Λίστες στην Python - Μέρος 2ο

Μιχάλης Δρακόπουλος

Απρίλιος 2020

2 ΛΙΣΤΕΣ - Μέρος Β

2.1 Συναρτήσεις που "δουλεύουν" με λίστες (καθώς και με άλλους τύπους της Python)

[1]:
```
L = [21, 34, 82, 30, 92]
s = 'Python3_rulez!'
```

- Μήκος

[2]:
```
len(L), len(s)
```

[2]: (5, 14)

- Μέγιστο και ελάχιστο στοιχείο (εφόσον υπάρχει κάποια σχέση διάταξης)

[3]:
```
max(L), min(L)
```

[3]: (92, 21)

[4]:
```
max(s), min(s)
```

[4]: ('z', '!')

- Άθροισμα στοιχείων (εφόσον είναι αριθμητικού τύπου)

[5]:
```
sum(L)
```

[5]: 259

Παράδειγμα: Μέσος όρος στοιχείων λίστας

[6]:
```
print('M.O. = ', sum(L)/len(L))
```

```
M.O. =  51.8
```

- Διάταξη στοιχείων (εφόσον είναι συγκρίσιμα). Επιστρέφει νέα λίστα

[7]:
```
M = sorted(L)
print(L)
```

```
print(M)
```

```
[21, 34, 82, 30, 92]
[21, 30, 34, 82, 92]
```

[8]:
```
M = sorted(L, reverse = True)
print(L)
print(M)
```

```
[21, 34, 82, 30, 92]
[92, 82, 34, 30, 21]
```

[9]:
```
T = sorted(s)
T
```

[9]:
```
['!', '3', 'P', '_', 'e', 'h', 'l', 'n', 'o', 'r', 't', 'u', 'y', 'z']
```

2.2 Μέθοδοι για λίστες

Οι **μέθοδοι** για λίστες είναι "συναρτήσεις" εξειδικευμένες για λίστες. Όπως είδαμε και στις συμβολοσειρές, καλούνται "πάνω" στο συγκεκριμμένο αντικείμενο στο οποίο πρόκειται να εφαρμοστουν > `object.method(parameters)`

"Λίστα" με τις μεθόδους για λίστες

[10]:
```
dir(list)
```

[10]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
```

```
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

Προς το παρόν, μας ενδιαφέρουν οι μέθοδοι που τα ονόματά τους δεν ξεκινούν με `__`:

[11]:
```
dir(list)[-11:]
```

[11]:
```
['append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

Όλες, εκτός από την `copy`, την `count` και την `index`, **αλλάζουν την ίδια τη λίστα πάνω στην οποία εφαρμόζονται!**

Πληροφορίες για θέση/δείκτη και πλήθος εμφανίσεων στοιχείου: `index` και `count`

- Ευρέση της θέσης/δείκτη στοιχείου: `index`

[12]:
```
M = dir(list)[-11:]
M
```

[12]:
```
['append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

[13]:
```
M.index('copy')
```

[13]:
```
2
```

Παράδειγμα: Θέση μεγίστου και ελαχίστου στοιχείου λίστας

[14]:
```
X = [5, 100, 0, -1, 100, -1, 12]
print(X.index(max(X)))
print(X.index(min(X)))
```

```
1
3
```

