ,   0.37108323],
       [  0.16177285,   0.31801499,   0.10796055,   0.45283983],
       [ 10.        ,   0.40167612,   0.42314142,  10.        ],
       [ 10.        ,  10.        ,  10.        ,   0.41604735],
       [ 10.        ,  10.        ,  10.        ,  10.        ],
       [ 10.        ,   0.36884681,  10.        ,  10.        ],
       [  0.39804021,   0.08909003,   0.05669355,   0.16086856],
       [ 10.        ,   0.47762343,  10.        ,  10.        ]])
```

Μπορούμε να κάνουμε ανάθεση ολόκληρο πίνακα. Πρέπει όμως το πλήθος των στοιχείων του να είναι ίδιο με το πλήθος των στοιχείων που αντικαθιστά:

In [530]:
```
b = np.arange(1,11)
b
```

Out[530]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [531]:
```
b[b<4] = np.array([20,21,22])
b
```

Out[531]:
```
array([20, 21, 22,  4,  5,  6,  7,  8,  9, 10])
```

Ο τελεστής '''~''' σε ένα index σημαίνει το αντίθετο. Π.χ:

In [514]:
```
a = np.random.random((3,4))
a
```

Out[514]:
```
array([[ 0.15776221,  0.43247282,  0.49855915,  0.57465768],
       [ 0.14101421,  0.19224792,  0.10055011,  0.85338245],
       [ 0.36397825,  0.10033724,  0.38265009,  0.85358012]])
```

Όλα τα στοιχεία που είναι μεγαλύτερο από 0.8:

In [516]:
```
a[a>0.8]
```

Out[516]:
```
array([ 0.85338245,  0.85358012])
```

Όλα τα στοιχεία που ΔΕΝ είναι μεγαλύτερα από 0.8:

In [518]:
```
a[~(a>8)]
```

Out[518]:
```
array([ 0.15776221,  0.43247282,  0.49855915,  0.57465768,  0.14101421,
        0.19224792,  0.10055011,  0.85338245,  0.36397825,  0.10033724,
        0.38265009,  0.85358012])
```

Επίσης μπορούμε να χρησιμοποιήσουμε and, or, ...

In [520]:
```
a[np.bitwise_or(a<0.3, a>0.8)] # Όλα τα στοιχεία που είναι μικρότερα από 0.3 κα
ι μεγαλύτερα από 0.8
```

Out[520]:
```
array([ 0.15776221,  0.14101421,  0.19224792,  0.10055011,  0.85338245,
        0.10033724,  0.85358012])
```

Η numpy επίσης υποστηρίζει κάποιες ειδικές τιμές:

In [521]:
```
np.inf # Το άπειρο!
```

Out[521]:
```
inf
```

In [522]:
```
np.inf > 100000000
```

Out[522]:
```
True
```

Π.χ:

In [525]:
```
np.array([1])/np.array([0])
```

```
/Users/alexandroskanterakis/anaconda3/envs/arkalos/lib/python3.6/site-packages
/ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountered in true_d
ivide
  """Entry point for launching an IPython kernel.
```

Out[525]:
```
array([ inf])
```

In [524]:
```
np.array([-1])/np.array([0])
```

