Κεφάλαιο 5ο
Δομές Δεδομένων Ι

Εισαγωγή

Μία **δομή δεδομένων** (**data structure**) είναι σύνολο δεδομένων (τιμών) μαζί με ένα σύνολο επιτρεπτών λειτουργιών (πράξεων) επί αυτών. Οι πιο βασικές λειτουργίες είναι οι ακόλουθες: προσπέλαση, εισαγωγή, διαγραφή, αναζήτηση, ταξινόμηση, συγχώνευση και διαχωρισμός. Τα δεδομένα που χειρίζονται τα διάφορα προγράμματα που αναπτύσσουμε θα πρέπει να είναι οργανωμένα σε δομές δεδομένων. Ουσιαστικά ένα πρόγραμμα είναι ένα σύνολο αλγορίθμων και δομών δεδομένων.

Η Python διαθέτει αρκετές δομές δεδομένων και παρέχει επαρκή υποστήριξη αυτών. Οι συμβολοσειρές που είδαμε στο προηγούμενο κεφάλαιο μπορούν να θεωρηθούν δομές δεδομένων που αποθηκεύουν αυστηρά μόνο χαρακτήρες. Άλλες χρήσιμες δομές δεδομένων που μπορούν να αποθηκεύουν όλων των ειδών δεδομένα είναι οι **λίστες**, οι **πλειάδες** και τα **λεξικά**.
Η λίστα αποτελεί τη βασική δομή δεδομένων της Python, ενώ η δομή του λεξικού χρησιμεύει, όταν θέλουμε να κάνουμε γρήγορη αναζήτηση και ανάκληση πληροφορίας σχετική με κάποια συμβολοσειρά. Μια άλλη ενσωματωμένη δομή είναι η **πλειάδα** (tuple), η οποία μοιάζει με μια λίστα, αλλά είναι δομή που δεν μπορεί να τροποποιηθεί.

5.1 Στατικές και Δυναμικές Δομές Δεδομένων

Οι δομές δεδομένων, γενικά, χωρίζονται σε δύο βασικές κατηγορίες τις **στατικές** και τις **δυναμικές**, ανάλογα με τη φάση της ανάπτυξης του προγράμματος στην οποία καθορίζεται το μέγεθος της δομής.
Το μέγεθος των **στατικών** δομών παραμένει σταθερό κατά την εκτέλεση του προγράμματος, αφού δεν μπορούμε να αφαιρέσουμε ούτε να προσθέσουμε αντικείμενα στις δομές αυτές. Στην Python στατικές δομές δεδομένων είναι οι πλειάδες (tuples).
Από την άλλη οι **δυναμικές** δομές δεδομένων στηρίζονται σε μια λειτουργία που λέγεται **δυναμική εκχώρηση μνήμης** Κατά τη δυναμική εκχώρηση μνήμης, το πρόγραμμα μπορεί να ζητήσει από το λειτουργικό σύστημα όση μνήμη απαιτείται για τη δημιουργία της δομής δεδομένων, κατά την εκτέλεση του προγράμματος. Οι δυναμικές δομές δεδομένων μπορούν να μεταβάλλουν το μέγεθός τους, προσθέτοντας ή αφαιρώντας αντικείμενα. Η πιο γνωστή δυναμική δομή δεδομένων είναι η λίστα (list), η οποία είναι και η βασική δομή δεδομένων της Python. Με βασικό δομικό λίθο τη λίστα μπορούμε να υλοποιήσουμε όποια σύνθετη δομή δεδομένων θέλουμε, όπως η στοίβα, η ουρά, το δέντρο κ.λπ. Ουσιαστικά, η λίστα της Python δεν είναι τίποτα παραπάνω από ένας δυναμικός πίνακας, δηλαδή ένας πίνακας του οποίου το μέγεθος μπορεί να αυξομειώνεται κατά την εκτέλεση του προγράμματος. Ενδιαφέρον παρουσιάζει ο τύπος της συμβολοσειράς (str) ο οποίος επιτρέπει τη δυναμική δέσμευση μνήμης, αλλά όχι τη μεταβολή του μεγέθους της.