- Πλήθος εμφανίσεων στοιχείου: `count`

[15]:
```
[10, 5, 10, 1, 1, 10].count(10)
```

[15]:
```
3
```

[16]:
```
[2, 3, 4, 88].count(22)
```

[16]:
```
0
```

Μέθοδοι που αλλάζουν τη σειρά των στοιχείων της λίστας: `reverse` και `sort`

[17]:
```
L = [10, 1, 8, 14, -1, 5, -4]
L
```

[17]:
```
[10, 1, 8, 14, -1, 5, -4]
```

- Αντιστροφή στοιχείων λίστας - `reverse`

[18]:
```
L.reverse()
L
```

[18]:
```
[-4, 5, -1, 14, 8, 1, 10]
```

- Διάταξη μεγέθους (αύξουσα ή φθίνουσα) στοιχείων λίστας - `sort`

[19]:
```
L = list(range(8))
L
```

[19]:
```
[0, 1, 2, 3, 4, 5, 6, 7]
```

[20]:
```
L.sort()
L = list(range(10, 1, -1))
```

[21]:
```
L
```

[21]:
```
[10, 9, 8, 7, 6, 5, 4, 3, 2]
```

[22]:
```
L.sort()
L
```

[22]:
```
[2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[23]:
```
L.sort(reverse = True)
L
```

[23]:
```
[10, 9, 8, 7, 6, 5, 4, 3, 2]
```

Σε αντίθεση με τη **συνάρτηση** `sorted` που είδαμε παραπάνω, η **μέθοδος** `sort` **δεν επιστρέφει μια νέα λίστα!** Ολες οι μέθοδοι για λίστες (εκτός της `copy`) αλλάζουν τη λίστα στην οποία αναφέρονται και δεν επιτρέφουν μια νέα λίστα. Είναι επομένως **λάθος** η εκχώρηση αποτελέσματος:

[24]:
```
M = sorted(L)
print(L)
print(M)
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2]
[2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[25]:
```
M = L.sort()
print(L)
print(M)
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10]
None
```

[26]:
```
L = L.sort()
print(L)
```

```
None
```

Το αντικείμενο στο οποίο αναφέρεται αρχικά η `L` διατάσσεται μεν, αλλά αμέσως μετά την εκχώρηση η μεταβλητή `L` αναφέρεται στο αντικείμενο `None`.

- Διάταξη με βάση την τιμή κάποιας συνάρτησης `function` που εφαρμόζεται σε κάθε στοιχείο.

  Μια χρήσιμη παράμετρος των `sort` των `sorted`: `key = function`

Η διάταξη προκύπτει εφαρμόζοντας (*μόνο για τις ανάγκες της σύγκρισης*) πρώτα τη `function` σε κάθε στοιχείο της λίστας.

Παραδείγματα

1. Λεξικογραφική διάταξη αγνοώντας κεφαλαία-μικρά

[27]:
```
words = ['Xyz', 'aBc', 'ABb']
sorted(words)
```

[27]:
```
['ABb', 'Xyz', 'aBc']
```

[28]:
```
sorted(words, key=str.lower)
```

[28]:
```
['ABb', 'aBc', 'Xyz']
```

[29]:
```
words
```

[29]:
```
['Xyz', 'aBc', 'ABb']
```

[30]:
```
words.sort(key=str.lower)
```

[31]:
```
words
```

[31]:
```
['ABb', 'aBc', 'Xyz']
```

2. Διάταξη με βάση το μήκος των στοιχείων

[32]:
```
names = ['Python', 'C', 'Java', 'Ada']
```

[33]:
```
sorted(names)
```

[33]:
```
['Ada', 'C', 'Java', 'Python']
```

[34]:
```
sorted(names, key=len)
```

[34]:
```
['C', 'Ada', 'Java', 'Python']
```

[35]:
```
names
```

[35]:
```
['Python', 'C', 'Java', 'Ada']
```

[36]:
```
names.sort(key=len)
names
```

[36]:
```
['C', 'Ada', 'Java', 'Python']
```

Μέθοδοι που "προσθέτουν" νέα στοιχεία σε μια λίστα: `insert`, `append`, `extend`.

