ΜΕΜ104 Γλώσσα Προγραμματισμού Ι

Μιχάλης Πλεξουσάκης, Ιωάννης Λιλής

9-13 Νοεμβρίου 2019

Μαθηματικά και Εφαρμοσμένα Μαθηματικά

Περιεχόμενα

Λίστες

Λίστες

Λίστα είναι μια διατεταγμένη ακολουθία τιμών όπου κάθε τιμή προσδιορίζεται από έναν αριθμοδείκτη (index).

Μια λίστα περιγράφεται γράφοντας τα στοιχεία της μέσα σε αγκύλες. Για παράδειγμα,

```
L = ['MEM104', 'Programming', 'Thursday 11-1']
```

Μια κενή λίστα μπορεί να γραφεί ως `[]`. Η προσπέλαση και ο τεμαχισμός (slicing) λειτουργούν όπως και στις συμβολοσειρές:

```
print(L[-1])
print(L[1:])
for i in range(len(L)):
    print(L[i])
```

Λίστες

Μπορούμε να αποφύγουμε τη χρήση του αριθμοδείκτη κατά τη διάνυση μιας λίστας. Για παράδειγμα, για τη διάνυση της λίστας

```
L = ['MEM104', 'Programming', 'Thursday 11-1']
```

μπροούμε να γράψουμε

```
for i in range(len(L)):
    print(L[i])
```

αλλά και

```
for e in L:
    print(e)
```

το οποίο είναι απλούστερο, και πιο γρήγορο από τη χρήση αριθμοδεικτών!

Λίστες

Μπορούμε να διατρέξουμε τα στοιχεία μιας λίστας με την ανάποδη σειρά χρησιμοποιώντας τις εντολές

```
for i in range(len(L)-1,-1,-1):
    print(L[i])
```

ή ευκολότερα με

```
for e in reversed(L):
    print(e)
```

Σημείωση: Η συνάρτηση `reversed()` δουλεύει τόσο για λίστες, όσο και για αντικείμενα τύπου `string` αλλά και `range` (και με `tuple` που θα δούμε αργότερα)!!

Λίστες

Μπορούμε ακόμα να διανύσουμε τόσο τα στοιχεία μιας λίστας όσο και τους αριθμοδείκτες με χρήση της συνάρτησης `enumerate`. Αν

```
colors = ['red', 'green', 'blue', 'yellow']
```

τότε οι εντολές

```
for i, color in enumerate(colors):
    print(i, '-->', color)
```

εκτυπώνουν

```
0 --> red
1 --> green
2 --> blue
3 --> yellow
```

Λίστες

Μπορούμε να διατρέξουμε δύο λίστες ταυτόχρονα με την εντολή `zip`. Αν

```
names = ['George', 'Maria', 'Michael']
colors = ['red', 'green', 'blue', 'yellow']
```

τοτε οι εντολές

```
for name, color in zip(names, colors):
    print(name, '-->', color)
```

θα εκτυπώσει

```
George --> red
Maria --> green
Michael --> blue
```

Λίστες

Οι λίστες είναι *μεταλλάξιμες* (mutable), δηλαδή μπορούμε να αλλάξουμε την τιμή των στοιχείων τους μετά τη δημιουργία τους:

```
>>> L = ['MEM104', 'Programming', 'Thursday 11-1']
>>> print(L)
['MEM104', 'Programming', 'Thursday 11-1']

>>> L[1] = 'Programming I'
>>> L[2] = 'Tuesday 11-1'
>>> print(L)
['MEM104', 'Programming I', 'Tuesday 11-1']
```

Λίστες

Πολύ συχνά, οι λίστες δημιουργούνται ξεκινώντας από μια κενή λίστα και προσθέτοντας στοιχεία στο τέλος της με τη μέθοδο `append()`:

```
>>> L = []
>>> L.append('MEM104')
>>> L
['MEM104']
>>> L.append('Programming I')
>>> L.append('Thursday 11-1')

>>> L
['MEM104', 'Programming I', 'Thursday 11-1']
```