5.2 Συμβολοσειρές (str)

Μια **συμβολοσειρά** (**string**) είναι μια ακολουθία από χαρακτήρες (αριθμητικά ψηφία, γράμματα, σύμβολα) και ανήκει στον τύπο **str**. Μπορούμε να ορίσουμε συμβολοσειρές με μονά, διπλά ή και τριπλά εισαγωγικά.

Σε αντίθεση με τους τύπους int και float, ο τύπος str αποτελείται από μικρότερα κομμάτια, τους χαρακτήρες, γι’ αυτό και λέγεται **σύνθετος**.

Δήλωση :
Η **δήλωση** μίας συμβολοσειρά γίνεται ως μεταβλητή τοποθετώντας δεξιά ένα κείμενο σε μονά η διπλά εισαγωγικά.

συμβολοσειρά = **'**κείμενο**'**

Προσπέλαση με δείκτες

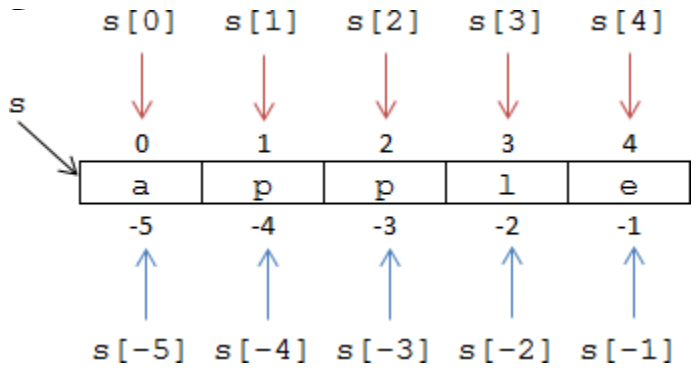
Κατά την εργασία μας με συμβολοσειρές απαιτείται πολλές φορές να προσπελάσουμε ξεχωριστά τους χαρακτήρες από τους οποίους αποτελούνται. Για να γίνει αυτό, χρησιμοποιούμε **δείκτες**. Γενικά ένας δείκτης (index) αναφέρεται σε μέλος ενός διατεταγμένου συνόλου, στη δική μας περίπτωση του συνόλου χαρακτήρων μιας συμβολοσειράς. Δείκτης μπορεί να είναι κάθε ακέραιος αριθμός ή έκφραση.

Η **προσπέλαση** ενός χαρακτήρα μιας συμβολοσειράς γίνεται με την εντολής εκχώρησης

χαρακτήρας = συμβολοσειρά **[δείκτης]**

Βλέπουμε ότι, για να προσπελάσουμε έναν χαρακτήρα μιας συμβολοσειράς, χρησιμοποιούμε το όνομα της συμβολοσειράς ακολουθούμενο από ένα ζευγάρι αγκυλών **[ ]** που περιέχουν έναν δείκτη. Ο δείκτης υποδεικνύει τον χαρακτήρα που θέλουμε να προσπελάσουμε κάθε φορά. Η αρίθμηση των δεικτών ξεκινάει από το μηδέν. Υποθέστε ότι ένας δείκτης μετράει την «απόσταση» από τον πρώτο χαρακτήρα μιας συμβολοσειράς, όπως ακριβώς και ένας χάρακας ξεκινάει από το μηδέν. Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε και αρνητικούς δείκτες, οι οποίοι μετράνε από το τέλος της συμβολοσειράς προς την αρχή. Ο δείκτης -1 δίνει τον τελευταίο χαρακτήρα μιας συμβολοσειράς. Για καλύτερη κατανόηση ας δούμε το παραπάνω παράδειγμα με μορφή διαγράμματος:

π.χ. για τη συμβολοσειρά s = 'apple'



