ΜΕΜ104 Γλώσσα Προγραμματισμού Ι

Μιχάλης Πλεξουσάκης

15 Δεκεμβρίου 2020

Μαθηματικά και Εφαρμοσμένα Μαθηματικά

Περιεχόμενα

Τυχαίοι αριθμοί

Τυχαίοι αριθμοί

Η Python παρέχει αρκετές χρήσιμες συναρτήσεις παραγωγής (*ψεύδο*)-*τυχαίων* αριθμών στη βιβλιοθήκη `random`.

Όλες σχεδόν οι συναρτήσεις της `random` δομούνται με χρήση της συνάρτησης `random.random` η οποία παράγει ένα τυχαίο αριθμό κινητής υποδιαστολής στο διάστημα $[0, 1)$.

```
>>> import random
>>> random.random()
0.893468922695049

>>> random.random()
0.26665605754617294
```

Τυχαίοι αριθμοί

Οι "τυχαίοι" αριθμοί που παράγει η `random.random` δεν είναι στην πραγματικότητα τυχαίοι αλλά είναι όροι μιας ακολουθίας.

Αρχικοποιούμε την ακολουθία τυχαίων αριθμών καλώντας τη συνάρτηση `seed()` με όρισμα κάποιο ακέραιο αριθμό (αν δεν δοθεί νοείται ένας ακέραιος που προκύπτει από την ώρα της ημέρας):

```
>>> random.seed(1331)
>>> random.random()
0.5770817438415198

>>> random.seed()
>>> random.random()
0.4885163689484523
```

Τυχαίοι αριθμοί

Εφαρμογή. Μπορούμε να προσομοιώσουμε τη ρίψη ενός νομίσματος χρησιμοποιώντας τη `random.random`. Οι αριθμοί που είναι μεγαλύτεροι από 0.5 αντιμετωπίζονται ως κορώνα και οι μικρότεροι ως γράμματα.

Η συνάρτηση που ακολουθεί επιστρέφει το ποσοστό εμφανίσεων κορώνας σε `N` ρίψεις ενός νομίσματος:

```
def flip(N):
    heads = 0
    for i in range(N):
        if random.random() > 0.5:
            heads += 1
    return heads / N
```

Τυχαίοι αριθμοί

Κάνουμε τώρα το ακόλουθο πείραμα: εκτελούμε `numTrials` δοκιμές και σε κάθε μια από αυτές ρίχνουμε ένα νόμισμα `N` φορές και υπολογίζουμε τον μέσο όρο του ποσοστού εμφανίσεων κορώνας:

```
def flipSim(N, numTrials):
    heads = []
    for i in range(numTrials):
        heads.append( flip(N) )
    return sum(heads) / numTrials
```

```
>>> flipSim(100, 10000)
0.4993480000000223
>>> flipSim(100, 100000)
0.5000012999999961
```

Το πρόβλημα του Pascal. Είναι επικερδές να στοιχηματίσουμε ότι σε 24 ρίψεις δύο ζαριών θα φέρουμε εξάρες;

Η απάντηση είναι, φυσικά, όχι:

- Η πιθανότητα να μην φέρουμε εξάρες σε μια ρίψη είναι $35/36$
- Η πιθανότητα να μην φέρουμε εξάρες σε 24 ρίψεις είναι $(35/36)^{24}$
- Άρα, η πιθανότητα να φέρουμε εξάρες σε 24 ρίψεις είναι $1 - (35/36)^{24} \approx 0.4914$

Μακροπρόθεσμα, *δεν είναι επικερδές* να στοιχηματίσουμε ότι θα φέρουμε εξάρες σε 24 ρίψεις

Τυχαίοι αριθμοί

Μπορούμε να προσομοιώσουμε τη ρίψη ενός ζαριού χρησιμοποιώντας τη συνάρτηση `random.choice`. Αν `seq` είναι μια ακολουθία, τότε η `random.choice(seq)` επιστρέφει ένα τυχαίο όρο της ακολουθίας.