- Εισαγωγή στοιχείου **πριν** από κάποια θέση (δείκτη) - `insert`

[37]:
```
L = list(range(10,15))
L
```

[37]:
```
[10, 11, 12, 13, 14]
```

[38]:
```
L.insert(1, 'spam')
L
```

[38]:
```
[10, 'spam', 11, 12, 13, 14]
```

[39]:
```
L.insert(-3, -9999)
L
```

[39]:
```
[10, 'spam', 11, -9999, 12, 13, 14]
```

[40]:
```
L.insert(0, 'START')
L
```

[40]:
```
['START', 10, 'spam', 11, -9999, 12, 13, 14]
```

[41]:
```
len(L)
```

[41]:
```
8
```

[42]:
```
L.insert(100, 'END')
L
```

[42]:
```
['START', 10, 'spam', 11, -9999, 12, 13, 14, 'END']
```

- Προσθήκη *ενός αντικειμένου* στο τέλος - `append`

[43]:
```
L = list(range(10, 16))
L
```

[43]:
```
[10, 11, 12, 13, 14, 15]
```

[44]:
```
L.append(42)
L
```

[44]: `[10, 11, 12, 13, 14, 15, 42]`

[45]:
```
L.append([42])
L
```

[45]: `[10, 11, 12, 13, 14, 15, 42, [42]]`

[46]:
```
L.append([1, 2, 3])
L
```

[46]: `[10, 11, 12, 13, 14, 15, 42, [42], [1, 2, 3]]`

Παρατήρηση: Παρόμοιο αποτέλεσμα θα παίρναμε με τον τελεστή συνένωσης, ο οποίος όμως δεν αλλάζει τη λίστα `L` αλλά δημιουργεί ένα νέο αντικείμενο που θα πρέπει να το συσχετίσουμε πάλι με τη μεταβλητή `L`. Στην περίπτωση αυτή και οι 2 όροι πρέπει να είναι λίστες. Η `append` είναι πιο γρήγορη.

[47]:
```
L = [1, 2, 3]
L = L + [42]
L
```

[47]: `[1, 2, 3, 42]`

[48]:
```
L = L + [[42]]
L
```

[48]: `[1, 2, 3, 42, [42]]`

- Επέκταση με στοιχεία αντικειμένου που τηρεί το **πρωτόκολλο επανάληψης** - `extend`

[49]:
```
L = list(range(10,16))
L
```

[49]: `[10, 11, 12, 13, 14, 15]`

[50]:
```
L.extend(42)
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-50-654052f28172> in <module>() ----> 1 L.extend(42)
```

```
TypeError: 'int' object is not iterable
```

[51]:
```
L.extend([42])
L
```

[51]:
```
[10, 11, 12, 13, 14, 15, 42]
```

[52]:
```
L.extend([99, 100])
L
```

[52]:
```
[10, 11, 12, 13, 14, 15, 42, 99, 100]
```

[53]:
```
L.extend('eggs')
L
```

[53]:
```
[10, 11, 12, 13, 14, 15, 42, 99, 100, 'e', 'g', 'g', 's']
```

[54]:
```
L.extend(range(888,891))
L
```

[54]:
```
[10, 11, 12, 13, 14, 15, 42, 99, 100, 'e', 'g', 'g', 's', 888, 889, 890]
```

Παρατήρηση: Ο τελεστής επαυξημένης εκχώρησης `+=`, όταν εφαρμόζεται σε λίστα καλεί την `extend` και δεν δημιουργεί νέο αντικείμενο.

[55]:
```
L = [1, 2, 3]
L += [99, 100]
L
```

[55]:
```
[1, 2, 3, 99, 100]
```

[56]:
```
L += 'spam'
L
```

[56]:
```
[1, 2, 3, 99, 100, 's', 'p', 'a', 'm']
```

[57]:
```
L += range(3)
L
```

[57]:
```
[1, 2, 3, 99, 100, 's', 'p', 'a', 'm', 0, 1, 2]
```

Μέθοδοι που "αφαιρούν" στοιχεία από μια λίστα: `clear`, `pop`, `remove` **και η εντολή** `del`

[58]:
```
L
```

[58]:
```
[1, 2, 3, 99, 100, 's', 'p', 'a', 'm', 0, 1, 2]
```

- Διαγραφή όλων των στοιχείων - `clear`

[59]:
```
L.clear()
L
```

[59]:
```
[]
```

[60]:
```
L = list(range(10,18))
L
```

[60]:
```
[10, 11, 12, 13, 14, 15, 16, 17]
```

- Διαγραφή στοιχείου με βάση τη θέση του στη λίστα και επιστροφή της τιμής του - `pop`

[61]:
```
element = L.pop()
print(element)
print(L)
```

```
17
[10, 11, 12, 13, 14, 15, 16]
```

[62]:
```
new = L.pop(2)
print(new)
print(L)
```

```
12
[10, 11, 13, 14, 15, 16]
```

- Διαγραφή στοιχείου με βάση την τιμή του - `remove`

[63]:
```
L.remove(11)
L
```

[63]:
```
[10, 13, 14, 15, 16]
```

- Διαγραφή με την εντολή `del`

[64]:
```
del L[2]
L
```

[64]:
```
[10, 13, 15, 16]
```