Φέτα συμβολοσειράς

Μπορούμε να πάρουμε ένα τμήμα της συμβολοσειράς με χρήση του τελεστή διαμέρισης (slice operator) ":". Όταν γράφουμε s [**αρχή : τέλος**], επιστρέφεται το μέρος της συμβολοσειράς που ξεκινάει από το χαρακτήρα στη θέση **αρχή** μέχρι τη θέση **τέλος**, χωρίς να περιλαμβάνει το χαρακτήρα της θέσης **τέλος**. Στη βιβλιογραφία το τμήμα αυτό της συμβολοσειράς αναφέρεται και ως **φέτα** (slice).
Αν αφήσουμε κενό τον πρώτο δείκτη μιας φέτας, η Python υποθέτει ότι εννοούμε τον δείκτη 0 και, αν αφήσουμε κενό τον δεύτερο δείκτη, η Python υποθέτει ότι εννοούμε όλο το υπόλοιπο τμήμα μέχρι το τέλος της συμβολοσειράς.

π.χ. για τη συμβολοσειρά s = 'apple'
t = s [1 : 3]

οπότε δημιουργήσαμε μία νέα συμβολοσειρά t που περιέχει το κείμενο 'pp'

Πέρασμα συμβολοσειράς με βρόγχο while και for

Η Python μάς προσφέρει την ενσωματωμένη συνάρτηση **len**, η οποία επιστρέφει το πλήθος των χαρακτήρων μιας συμβολοσειράς ή αλλιώς το μήκος μιας συμβολοσειράς.

```
index = 0
while index < len (s) :
    print s [index]
    index = index + 1
```

ή πιο απλά με τη σύνταξη :

```
for ch in s :
   print ch
```

Σε κάθε πέρασμα του βρόγχου for ο επόμενος χαρακτήρας της συμβολοσειράς string εκχωρείται στη μεταβλητή ch. Ο βρόγχος συνεχίζει μέχρι να εξαντληθούν οι χαρακτήρες.

Σύγκριση συμβολοσειρών

Μπορούμε να χρησιμοποιήσουμε τους τελεστές σύγκρισης, για να συγκρίνουμε συμβολοσειρές. Η σύγκριση βασίζεται στη διάταξη των χαρακτήρων στο σχήμα κωδικοποίησης του υπολογιστή μας. Τα κεφαλαία προηγούνται των πεζών γραμμάτων και τα Λατινικά προηγούνται των Ελληνικών γραμμάτων. Μια σημαντική χρήση της σύγκρισης συμβολοσειρών είναι η αλφαβητική ταξινόμηση ονομάτων.

Ο τελεστής in

Ένας σημαντικός τελεστής που θα συναντήσουμε και αργότερα στις λίστες είναι ο τελεστής **in**, ο οποίος ελέγχει αν ένα αντικείμενο ανήκει σε ένα σύνολο αντικειμένων.
Ο τελεστής in είναι ένας λογικός τελεστής που εφαρμόζεται ανάμεσα σε ένα συνδυασμό χαρακτήρων ch και μία συμβολοσειρά s και ελέγχει αν ο ένας εμφανίζεται μέσα στην άλλη.
Η πρόταση ‘ch’ **in** s επιστρέφει true ή false και με αυτό τον τρόπο μπορούμε να το χρησιμοποιήσουμε σε μία συνθήκη.

Οι συμβολοσειρές είναι αμετάβλητες

Οι συμβολοσειρές είναι αντικείμενα αμετάβλητα (immutable) , δηλαδή, δεν είναι τροποποιήσιμα και έτσι δεν μπορούμε να αλλάξουμε μέρος της συμβολοσειράς.
Αν δοκιμάσουμε να χρησιμοποιήσουμε την εντολή s [index] = ‘c’, στην αριστερή πλευρά μιας εκχώρησης και με στόχο να αλλάξουμε έναν χαρακτήρα σε μια συμβολοσειρά, η Python δεν θα μας αφήσει, και θα εμφανιστεί μήνυμα λάθους.
Το μόνο που μπορούμε να κάνουμε είναι να δημιουργήσουμε μια νέα συμβολοσειρά που θα είναι παραλλαγή της αρχικής.
π.χ. για τη συμβολοσειρά s = ‘apple’

s = s + ‘s’

οπότε δημιουργήσαμε μία νέα συμβολοσειρά s που περιέχει το κείμενο ‘apples’