```
/Users/alexandroskanterakis/anaconda3/envs/arkalos/lib/python3.6/site-packages
/ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountered in true_d
ivide
  """Entry point for launching an IPython kernel.
```

Out[524]:
```
array([-inf])
```

Επίσης υπάρχει και η ειδική τιμή nan (Not a Number)

In [529]:
```
np.nan
```

Out[529]:
```
nan
```

Οι συναρτήσεις np.isnan και np.isinf επιστρέφουν True/False ανάλογα και μπορούν να χρησιμοποιηθούν για να "βγάλουμε" αυτές τις τιμές από έναν πίνακα:

In [572]:
```
a=np.array([1,2,3,np.nan,4, np.nan,5])
a
```

Out[572]:
```
array([  1.,   2.,   3.,  nan,   4.,  nan,   5.])
```

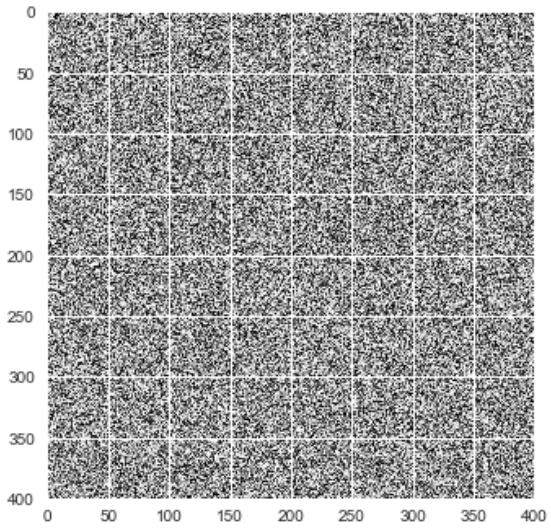
In [573]:
```
a[~np.isnan(a)]
```

Out[573]:
```
array([ 1.,  2.,  3.,  4.,  5.])
```

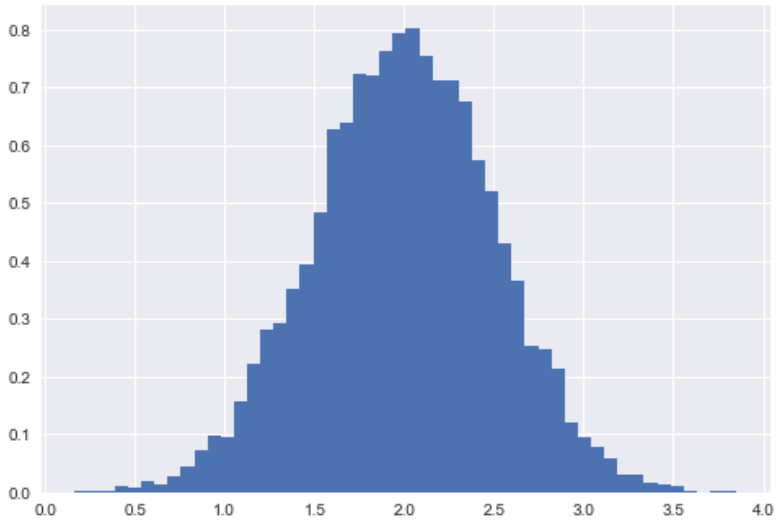
η numpy υποστηρίζεται από τη matplotlib και όλες τις επιστημονικές βιβλιοθήκες της python:

In [555]:
```
import matplotlib.pyplot as plt
```

In [569]:
```
# Φτιάξε ένα image από έναν πίνακα:
plt.imshow(np.random.random((400,400)))
plt.show()
```



In [570]:
```
# Ιστόγραμμα:
mu, sigma = 2, 0.5
v = np.random.normal(mu,sigma,10000)
plt.hist(v, bins=50, normed=1)
plt.show()
```



Γραμμική άλγεβρα

Ο αντίστροφος ενός πίνακα:

In [563]:
```
a = np.random.random((3,3))
a
```

Out[563]:
```
array([[ 0.38449346,  0.29866245,  0.50693455],
       [ 0.27030156,  0.94599582,  0.03163946],
       [ 0.04847714,  0.49405979,  0.43031744]])
```

In [564]:
```
np.linalg.inv(a)
```

Out[564]:
```
array([[ 2.43620688,  0.75888255, -2.92576581],
       [-0.71435503,  0.87677738,  0.77707859],
       [ 0.54572215, -1.09214466,  1.7612799 ]])
```

In [565]:
```
np.dot(a,np.linalg.inv(a))
```

Out[565]:
```
array([[  1.00000000e+00,   5.52820098e-17,  -9.52376299e-17],
       [ -2.08616481e-16,   1.00000000e+00,   9.61564695e-17],
       [ -4.59407761e-17,  -6.90990097e-17,   1.00000000e+00]])
```

In [566]:
```
a = np.array([[3,5], [0,4]])
np.dot(a,np.linalg.inv(a))
```

Out[566]:
```
array([[ 1.,  0.],
       [ 0.,  1.]])
```

Ο μοναδιαίος πίνακας (I)

In [567]:
```
np.eye(4)
```

Out[567]:
```
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

In [574]:
```
arr = np.array([[1, 2], [3, 4]])
```

Η ορίζουσα ενός πίνακα:

In [576]:
```
np.linalg.det(arr)
```

Out[576]:
```
-2.0000000000000004
```

Για ιδιόμορφο πίνακα (η ορίζουσα είναι μηδέν) θα οδηγήσει σε Error τύπου LinAlgError:

In [578]:
```
arr = np.array([[3, 2], [6, 4]])
np.linalg.inv(arr)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-578-8e7f61226c1e> in <module>()
      1 arr = np.array([[3, 2], [6, 4]])
----> 2 np.linalg.inv(arr)

~/anaconda3/envs/arkalos/lib/python3.6/site-packages/numpy/linalg/linalg.py in
inv(a)
    511     signature = 'D->D' if isComplexType(t) else 'd->d'
    512     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 513     ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    514     return wrap(ainv.astype(result_t, copy=False))
    515

~/anaconda3/envs/arkalos/lib/python3.6/site-packages/numpy/linalg/linalg.py in
_raise_linalgerror_singular(err, flag)
     88
     89 def _raise_linalgerror_singular(err, flag):
---> 90     raise LinAlgError("Singular matrix")
     91
     92 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

Αποθήκευση και φόρτωμα δεδομένων

Η numpy έχει το δικό της format για αποθήκευση δεδομένων:

In [587]:
```
A = np.random.random((2,3))
A
```

Out[587]:
```
array([[ 0.00116433,  0.0435687 ,  0.87706621],
       [ 0.90810222,  0.15021207,  0.42381173]])
```

In [588]:
```
np.save('my_data.npy', A)
```

In [589]:
```
! ls -l my_data.npy
```

```
-rw-r--r--  1 alexandroskanterakis  staff  128 Jan 13 12:52 my_data.npy
```

In [590]:
```
B = np.load('my_data.npy')
B
```

Out[590]:
```
array([[ 0.00116433,  0.0435687 ,  0.87706621],
       [ 0.90810222,  0.15021207,  0.42381173]])
```

Scipy : high-level scientific computing

Σημαντικά πακέτα (routines) της scipy

- File input/output: **scipy.io**
- Special functions: **scipy.special**
- Linear algebra operations: **scipy.linalg**
- Fast Fourier transforms: **scipy.fftpack**
- Optimization and fit: **scipy.optimize**
- Statistics and random numbers: **scipy.stats**
- Interpolation: **scipy.interpolate**
- Numerical integration: **scipy.integrate**
- Signal processing: **scipy.signal**
- Image processing: **scipy.ndimage**

Routines	Περιγραφή
scipy.cluster	Vector quantization / Kmeans
scipy.constants	Physical and mathematical constants
scipy.fftpack	Fourier transform
scipy.integrate	Integration routines
scipy.interpolate	Interpolation
scipy.io	Data input and output
scipy.linalg	Linear algebra routines
scipy.ndimage	n-dimensional image package
scipy.odr	Orthogonal distance regression
scipy.optimize	Optimization
scipy.signal	Signal processing
scipy.sparse	Sparse matrices
scipy.spatial	Spatial data structures and algorithms
scipy.special	Any special mathematical functions
scipy.stats	Statistics

Παράδειγμα: Γραμμική άλγεβρα με scipy:

In [593]:
```
from scipy import linalg
```

In [594]:
```
arr = np.array([[1, 2], [3, 4]])
```

LU παραγοντοποίση

In [597]:
```
P, L, U = linalg.lu(arr)
```

In [598]:
```
# Επαλήθευση
from scipy import allclose, diag, dot
allclose(arr, P.dot(L.dot(U)))
```

Out[598]:
```
True
```

QR παραγοντοποίση

In [599]:
```
Q, R = linalg.qr(arr)
```

In [600]:
```
# Επαλήθευση
allclose(arr, Q.dot(R))
```

Out[600]:
```
True
```

SVD παραγοντοποίση

In [601]:
```
S, V, D = linalg.svd(arr)
```

In [602]:
```
# Επαλήθευση
allclose(arr, S.dot(diag(V)).dot(D))
```

Out[602]:
```
True
```

Υπολογισμός ιδιοτιμών και ιδιοδυανισμάτων (eigenvalues - eigenvectors)

In [604]:
```
eigvals, eigvecs = linalg.eig(arr)
```

In [606]:
```
eigvals
```

Out[606]:
```
array([-0.37228132+0.j,  5.37228132+0.j])
```

In [607]:
```
eigvecs
```

Out[607]:
```
array([[-0.82456484, -0.41597356],
       [ 0.56576746, -0.90937671]])
```