Λίστες στην Python

Η βασικώτερη δομή δεδομένων στην Pyhton είναι η **ακολουθία (sequence)**. Σε κάθε στοιχείο μιας ακολουθίας ανατίθεται ένας ακέραιος που ονομάζεται *δείκτης* (index) *ή θέση* του στοιχείου. Ο δείκτης του πρώτου στοιχείου μιας ακολουθίας είναι μηδέν, του δεύτερου ένα, του τρίτου δύο κ.ο.κ. Η Python προσφέρει επτά τύπους ακολουθιών με πιο κοινό τον τύπο `list` (λίστα).

Μπορούμε να ορίσουμε μια λίστα απαριθμώντας τα στοιχεία της μέσα σε τετραγωνικές παρανθέσεις και χωρίζοντας το ένα στοιχείο από το άλλο με κόμμα:

```
>>> numbers = [3, 21, 8, 17, 9, 5]
>>> names = ['John', 'Maria', 'Dina', 'Mike']
>>> course_grade = ['Math', 5, 'Chemistry', 8, 'Physics', 7]
```

Αναφερόμαστε σε κάποιο συγκεκριμένο στοιχείο μιας λίστας γράφοντας το όνομα της λίστας ακολουθούμενο από το δείκτη του στοιχείου που μας ενδιαφέρει μέσα σε τετραγωνικές παρενθέσεις:

```
>>> print 'My favourite number is', numbers[3]
My favourite number is 17
>>> names[0]
'John'
>>> print 'My grade in', course_grade[2], 'was', course_grade[3]
My grade in Chemistry was 8
```

Η *συνάρτηση* `len()` επιστρέφει το πλήθος των στοιχείων μιας λίστας:

```
>>> len(names)
4
>>> print 'The length of list course_grades is', len(course_grade)
The length of list course_grades is 6
```

Μπορούμε να αναφερθούμε σε ένα κομμάτι (slice) ή υπολίστα μιας λίστας, π.χ. της `numbers`, χρησιμοποιώντας το συμβολισμό `numbers[i:j]` ή `numbers[i:j:step]`. Για παράδειγμα,

```
>>> numbers[0:2]
[3, 21]
>>> numbers[1:4]
[21, 8, 17]
>>> numbers[1:23]
[21, 8, 17, 9, 5]
>>> numbers[13:17]
[]
>> numbers[0:5:2]
[3, 8, 9]
>>> numbers[4:23:3]
[9]
```

Προσέξτε ότι κάθε μια από τις παραπάνω εντολές έχει σαν αποτέλεσμα μια λίστα. Οι κανόνες που διέπουν τον τρόπο αναφοράς σε κομμάτια μιας λίστας έχουν ως εξής:

- Αν ο δείκτης `i` είναι μικρότερος από τον `-len(list)` αντικαθίσταται με μηδέν. Διαφορετικά, αν ο δείκτης `i` είναι αρνητικός αντικαθίσταται από τον `i + len(list)`. Ομοίως για τον δείτη `j`.

- Το κομμάτι, `list[i:j]`, μιας λίστας `list` ορίζεται ως η λίστα με στοιχεία `list[k]`, όπου `i <= k < j`. Αν είτε ο δείκτης `i` είτε ο δείκτης `j` είναι μεγαλύτεροι από `len(list)` αντικαθίστανται με `len(list)`. Αν ο πρώτος δείκτης `i` έχει παραληφθεί αντικαθίσταται με 0 (μηδέν). Αν ο δεύτερος δείκτης `j` έχει παραληφθεί αντικαθίσταται με `len(list)`. Αν ο δείκτης `i` είναι μεγαλύτερος ή ίσος από τον δείκτη `j` το κομμάτι της λίστας είναι κενό.
- Η υπολίστα `list[i:j:step]` ορίζεται ως η λίστα με στοιχεία `list[i+n*step]`, όπου ο ακέραιος `n` είναι τέτοιος ώστε `0 <= n < (j-i) / step`. Αν ο ακέραιος έχει παραληφθεί αντικαθίσταται με 1. Αν είτε ο δείκτης `i` είτε ο δείκτης `j` είναι μεγαλύτεροι από `len(list)` αντικαθίστανται με `len(list)`. Αν ένας ή και οι δύο δείκτες έχουν παραληφθεί αντικαθίστανται είτε από `0` είτε από `len(list)`, ανάλογα με το πρόσημο του `step`.