Μέθοδοι για συμβολοσειρές

Οι εργασίες που καλούμαστε να κάνουμε σε συμβολοσειρές είναι αρκετές, όπως εργασίες ελέγχου (π.χ. αν μία συμβολοσειρά περιέχει μόνο γράμματα ή ψηφία), εργασίες αναζήτησης (π.χ. αναζήτηση χαρακτήρα σε μια συμβολοσειρά), εργασίες μορφοποίησης, κ.λπ. Η Python μάς διευκολύνει με αυτές τις εργασίες προσφέροντάς μας ένα σύνολο από χρήσιμες μεθόδους. Οι μέθοδοι είναι όμοιες με τις συναρτήσεις (παίρνουν ορίσματα και επιστρέφουν κάποια τιμή), αλλά διαφέρουν στον τρόπο με τον οποίο συντάσσονται με τη χρήση του συμβόλου της τελείας “**.**’’ (dot notation).

Για να αναζητήσουμε χαρακτήρες μέσα σε μία συμβολοσειρά, μπορούμε να χρησιμοποιήσουμε τις παρακάτω μεθόδους :
n = s.**find(**ch**)** : επιστρέφει τον πρώτο δείκτη, στον οποίο ξεκινάει συνδυασμός χαρακτήρων ch μέσα στη s, αλλιώς επιστρέφει -1.

n=s.**count(**ch**)** : μπορούμε να μετρήσουμε την εμφάνιση συνδυασμού χαρακτήρων ch μέσα στη s.

n=s.**index(**letter**)** : επιστρέφει τον πρώτο δείκτη που θα εμφανιστεί ο χαρακτήρας letter μέσα στη s.

Για κάνουμε μετατροπές μεταξύ κεφαλαίων και πεζών γραμμάτων, μπορούμε να χρησιμοποιήσουμε τις παρακάτω μεθόδους :

s2 = s.**capitalize ()** : το πρώτο γράμμα της s, το s[0] μετατρέπεται σε κεφαλαίο.

s2 = s.**upper ()** : όλα τα γράμματα της s μετατρέπονται σε κεφαλαία.

s2 = s.**lower ()** : όλα τα γράμματα της s μετατρέπονται σε πεζά.

5.3 Λίστα

Η λίστα στην Python αποτελεί τη βασική δομή δεδομένων της γλώσσας.
Μια λίστα είναι ουσιαστικά μια διατεταγμένη ακολουθία από αντικείμενα τα οποία συνήθως είναι του ίδιου τύπου. Όμως μια λίστα μπορεί να αποτελείται και από αντικείμενα διαφορετικού τύπου.
Μία **λίστα** (**list**) είναι μια διατεταγμένη συλλογή τιμών, οι οποίες αντιστοιχίζονται σε δείκτες. Οι τιμές που είναι μέλη μιας λίστας ονομάζονται **στοιχεία** (**elements**). Τα στοιχεία μιας λίστας δεν χρειάζεται να είναι ίδιου τύπου και ένα στοιχείο σε μία λίστα μπορεί να υπάρχει περισσότερες από μία φορές.
Μία λίστα μέσα σε μία άλλη λίστα ονομάζεται **εμφωλευμένη λίστα** (**nested list**). Επιπρόσθετα, τόσο οι λίστες όσο και οι συμβολοσειρές, που συμπεριφέρονται ως διατεταγμένες συλλογές τιμών, ονομάζονται **ακολουθίες** (**sequences**).

Δήλωση :
Η **δήλωση** μίας μιας δομής δεδομένων λίστας γίνεται ως μεταβλητή τοποθετώντας δεξιά μία σειρά από στοιχεία - τα οποία χωρίζουμε με κόμμα ‘’,’’ - μέσα σε **τετράγωνες αγκύλες []**.

λίστα **= [** στοιχείο0**,** στοιχείο1**,** στοιχείο2**,** στοιχείο3**,** στοιχείο4 **]**

Σημείωση : Η συνάρτηση **list** (s) «σπάει» μια συμβολοσειρά σε χαρακτήρες δημιουργώντας μία λίστα.