Η ρίψη ενός ζαριού μπορεί να προσομοιωθεί με τη συνάρτηση

```
def rollDie():
    return random.choice( [1, 2, 3, 4, 5, 6] )

>> rollDie()
2
>>> rollDie()
6
```

Τυχαίοι αριθμοί

Για το πρόβλημα του Pascal μπορούμε να χρησιμοποιήσουμε την προσομοίωση

```
def pascal(numTrials):
    wins = 0
    for i in range(numTrials):
        for j in range(24):
            d1 = rollDie();  d2 = rollDie()
            if d1 == 6 and d2 == 6:
                wins += 1
                break
    print('Prob. of winning =', wins / numTrials)

>> pascal(10000)
Prob. of winning = 0.4951
```

Εύρεση του π. Οι Γάλλοι μαθηματικοί Buffon (1707-1788) και Laplace (1749-1827) πρότειναν τη χρήση μιας στοχαστικής προσομοίωσης για την εκτίμηση της τιμής του π.

Σε ένα τετράγωνο με πλευρά 2, ο εγγεγραμμένος κύκλος έχει εμβαδό ίσο με π. Ο Buffon ισχυρίστηκε ότι πετώντας ένα μεγάλο πλήθος από βελόνες πάνω από το τετράγωνο, ο λόγος του πλήθους των βελονών με τις μύτες μέσα στο τετράγωνο προς το πλήθος των βελονών με τις μύτες μέσα στον κύκλο προσεγγίζει το $\pi/4$. Ο λόγος γι' αυτό είναι προφανής:

$$\frac{\text{βελόνες στον κύκλο}}{\text{βελόνες στο τετράγωνο}} = \frac{\text{εμβαδό κύκλου}}{\text{εμβαδό τετραγώνου}} = \frac{\text{εμβαδό κύκλου}}{4}$$

Τυχαίοι αριθμοί

Προσομοίωση της ρίψης μιας βελόνας. Χρησιμοποιούμε τη `random.random` για να παράγουμε τις συντεταγμένες (x, y) της μύτης της βελόνας. Η μύτη της βελόνας είναι μέσα στον κύκλο αν η απόστασή της από την αρχή των αξόνων είναι μικρότερη από 1.

```
def throwNeedle(numNeedles):
    inside = 0
    for i in range(numNeedles):
        x = random.random()
        y = random.random()
        if x**2 + y**2 <= 1.0:
            inside += 1
    return 4 * inside / numNeedles
```

Χρησιμοποιώντας αυτόν τον κώδικα έχουμε:

```
>>> throwNeedle(1000)
3.168
>>> throwNeedle(2000)
3.136
>>> throwNeedle(5000)
3.1552
>>> throwNeedle(7500)
3.1659
>>> throwNeedle(10000)
3.1376
```

Ερώτηση. Βελτιώνονται οι εκτιμήσεις του π με την αύξηση του αριθμού των επαναλήψεων του πειράματος;

Τυχαίοι αριθμοί

Μπορούμε να υπολογίσουμε την *τυπική απόκλιση* (standard deviation) σ της συλλογής $\mathcal{X}$ των αποτελεσμάτων, δηλαδή

$$\sigma = \left(\frac{1}{\texttt{numTrials}} \sum_{x \in \mathcal{X}} (x - \mu)^2 \right)^{1/2},$$

ένα μέτρο του ποσοστού των τιμών που βρίσκονται κοντά στον μέσο μ.

- Αν υπάρχουν πολλές τιμές γύρω από τον μέσο, η τυπική απόκλιση θα είναι μικρή
- Αν υπάρχουν πολλές τιμές σχετικά μακριά από τον μέσο, η τυπική απόκλιση θα είναι μεγάλη
- Αν όλες οι τιμές είναι ίσες, τότε η τυπική απόκλιση είναι μηδέν

Τυχαίοι αριθμοί

```
import random, statistics

def estimate(numNeedles, numTrials):
    estim = []
    for i in range(numTrials):
        guess = throwNeedle(numNeedles)
        estim.append(guess)
    return statistcs.mean(estim), \
           statistics.stdev(estim)

numNeedles = 1000; numTrials = 100
while numNeedles < 200000:
    mu, sd = estimate(numNeedles, numTrials)
    numNeedles *= 2
```