Ακολουθούν μερικά ακόμα παραδείγματα με επεξηγήσεις:

```
>>> numbers[2:3]
[8]
```

Αναφερόμαστε στη λίστα με στοιχεία `numbers[k]` όπου `2 <= k < 3`, επομένως στη λίστα με μοναδικό στοιχείο το `numbers[2]`.

```
>>> numbers[1:]
[21, 8, 17, 9, 5]
```

Μια και ο δεύτερος δείκτης έχει παραληφθεί, αντικαθίσταται με `len(numbers)`, δηλαδή 6. Αναφερόμαστε δηλαδή στην υπολίστα με στοιχεία `numbers[k]` όπου `1 <= k < 6`.

```
>>> numbers[:3]
[3, 21, 8]
```

Εδώ έχει παραληφθεί ο πρώτος δείκτης και επομένως αντικαθίσταται με 0 (μηδέν). Αναφερόμαστε δηλαδή στην υπολίστα με στοιχεία `numbers[k]` όπου `0 <= k < 3`.

```
>>> numbers[:]
[3, 21, 8, 17, 9, 5]
```

Κατ' αναλογία με τα δύο προηγούμενα παραδείγματα, η παραπάνω εντολή είναι ισοδύναμη με την `numbers[0:6]`, αναφερόμαστε δηλαδή σε ολόκληρη τη λίστα `numbers`.

```
>>> numbers[2:-1]
[8, 17, 9]
```

Εδώ ο δείκτης `-1` αντικαθίσταται από τον `-1+len(numbers) = 5`. Αναφερόμαστε δηλαδή στην υπολίστα `numbers[2:5]`.

```
>>> numbers[-4:-2]
[8, 17]
```

Ο πρώτος δείκτης αντικαθίσταται από το `2` και ο δεύτερος από το `4`. 'Ετσι αναφερόμαστε στην υπολίστα `numbers[2:4]`.

```
>>> numbers[-4:101]
[8, 17, 9, 5]
```

Όπως στο προηγούμενο παράδειγμα, ο δείκτης `-4` αντικαθίσταται από τον δείκτη `2` ενώ ο δείκτης `101`, επειδή είναι μεγαλύτερος από το `len(numbers)`, αντικαθίσταται από τον δείκτη `len(numbers)`, δηλαδή τον δείκτη `6`. Αναφερόμαστε λοιπόν στην υπολίστα `numbers[2:6]`.

```
>>> numbers[-11:5]
[3, 21, 8, 17, 9]
```

Εδώ, ο δείκτης -11 είναι μικρότερος από τον δείκτη -len(numbers) οπότε αντικαθίσταται από τον δείκτη 0. Αναφερόμαστε λοιπόν στην υπολίστα numbers[0:5].

```
>>> numbers[-101:-5]
[3]
```

Όπως στο προηγούμενο παράδειγμα, ο δείκτης -101 αντικαθίσταται από τον δείκτη 0. Ο δείκτης -5 αντικαθίσταται από τον δείκτη 1 οπότε αναφερόμαστε στην υπολίστα numbers[0:1].

```
>>> numbers[1:-3:2]
[21]
```

Ο δείκτης -3 αντικαθίσταται από τον δείκτη 3. Αναφερόμαστε στη λίστα numbers[1:3:2] δηλαδή στη λίστα [21].

```
>>> numbers[5:0:-2]
[5, 17, 21]
```

Μια και το βήμα step είναι αρνητικό η υπολίστα αποτελείται από τα στοιχεία numbers[5], numbers[3] και numbers[1].

```
>>> numbers[:-1:-2]
[5, 17]
```

Ο πρώτος δείκτης έχει παραληφθεί. Αντικαθίσταται από 6 μια και το βήμα είναι αρνητικό. Ο δείκτης -1 αντικαθίσταται από 5. Αναφερόμαστε λοιπόν στην υπολίστα numbers[5:1:-2].

```
>>> numbers[10:-13:-2]
[5, 17, 21]
```

Ο δείκτης 10 αντικαθίσταται από 6 μια και το βήμα είναι αρνητικό. Ο δείκτης -13 αντικαθίσταται από 0. Αναφερόμαστε λοιπόν στην υπολίστα numbers[6:0:-2].

```
>>> numbers[-101:101:3]
[3, 17]
```

Ο δείκτης -101 αντικαθίσταται από τον δείκτη 0. Ο δείκτης 101 από τον δείκτη 6. Η υπολίστα στην οποία αναφερόμαστε είναι η numbers[0:6:3].

Εισαγωγή στοιχείων σε μια λίστα. Η συνάρτηση append() προσθέτει το όρισμά της σε μιά λίστα. Για παράδειγμα,

```
>>> names.append('Sofia')
>>> names
['John', 'Maria', 'Dina', 'Mike', 'Sofia']
```

Η συνάρτηση extend(L) επεκτείνει μια λίστα προσαρτώντας τα στοιχεία της λίστας L. Μπορούμε να επιτύχουμε το ίδιο αποτέλεσμα με τον τελεστή '+'.

```
>>> names.extend(['Olga', 'Mark')
>>> names
['John', 'Maria', 'Dina', 'Mike', 'Sofia', 'Olga', 'Mark']
>>> names = names + ['Laura']
>>> names
'John', 'Maria', 'Dina', 'Mike', 'Sofia', 'Olga', 'Mark', 'Laura']
```

Η συνάρτηση `insert(i, x)` προσθέτει το αντικείμενο `x` πρίν ακριβώς τη θέση `i`. Αν η θέση δωθεί ως 0, το αντικείμενο προστίθεται στην αρχή της λίστας. Αν ως θέση δωθεί το μήκος της λίστας το αντικείμενο προσαρτάται στη λίστα:

```
>>> names.insert(3, 'Jenny')
>>> names
['John', 'Maria', 'Dina', 'Jenny', 'Mike', 'Sofia', 'Olga', 'Mark', 'Laura']
```

Παρατηρήστε ότι το στοιχείο `'Jenny'` είναι τώρα το στοιχείο με δείκτη `3`.