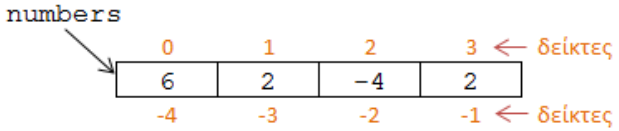
Παράδειγμα εναλλακτικής δημιουργίας μιας αριθμητικής λίστας
numbers = **list** (**range** (1, 11))
Η **print** numbers *θα μας δώσει τη λίστα* [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Προσπέλαση με δείκτες
Για κάθε αντικείμενο που εισάγεται στη λίστα δίνεται ένας αύξων αριθμός, που χρησιμοποιείται για την αναφορά του στο αντικείμενο. Η προσπέλαση στα στοιχεία της λίστας γίνεται όπως και στις συμβολοσειρές, με δείκτες :

μεταβλητή **=** λίστα **[δείκτης]**

Ως δείκτης μπορεί να χρησιμοποιηθεί οποιαδήποτε ακέραια έκφραση Αν προσπαθήσουμε να προσπελάσουμε στοιχείο που δεν υπάρχει, τότε θα εμφανιστεί μήνυμα λάθους.

π.χ. για τη λίστα numbers = **[** 6, 2, -4, 2 **]**



η εντολή **print** numbers [2] τυπώνει το αντικείμενο -4

Μία λίστα που δεν περιέχει στοιχεία ονομάζεται **άδεια λίστα** και συμβολίζεται με **[ ]**

Η λίστα μπορεί και αυτή να θεωρηθεί ως ένα σύνολο από αντικείμενα, οπότε μπορούμε να χρησιμοποιήσουμε τον υπαρξιακό τελεστή **in** και τη συνάρτηση **len**. Η συνάρτηση **len** επιστρέφει το μήκος μιας λίστας (το πλήθος των στοιχείων της). Αν μία λίστα περιέχει άλλη λίστα ως στοιχείο, τότε η εμφωλευμένη λίστα μετράει ως απλό στοιχείο.

Με τον λογικό τελεστή **in** μπορούμε να ελέγξουμε αν μια τιμή ανήκει σε μια λίστα και δουλεύει όπως και στις συμβολοσειρές. Μπορούμε επίσης να χρησιμοποιήσουμε την έκφραση **not in**.

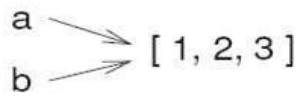
Με τον τελεστή **+** μπορούμε να συνενώσουμε λίστες (αλληλουχία) και με τον τελεστή ***** να επαναλάβουμε μια λίστα.

Ο τελεστής φέτας δουλεύει και με λίστες.

Οι λίστες είναι μετατρέψιμες (mutable)
Σε αντίθεση με τις συμβολοσειρές, οι οποίες δεν μπορούν να τροποποιηθούν (immutable), τα στοιχεία μιας λίστας μπορούν να αλλάξουν. Μπορεί κανείς ανά πάσα στιγμή να προσθέσει, να διαγράψει ή να αλλάξει ένα αντικείμενο σε μία λίστα.

Αντιγραφή μιας λίστας
Εφόσον οι μεταβλητές αναφέρονται σε αντικείμενα, αν εκχωρήσουμε μια μεταβλητή σε μία άλλη, και οι δύο θα αναφέρονται στο ίδιο αντικείμενο :

π.χ.
a = [1, 2, 3]
b = a



Εάν μία λίστα έχει δύο ονόματα, λέμε ότι έχει **ψευδώνυμα**.
Προσοχή : Αλλαγές που γίνονται σε αυτήν την περίπτωση στο ένα ψευδώνυμο επηρεάζουν το άλλο.

Λίστες - κλώνοι
Ο ευκολότερος τρόπος για να κλωνοποιήσουμε μια λίστα είναι με χρήση του τελεστή φέτας ":". Η κλωνοποίηση δημιουργεί καινούρια λίστα, στην οποία μπορούμε να κάνουμε αλλαγές χωρίς να επηρεάζεται η αρχική.