Τυχαίοι αριθμοί

Ο κώδικας της προηγούμενης σελίδας παράγει

```
Needles =   1000  pi estim. = 3.145680  stdev = 0.047326
Needles =   2000  pi estim. = 3.141400  stdev = 0.039496
Needles =   4000  pi estim. = 3.139800  stdev = 0.027482
Needles =   8000  pi estim. = 3.143445  stdev = 0.017039
Needles =  16000  pi estim. = 3.142138  stdev = 0.012154
Needles =  32000  pi estim. = 3.142696  stdev = 0.008820
Needles =  64000  pi estim. = 3.141208  stdev = 0.006873
Needles = 128000  pi estim. = 3.141552  stdev = 0.004637
```

Τυχαίοι αριθμοί

Άλλες χρήσιμες συναρτήσεις από τη βιβλιοθήκη `random` είναι:

`random.randrange(stop)` Επιστρέφει έναν τυχαίο ακέραιο στο διάστημα [0, `stop`)

`random.randrange(start, stop)` Επιστρέφει έναν τυχαίο ακέραιο στο διάστημα [`start`, `stop`)

`random.randrange(start, stop, step)` Επιστρέφει έναν τυχαίο ακέραιο στο `range(start, stop, step)`

```
>>> random.randrange(10)  # τυχαίος στο διάστημα [0,10)
7
>>> random.randrange(100, 200) # διάστημα [100, 200)
142
```

Τυχαίοι αριθμοί

`random.randint(a,b)` Επιστρέφει έναν τυχαίο ακέραιο μεταξύ των `a` και `b`, συμπεριλαμβανομένων και αυτών

```
>>> random.randint(0,10)
4
```

Ισοδύναμη με την `random.randrange(a, b+1)`

`random.sample(population, k)` Επιλέγει `k` στοιχεία της ακολουθίας `population` χωρίς επανάληψη.

```
>>> deck = ['four', 'ten', 'jack', 'nine', 'ace']
>>> random.sample(deck, 2)
['nine', 'ten']

>>> random.sample(range(100), 5)
[6, 47, 29, 78, 24]
```

`random.shuffle(seq)` Αναδιατάσσει την ακολουθία `seq`

```
>>> deck = ['four', 'ten', 'jack', 'nine', 'ace']
>>> random.shuffle(deck)
>>> deck
['jack', 'four', 'ace', 'nine', 'ten']

>>> nums = list(range(10))
>>> random.shuffle(nums)
>>> nums
[3, 5, 6, 4, 0, 8, 2, 9, 1, 7]
```

Παράδειγμα. Μπορούμε να φτιάξουμε μια τράπουλα με τις εντολές

```
ranks = ['A', '2', '3', '4', '5', ..., '9', '10', 'J', 'Q', 'K']
suits = ['Σπαθί', 'Καρό', 'Καρδιά', 'Μπαστούνι']

deck = [ rank+' '+suit for rank in ranks for suit in suits ]
```

Μπορούμε να ανακατέψουμε την τράπουλα και να επιλέξουμε 5 χαρτιά με τις εντολές

```
>>> random.shuffle(deck)
>>> random.sample(deck,5)
['10 Καρό', 'K Σπαθί', '10 Μπαστούνι', '3 Καρό', '7 Σπαθί']
```

Η βιβλιοθήκη numpy

Η βιβλιοθήκη numpy

Η `numpy` είναι μια βιβλιοθήκη της Python που παρέχει μεθόδους για τη διαχείριση πινάκων και μαθηματικές συναρτήσεις που δρουν πάνω σε αυτούς.