Λίστες

Μπορούμε ακόμα να φτιάξουμε λίστες μετατρέποντας άλλες ακολουθίες με την εντολή `list`. Για παράδειγμα, μπορούμε να μετατρέψουμε τις ακολουθίες που παράγει η `range` σε λίστες

```
>>> L = list(range(1,11))
>>> L
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> M = list(range(19,0,-2))
>>> M
[19, 17, 15, 13, 11, 9, 7, 5, 3, 1]
```

αλλά και συμβολοσειρές σε λίστες:

```
>>> print(list('Python'))
['P', 'y', 't', 'h', 'o', 'n']
```

Λίστες

Το παρακάτω πρόγραμμα διαβάζει συμβολοσειρές και τις αποθηκεύει στη λίστα `L`. Η είσοδος τερματίζεται όταν ο χρήστης δώσει μια κενή συμβολοσειρά:

```
L = []
while True:
    s = input('Enter string: ')
    if not s:
        break
    L.append(s)
print(L)
```

Παρατηρήστε ότι η ατέρμονη επανάληψη `while` τερματίζεται με την εντολή `break` αν η συμβολοσειρά `s` είναι κενή.

Λίστες

Μπορούμε να προσαρτήσουμε μια ολόκληρη λίστα σε μια άλλη. Το αποτέλεσμα είναι μια λίστα μέσα σε μια άλλη λίστα:

```
>>> L = [1, 2, 3]
>>> M = [4, 5, 6]
>>> L.append(M)
>>> L
[1, 2, 3, [4, 5, 6]]
```

Παρατηρήστε ότι τότε θα έχουμε:

```
>>> L[1]
2
>>> L[3]
[4, 5, 6]
>>> L[3][1]
5
```

Λίστες

Συνένωση λιστών μπορούμε να επιτύχουμε είτε με τον τελεστή `+` είτε με τη μέθοδο `extend`:

```
>>> L = [1, 2, 3]
>>> M = [4, 5, 6]
>>> K = L + M
>>> K
[1, 2, 3, 4, 5, 6]

>>> K.extend([7, 8, 9, 10])
>>> K
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Λίστες

Άλλες χρήσιμες μέθοδοι για το χειρισμό των λιστών είναι:

- `L.insert(i, e)` Εισάγει το αντικείμενο `e` στη λίστα `L`, στη θέση `i`.
- `L.remove(e)` Διαγράφει την πρώτη παρουσία του αντικειμένου `e` από τη λίστα `L`.
- `L.pop(i)` Αφαιρεί και επιστρέφει το στοιχείο της λίστας `L` στη θέση `i`. Αν το `i` παραληφθεί, τότε νοείται ως -1 και επιστρέφεται το τελευταίο στοιχείο της λίστας.
- `L.index(e)` Επιστρέφει τον δείκτη θέσης της πρώτης εμφάνισης του `e` στη λίστα `L`. Εμφανίζεται μήνυμα σφάλματος αν το `e` δεν είναι στη λίστα.

Λίστες

- `L.count(e)` Επιστρέφει τον αριθμό των φορών που εμφανίζεται το `e` στη λίστα `L`.
- `L.sort()` Ταξινομεί τα στοιχεία της λίστας σε αύξουσα σειρά,
- `L.reverse()` Αντιστρέφει τη σειρά των στοιχείων της λίστας `L`.

Η Python έχει πολλές ακόμα μεθόδους για την επεξεργασία λιστών. Πληροφορίες μπορεί να δεί κανείς γράφοντας `help(list)` στη γραμμή εντολών της Python.

Λίστες

Παράδειγμα. Αν θέλουμε να ταξινομήσουμε, κατ' αύξουσα σειρά, τα στοιχεία της λίστας `L` αρκεί να γράψουμε `L.sort()`.

Για να δημιουργήσουμε μια καινούργια ταξινομημένη λίστα από τα στοιχεία της `L` μπορούμε να πούμε `M = sorted(L)`.