Διαγραφή στοιχείων από μια λίστα. Η συνάρτηση `remove(x)` διαγράφει από μιά λίστα το πρώτο στοιχείο με τιμή ίση με `x`. Μπορούμε να διαγράψουμε το στοιχείο της λίστας `list` στη θέση `i` γράφοντας `del list[i]`

```
>>> names.remove('Mike)
>>> names
['John', 'Maria', 'Dina', 'Jenny', 'Sofia', 'Olga', 'Mark', 'Laura']
>>> del names[1]
>>> names
['John', 'Dina', 'Jenny', 'Sofia', 'Olga', 'Mark', 'Laura']
```

Η συνάρτηση `pop([i])` διαγράφει και επιστρέφει το στοιχείο από τη θέση `i`. Οι τετραγωνικές παρανθέσεις γύρω από τον δείκτη `i` δηλώνουν ότι το όρισμα της συνάρτησης `pop` είναι προαιρετικό. Στην περίπτωση που η συνάρτηση αυτή καλείται χωρίς όρισμα, διαγράφει και επιστρέφει το τελευταίο στοιχείο της λίστας:

```
>>> print names.pop()
Laura
>>> names
['John', 'Dina', 'Jenny', 'Sofia', 'Olga', 'Mark']
>>> gone = names.pop(2)
>>> print gone, 'has left the company'
Jenny has left the company
>>> names
['John', 'Dina', 'Sofia', 'Olga', 'Mark']
```

Παρατηρήστε ότι μετά την εντολή `names.pop(2)` το στοιχείο `'Jenny'` αποθηκεύτηκε στην μεταβλητή `gone`.

Άλλες συναρτήσεις: Η συνάρτηση `reverse()` αντιστρέφει τη σειρά των στοιχείων σε μια λίστα και η συνάρτηση `sort()` ταξινομεί τα στοιχεία μιας λίστας κατ΄αύξουσα σειρά. Τόσο η αντιστροφή όσο και η ταξινόμηση γίνονται επιτόπου. Η ταξινόμηση συμβολοσειρών γίνεται λεξικογραφικά.

```
>>>  names.reverse()
>>> names
['Mark', 'Olga', 'Sofia', 'Dina', 'John']
>>> names.sort()
>>> names
['Dina', 'John', 'Mark', 'Olga', 'Sofia']
>>> names.sort(reverse=True)
>>> names
['Sofia', 'Olga', 'Mark', 'John', 'Dina']
```

Οι συναρτήσεις `min()` και `max()` επιστρέφουν το ελάχιστο, αντίστοιχα, μέγιστο στοιχείο μιας λίστας:

```
>>> grades = [4, 2.1, 5, 6.7, 9.2, 5.4, 5, 7.1, 5]
>>> print 'Min grade is', min(grades)
Min grade is 2.1
>>> print 'Max grade is', max(grades)
Max grade is 9.2
>>> max(names)
'Sofia'
>>> min(names)
'Dina'
```

Η συνάρτηση `index(x)` επιστρέφει τη θέση του πρωτου στοιχείου στη λίστα με τιμή `x`, αν υπάρχει τέτοιο στοιχείο. Η συνάρτηση `count(x)` επιστρέφει τον αριθμό εμφανίσεων του στοιχείου `x` στη λίστα. Τέλος οι εντολές `x in list` και `x not in list` επιστρέφουν True, αντίστοιχα False αν υπάρχει, αντίστοιχα, δεν υπάρχει, στοιχείο στη λίστα `list` με τιμή `x`.

```
>>> grades.index(5)
2
>>> grades.count(5)
3
>>> 2.1 in grades
True
>>> 2.3 in grades
False
>>> 2.3 not in grades
True
```

Σημειώνουμε ακόμα ότι μπορούμε να ορίσουμε μια κενή λίστα χρησιμοποιώντας το συμβολισμό:

```
>>> empty = [None]
>>> empty.append('York')
>>> empty
[None, 'York']
>>> empty.insert(0, 'New')
>>> empty
['New', 'York']
```

ενώ αν `list` είναι μια λίστα και `n` είναι ένας ακέραιος τότε `list*n` ή `n*list` είναι μια λίστα η οποία αποτελείται από `n` αντίγραφα της λίστας `list`:

```
>>> list = [1, 2]
>>> list*3
[1, 2, 1, 2, 1, 2]
```