Τα βασικά αντικείμενα που ορίζονται στην `numpy` είναι οι *πολυδιάστατοι πίνακες*, αντικείμενα, δηλαδή, τύπου `ndarray`:

```
>>> import numpy as np

>>> a = np.array( [1, 2, 3] )
>>> type(a)
<class 'numpy.ndarray'>

>>> A = np.array([ [1, 2, 3], [4, 5, 6] ])
>>> type(A)
<class 'numpy.ndarray'>
```

Η βιβλιοθήκη numpy

Χαρακτηριστικά ενός αντικειμένου τύπου `ndarray` είναι:

`ndarray.ndim` Ο αριθμός των διαστάσεων (αξόνων) ενός αντικειμένου

```
>>> a = np.array( [1, 2, 3] )
>>> a.ndim
1
>>> A = np.array([ [1, 2, 3], [4, 5, 6] ])
>>> A.ndim
2
```

`ndarray.shape` Πλειάδα (tuple) με το πλήθος των στοιχείων κατά μήκος κάθε διάστασης

```
>>> a.shape
(3,)
>>> A.shape
(2,2)
```

`ndarray.size` Το πλήθος των στοιχείων του αντικειμένου

```
>>> a = np.array( [1, 2, 3] )
>>> a.size
3
>>> A = np.array([ [1, 2, 3], [4, 5, 6] ])
>>> A.size
6
```

Άλλα χαρακτηριστικά ενός αντικειμένου τύπου `ndarray` είναι το `ndarray.dtype`, ο τύπος των στοιχείων του αντικειμένου, και το `ndarray.itemsize`, το μέγεθος σε bytes κάθε στοιχείου.

Μερικοί χρήσιμοι πίνακες

```
>>> print( np.zeros( (4,3) ) )
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

 >>> print( np.ones( (3, 2) ) )
 [[1. 1.]
  [1. 1.]
  [1. 1.]]

 >>> print( np.eye(3) )  # ή np.identity(3)
 [[1. 0. 0.]
  [0. 1. 0.]
  [0. 0. 1.]]
```

Μπορούμε ακόμα να μετατρέψουμε ακολουθίες αριθμών σε πίνακες με τη συνάρτηση `arange` και τη μέθοδο `reshape`:

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])

>>> np.arange( 0, 2, 0.3 )
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])

>>> np.arange(10, 55, 5).reshape(3, 3)
array([[10, 15, 20,]
       [25, 30, 35],
       [40, 45, 50]])

>>> np.arange( 0.2, 1, 0.2 ).reshape(2, 2)
array([[0.2, 0.4],
       [0.6, 0.8]])
```

Η βιβλιοθήκη numpy

Μπορούμε να κατασκευάσουμε πίνακες χρησιμοποιώντας συναρτήσεις. Για παράδειγμα, οι εντολές

```
def f(i, j):
    return 10*i+j

A = np.fromfunction(f, (5,4), dtype=int)
print(A)
```

δίνουν

```
[[ 0,  1,  2,  3],
 [10, 11, 12, 13],
 [20, 21, 22, 23],
 [30, 31, 32, 33],
 [40, 41, 42, 43]]
```

Η προσπέλαση των στοιχείων γίνεται ενός αντικειμένου τύπου `ndarray` γίνεται χρησιμοποιώντας αριθμοδείκτες, όπως με τις λίστες ή άλλες ακολουθίες:

```
>>> A = np.arange(1, 31).reshape(5, 6)
>>> A[0,2]  # γραμμή 0, στήλη 2
3

>>> A[2]  # 2η γραμμή του A
array([13, 14, 15, 16, 17, 18])

>>> A[:, 3]  # η στήλη 3 του A
array([ 4, 10, 16, 22, 28])
```

```
>>> A[-1]  # τελευταία γραμμή του A
array([25, 26, 27, 28, 29, 30])

>>> A[:,-1]  # τελευταία στήλη του A
array([ 6, 12, 18, 24, 30])

>>> A[2:, ::2] # κάθε δεύτερη στήλη, γραμμές 2-5
array([[13, 15, 17],
       [19, 21, 23],
       [25, 27, 29]])

>>> A[::-1, 2]  # 2η στήλη, γραμμές 4 εώς 0
array([27, 21, 15,  9,  3])
```

Πράξεις σε στοιχεία πινάκων:

```
>>> x = np.linspace( 0, 2, 9 )
>>> y = np.sin(x)
```

Η συνάρτηση `linspace`

```
numpy.linspace(start, stop, num=50, endpoint=True, retstep=False,
               dtype=None, axis=0)
```

παράγει `num` σημεία, ομοιόμορφα κατανεμημένα στο διάστημα `[start, stop]`, συμπεριλαμβανομένων και των άκρων, εκτός εάν `endpoint = False`.

Η βιβλιοθήκη numpy

```
>>> v = np.arange(1,6)
>>> v**3
array([  1,   8,  27,  64, 125])

>>> np.exp(v)
array([ 2.71828183,   7.3890561 ,  20.08553692,
       54.59815003, 148.4131591 ])

>>> np.sqrt(v)
array([1.        , 1.41421356, 1.73205081,
       2.        , 2.23606798])

>>> np.log(v)
array([0., 0.69314718, 1.09861229, 1.38629436,
       1.60943791])
```

Η βιβλιοθήκη numpy

```
>>> a = np.random.random( (2, 3) )
array([[0.08758247, 0.86030845, 0.07928723],
       [0.56943072, 0.75171286, 0.56351744]])

>>> a.sum()
2.911839170103163

>>> a.min()
0.07928722660518017

>>> a.max()
0.8603084542863427

>>> a < 0.5
array([[ True, False,  True],
       [False, False, False]])
```

Η βιβλιοθήκη numpy

Αριθμητικοί τελεστές δρούν σε πίνακες, *κατά στοιχείο*, εν γένει

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> a - b
array([20, 29, 38, 47])

>>> a*b    # γινόμενο κατά συνιστώσα
array([  0,  30,  80, 150])

>>> A = np.array( [[1,1], [0,1]] )
>>> B = np.array( [[2,0], [3,4]] )

>>> A * B       # γινόμενο κατά συνιστώσα
array([[2, 0],
       [0, 4]])
```

με εξαίρεση τις πράξεις

```
>>> A @ B             # γινόμενο πινάκων
array([[5, 4],
       [3, 4]])

>>> np.dot(A, B) # ή A.dot(B) γινόμενο πινάκων A και B
array([[5, 4],
       [3, 4]])

>>> x = [1, 2, 3]
>>> y = [4, 5, 6]

>>> np.cross(x, y)    # εξωτερικό γινόμενο
array([-3,  6, -3])
```

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> np.dot(x, y)    # εσωτερικό γινόμενο
32

# Ιδιοτιμές και ιδιοδιανύσματα

>>> from numpy import linalg as LA

>>> w, v = LA.eig(np.array([[1, -1], [-1, 4]]))
>>> w
array([0.69722436, 4.30277564])
>>> v  # Οι στήλες του v είναι τα ιδιοδιανύσματα
array([[-0.95709203,  0.28978415],
       [-0.28978415, -0.95709203]])
```

Η βιβλιοθήκη numpy

```
>>> A = np.array([[1, 2], [3, 4]])
>>> LA.det(A)
-2.0

>>> LA.inv(A)   # Αντίστροφος πίνακας
array([[-2. ,  1. ],
       [ 1.5, -0.5]])

>>> x = np.array([ -1, 1])
>>> np.dot(A, x)  # ή A.dot(x)
array([1, 1])

>>> A.T.        # Ανάστροφος πίνακας
array([[1, 3],
       [2, 4]])
```

Μέτρο διανυσμάτων: Αν $x \in \mathbb{R}^n$ τότε η συνάρτηση `norm(x)` υπολογίζει την ποσότητα

$$(x_1^2 + x_2^2 + \cdots x_n^2)^{1/2}$$

```
>>> x = np.array( [1, 3, 4] )
>>> LA.norm(x)
5.0990195135927845

>>> y = np.array( [-1, 2, 1] )
>>> np.dot(x, y)
9

>>> np.dot(x.T, x)
26
```

Λύση γραμμικών συστημάτων

```
>>> A = np.array([[1, 2], [3, 4]])
>>> b = np.array([5, 6])

>>> x = LA.solve(A, b)     # Λύση του A x = b
array([-4. ,  4.5])