```
>>> L = [37, 12, 41, 21, 3]
>>> L.sort()
>>> L
[3, 12, 21, 37, 41]

>>> L = [37, 12, 41, 21, 3]
>>> M = sorted(L)
>>> M
[3, 12, 21, 37, 41]
```

Λίστες

Για να ταξινομήσουμε μια λίστα κατα φθίνουσα σειρά μπορούμε να χρησιμοποιήσουμε το προαιρετικό όρισμα `reverse = True`.

```
>>> L = [37, 12, 41, 21, 3]
>>> L.sort(reverse = True)
>>> L
[41, 37, 21, 12, 3]

>>> L = [37, 12, 41, 21, 3]
>>> M = sorted(L, reverse = True)
>>> M
[41, 37, 21, 12, 3]
```

Λίστες

Η ταξινόμηση συμβολοσειρών ακολουθεί τη λεξικογραφική σειρά:

```
>>> L = ['Mary', 'George', 'Helen', 'Michael']
>>> M = sorted(L)
>>> M
['George', 'Helen', 'Mary', 'Michael']

>>> L = ['Mary', 'George', 'Helen', 'Michael']
>>> M = sorted(L, reverse=True)
>>> M
['Michael', 'Mary', 'Helen', 'George']
```

Λίστες

Παράδειγμα. Ας υποθέσουμε ότι η λίστα L περιέχει θετικούς ακέραιους. Ο παρακάτω κώδικας κατασκευάζει τη λίστα P η οποία περιέχει τους περιττούς αριθμούς της λίστας L και τη λίστα E η οποία περιέχει του άρτιους αριθμούς της λίστας L:

```
P = []
E = []
for e in L:
    if e % 2 != 0:
        P.append(e)
    else:
        E.append(e)
```

Λίστες

Παράδειγμα. Ο κώδικας που ακολουθεί αποθηκεύει στη λίστα `M` τα κοινά στοιχεία των λιστών `L1` και `L2`:

```
M = []
for e in L1:
    if e in L2:
        if e not in M:
            M.append(e)
```

Ο έλεγχος `if e not in M` γίνεται έτσι ώστε η λίστα `M` να περιέχει κάθε κοινό στοιχείο μία και μόνο μία φορά.

Λίστες

Παράδειγμα. Ο κώδικας που ακολουθεί αφαιρεί από τη λίστα `L` πιθανά αντίγραφα των στοιχείων της.

```
M = []
for e in L:
    if e not in M:
        M.append(e)
```

Παράδειγμα. Ο κώδικας που ακολουθεί υπολογίζει την ένωση των λιστών `L1` και `L2`:

```
M = []
for e in L1:
    if e not in M: M.append(e)
for e in L2:
    if e not in M: M.append(e)
```

Λίστες

Παράδειγμα. Έστω ότι όλα τα στοιχεία της λίστας `L` είναι ίσα είτε με 1 είτε με 2. Θα μπορούσαμε να ταξινομήσουε τα στοιχεία της είτε με τη μέθοδο `sort()` είτε με τον κώδικα που ακολουθεί:

```
l = 0
r = len(L)-1
while l < r:
    while L[l] == 1:
        l += 1
    while L[r] == 2:
        r -= 1
    if l < r:
        t = L[l];  L[l] = L[r];  L[r] = t
```

Λίστες

Το πρόβλημα της Ολλανδικής σημαίας. Τα στοιχεία της λίστας L είναι όλα ίσα με 1 ή 2 ή 3. Μπορούμε μα ταξινομήσουμε τα στοιχεία της με τον παρακάτω κώδικα:

```
i = 0;  j = 0;  n = len(L)-1

while j <= n:
    if L[j] == 1:
        t = L[i];  L[i] = L[j];  L[j] = t
        i += 1;  j += 1
    elif L[j] == 3:
        t = L[j];  L[j] = L[n];  L[n] = t
        n -= 1
    else:
        j += 1
```

Λίστες

Αν θέλαμε να κατασκευάσουμε μια λίστα με τα τετράγωνα των ακέραιων αριθμών στο διάστημα $[1, 20]$ θα μπορούσαμε να γράψουμε, για παράδειγμα,