>>> A.dot(x)     # A x πρέπει να είναι ίσο με b
array([5., 6.])
```

Μερικές ασκήσεις με την numpy

Μερικές ασκήσεις με την numpy

Ερώτηση. Πως κατασκευάζουμε έναν μονοδιάστατο πίνακα;

```
>>> v = np.array( [1, 2, 3] )
>>> v
array([1, 2, 3])
```

Ερώτηση. Πως κατασκευάζουμε έναν πίνακα με δύο γραμμές και τρεις στήλες;

```
>>> A = np.array([ [1, 2, 3], [4, 5, 6] ])
>>> A
array([[1, 2, 3],
       [4, 5, 6]])
```

Μερικές ασκήσεις με την numpy

Ερώτηση. Πως εξάγουμε από έναν πίνακα τα στοιχεία που είναι περιττοί αριθμοί;

```
>>> v = np.array(1,11)
>>> v[ v % 2 == 1 ]
array([1, 3, 5, 7, 9])
```

Ερώτηση. Πως αντικαθστούμε τα στοιχεία ενός πίνακα που είναι περιττοί αριθμοί από το -1;

```
>>> v = np.array(1,11)
>>> v[ v % 2 == 1 ] = -1
>>> v
array([-1,  2, -1,  4, -1,  6, -1,  8, -1, 10])
```

Ερώτηση. Πως αντικαθστούμε τα στοιχεία ενός πίνακα που είναι περιττοί αριθμοί από το -1;

```
>>> v = np.array(1, 11)
>>> out = np.where(v % 2 == 1,  -1, v)
>>> out
array([-1,  2, -1,  4, -1,  6, -1,  8, -1, 10])
```

Ερώτηση. Πως αλλάζουμε το σχήμα ενός πίνακα;

```
>>> v = np.array(1,11)
>>> A = v.reshape(2, 5)    # ή v.reshape(2, -1)
>>> A
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

Μερικές ασκήσεις με την numpy

Ερώτηση. Πως μπορώ να αυξήσω τον αριθμό των γραμμών ενός πίνακα;

```
>>> A = np.arange(10).reshape(2,-1)
>>> B = np.zeros((3,5))

>>> np.vstack([A, B])
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

Μπορούμε να επιτύχουμε το ίδιο αποτέλεσμα με την εντολή

```
np.concatenate([A, B], axis=0)
```

Ερώτηση. Πως μπορώ να αυξήσω τον αριθμό των στηλών ενός πίνακα;

```
>>> A = np.arange(6).reshape(2,-1)
>>> B = np.zeros((2,3))

>>> np.hstack( [A,B] )
array([[0., 1., 2., 0., 0., 0.],
       [3., 4., 5., 0., 0., 0.]])
```

Μπορούμε να επιτύχουμε το ίδιο αποτέλεσμα με την εντολή

```
np.concatenate([A, B], axis=1)
```

Ερώτηση. Πως μπορώ να βρώ τα κοινά στοιχεία δύο διανυσμάτων;

```
>>> a = np.array([1, 2, 3, 2, 3, 4, 3, 4, 5, 6])
>>> b = np.array([7, 2, 10, 2, 7, 4, 9, 4, 9, 8])

>>> np.intersect1d(a,b)
array([2, 4])
```

Ερώτηση. Πως μπορούμε να αφαιρέσουμε από ένα διάνυσμα τα στοιχεία που εμφανίζονται σε ένα άλλο διάνυσμα;

```
>>> a = np.array([1, 2, 3, 4, 5])
>>> b = np.array([5, 6, 7, 8, 9])

>>> np.setdiff1d(a, b)
array([1, 2, 3, 4])
```

Ερώτηση. Πως μπορώ να βρώ τις θέσεις των κοινών στοιχείων δύο διανυσμάτων;

```
>>> a = np.array([1, 2, 3, 2, 3, 4, 3, 4, 5, 6])
>>> b = np.array([7, 2, 10, 2, 7, 4, 9, 4, 9, 8])

>>> np.where(a == b)
(array([1, 3, 5, 7]),)

>>> np.where(a==b, a, -b)
array([ -7,  2, -10,  2, -7, 4,  -9,  4,  -9,  -8])
```

Ερώτηση. Πως μπορώ να βρώ τις θέσεις των στοιχείων ενός διανύσματος που ικανοποιούν μια δεδομένη συνθήκη;

```
>>> a = np.array([-7,2,-10, 2,-7, 4, -9, 4,-9, -8])
>>> i = np.where( (a >= -8) & (a < 4) )

>>> i
(array([0, 1, 3, 4, 9]), )

>>> a[i]
array([-7,  2,  2, -7, -8])
```

Μερικές ασκήσεις με την numpy

Ερώτηση. Πως μπορώ να εναλλάξω τη θέση δύο γραμμών ενός πίνακα;

```
>>> A = np.arange(9).reshape(3,-1)
>>> A = A[[1, 0, 2], :]
>>> A
array([[3, 4, 5],
       [0, 1, 2],
       [6, 7, 8]])
```

Ερώτηση. Πως αντιστρέφω τη σειρά των γραμμών ενός πίνακα

```
>>> A = np.arange(9).reshape(3,-1)
>>> A[::-1]
array([[6, 7, 8],
       [3, 4, 5],
       [0, 1, 2]])
```

Ερώτηση. Πως μπορώ να εναλλάξω τη θέση δύο στηλών ενός πίνακα;

```
>>> A = np.arange(9).reshape(3,-1)
>>> A=A[:,[1,0,2]]
>>> A
array([[1, 0, 2],
       [4, 3, 5],
       [7, 6, 8]])
```

Ερώτηση. Πως αντιστρέφω τη σειρά των στηλών ενός πίνακα

```
>>> A = np.arange(9).reshape(3,-1)
>>> A[:, ::-1]
array([[2, 1, 0],
       [5, 4, 3],
       [8, 7, 6]])
```

Μιγαδικοί αριθμοί

Μιγαδικοί αριθμοί

Στην Python, η φανταστική μονάδα συμβολίζεται με `j`. Ο τύπος ενός μιγαδικού αριθμού στην Python είναι `complex`.

```
>>> z = complex(3,2)
>>> z
(3+2j)
```

Πράξεις μεταξύ μιγαδικών αριθμών γίνονται όπως αναμένεται

```
>>> z = 3+2j
>>> w = 1-1j
>>> z + w
(4+1j)

>>> z * w
(5 - 1j)
```

Μιγαδικοί αριθμοί

```
>>> z / w
(0.5+2.5j)

>>> z.real, z.imag
(3.0, 2.0)

>>> z.conjugate()    # συζυγής αριθμός του z
(3-2j)

>>> abs(3+4j)   # μέτρο του αριθμού 3 + 4j
5.0
```

Μιγαδικοί αριθμοί

Η βιβλιοθήκη `cmath` ορίζει ένα πλήθος μαθηματικών συναρτήσεων για μιγαδικούς αριθμούς. Μερικές από αυτές φαίνονται παρακάτω:

`cmath.polar(z)` Πολικές συντεταγμένες του μιγαδικού `z`. Επιστρέφει τη πλειάδα (r, phi) με r το μέτρο του και phi η γωνιακή συντεταγμένη

```
>>> cmath.polar(1+1j)
(1.4142135623730951, 0.7853981633974483)
```

`cmath.rect(r, phi)` Καρτεσιανές συντεταγμένες του μιγαδικού αριθμού με πολικές συντεταγμένες (r, phi).

```
>>> cmath.rect(1, cmath.pi/6)
(0.8660254037844387+0.49999999999999994j)
```

Μιγαδικοί αριθμοί

Ορίζονται ακόμα οι σταθερές

`cmath.pi`, `cmath.e`, `cmath.tau`

οι εκθετικές και λογαριθμικές συναρτήσεις

`cmath.exp`, `cmath.log`, `cmath.log10`, `cmath.sqrt`

για παράδειγμα,

```
>>> cmath.sqrt(-1)
1j
```

οι τριγωνομετρικές και αντίστροφες τριγωνομετρικές συναρτήσεις, και οι υπερβολικές και αντίστροφες υπερβολικές συναρτήσεις.