```
squares = []
for i in range(1,21):
    squares.append(i**2)
```

Ένας συντομότερος τρόπος ο οποίος επιτρέπει την εφαρμογή μιας πράξης στις τιμές μιας λίστας ή γενικότερα μιας ακολουθίας είναι η λεγόμενη *συμπερίληψη λίστας* (list comprehension):

```
squares = [i**2 for i in range(1, 21)]
```

Λίστες

Ο όρος `for` μπορεί να ακολουθείται από μια ή περισσότερες εντολές `if` ή `for` οι οποίες εφαρμόνται στις τιμές της ακολουθίας που παράγονται από την πρώτη εντολή `for`. Για παράδειγμα, η λίστα `squares` μετά την εκτέλεση της εντολής

```
squares = [i**2 for i in range(1, 21) if i%3 == 0]
```

θα είναι η `[9, 36, 81, 144, 225, 324]`.

Ακόμα, οι εντολές

```
L = [2, 'a', 3.1, 3, 'maria', 4.1, -9.0, 5]
print([x**2 for x in L if type(x) == int])
```

θα εκτυπώσουν τη λίστα `[4, 9, 25]`.

Λίστες

Γενικότερα, μπορούμε να εφαρμόσουμε οποιαδήποτε συνάρτηση στα στοιχεία μιας λίστας, χρησιμοποιώντας κώδικα όπως

```
def applyToEach(L, f):
    for i in range(len(L)):
        L[i] = f( L[i] )

L = [2, 1, -3.3, -4]
applyToEach(L, abs)
```

Μπορούμε να αντικαστήσουμε το όρισμα `abs` στη κλήση της συνάρτησης `applyToEach` από οποιαδήποτε άλλη συνάρτηση (με μια τυπική παράμετρο) είτε ενσωματωμένη είτε δική μας.

Λίστες

Η Python παρέχει την ενσωματωμένη συνάρτηση `map` με παρόμοια (αν και πιο γενική) λειτουργία. Για παράδειγμα, μετά την εκτέλεση του προγράμματος

```
def square(x):
    return x*x

L = [1, -3, 12, 7]
L = list(map(square, L))
```

η λίστα `L` θα είναι η `[1, 9, 144, 49]`.

Γενικότερα, το πρώτο όρισμα της `map` μπορεί να είναι μια συνάρτηση με n ορίσματα οπότε σε αυτή την περίπτωση πρέπει να ακολουθείται από n ακολουθίες, για παράδειγμα, λίστες.

Λίστες

```
def addition(x, y):
    return x+y

L = [1, -3, 12, 7]
M = [9, 3, -1, 4]
K = list(map(addition, L, M))

fistName = ['Mickey', 'Peter', 'Tony']
lastName = ['Mouse', 'Parker', 'Stark']
names = list(map(addition, firstName, lastName))
```

Λίστες

Σημαντικό. Τόσο οι συμβολοσειρές (str) όσο και οι λίστες (list) αλλά και οι πλειάδες (tuple) που θα δούμε αργότερα αποτελούν διαφορετικούς τύπους ακολουθιών (sequences).

Ένα κοινό χαρακτηριστικό που έχουν αυτοί οι τύποι είναι ότι μπορούμε να εφαρμόσουμε στα αντικείμενά τους τις παρακάτω πράξεις:

- `seq[i]` για το `i`-στό στοιχείο της ακολουθίας
- `seq[i:j]` για τους χαρακτήρες από τη θέση `i` μέχρι και τη θέση `j-1`
- `len(seq)` για το μήκος της ακολουθίας
- `seq1 + seq2` για τη συνένωση των δύο ακολουθιών
- `n * seq` για την επανάληψη της ακολουθίας `seq` `n` φορές

Λίστες

αλλά και τις

- `e in seq` επιστρέφει `True` αν `e` περιέχεται στην ακολουθία `seq`
- `e not in seq` επιστρέφει `True` αν `e` den περιέχεται στην ακολουθία `seq`
- `for e in seq` για τη σειριακή διάνυση της ακολουθίας `seq`