Η εντολή b = a *δημιουργεί απλώς ένα ψευδώνυμο και δείχνει στην ίδια λίστα*
ενώ η εντολή b = a [:] *δημιουργεί μία νέα λίστα ή οποία είναι αντίγραφο της άλλης*

Προσοχή : δεν μπορεί να γίνει το ίδιο με τις συμβολοσειρές. Δηλαδή η εντολή s [:] δε δημιουργεί αντίγραφο της συμβολοσειράς s.

Ένας δεύτερος τρόπος για να δημιουργήσουμε αντίγραφο μιας λίστας είναι έμμεσα με χρήση του τελεστή **+** ο οποίος δημιουργεί μια νέα λίστα, ως άθροισμα δύο ήδη υπαρχουσών λιστών. Για να δημιουργήσουμε αντίγραφο μιας λίστας αρκεί να προσθέσουμε την κενή [ ]:

Η εντολή b = a + [ ] *δημιουργεί μία νέα λίστα ή οποία είναι αντίγραφο της άλλης*

Διαγραφή στοιχείων μιας λίστας
Ο ευκολότερος τρόπος για να διαγράψουμε στοιχεία από μια λίστα είναι να χρησιμοποιήσουμε τον τελεστή **del**. Ο **del** χειρίζεται και αρνητικούς δείκτες και φέτες.
π.χ. a = [1, 2, 3]
το **del a** [2] *σβήσει το στοιχείο* 3

Μέθοδοι για λίστες
Η Python διαθέτει έτοιμες μεθόδους και για τις λίστες.
Για να προσθαφαιρέσουμε στοιχεία μέσα σε μία λίστα, μπορούμε να χρησιμοποιήσουμε τις παρακάτω μεθόδους :

l2 = l.**pop (**n**)** : αφαιρεί και επιστρέφει το στοιχείο x, το οποίο βρίσκεται πριν το n-στο στοιχείο της λίστας l.
Αν καλέσουμε την **pop** χωρίς ορίσματα αφαιρεί το τελευταίο στοιχείο της λίστας l.
l.**append (**x**)** : προσθέτει το στοιχείο x στο τέλος της λίστας l.
l.**insert (**n, x**)** : προσθέτει το στοιχείο x πριν από το n-στο στοιχείο της λίστας l.
l1.**extend (**l2**)** : προσθέτει όλα τα στοιχεία της λίστας l2 στο τέλος της λίστας l1
l.**sort () :** ταξινομεί τα στοιχεία της λίστα l κατά αύξουσα σειρά.

5.4 Επεξεργασία Λιστών

Συνάρτηση δημιουργίας λίστας
Μπορούμε να δημιουργήσουμε μια λίστα με το όνομα array, ορίζοντας την παρακάτω συνάρτηση newList :

```
def newList (size):
   array = [ ]
   for i in range (0,size):
      array.append( 0 )
  return array
```

Η συνάρτηση αυτή παίρνει σαν όρισμα έναν αριθμό και επιστρέφει μια λίστα με αυτό το μέγεθος. Τα στοιχεία της λίστας έχουν όλα την τιμή 0. Αν θέλουμε να μη δώσουμε τιμή στα στοιχεία της λίστας, θα χρησιμοποιήσουμε την τιμή **None**, που η Python διαθέτει γι' αυτό το σκοπό

Στις λίστες στην Python, όπως και στις περισσότερες σύγχρονες γλώσσες προγραμματισμού, η αρίθμηση ξεκινάει από το 0 και όχι από το 1.

Προσπέλαση των στοιχείων μιας λίστας με τη σειρά

Μπορούμε να διατρέξουμε τα στοιχεία μια λίστας με το όνομα array, με τον παρακάτω τρόπο :

```
N = len (array)
for i in range (0, N) :
    print array [i]
```

Την ίδια εργασία μπορούμε να κάνουμε με το ιδίωμα της Python :

```
for item in array :
    print item
```

Δημιουργία διδιάστατου πίνακα με εμφωλευμένες λίστες

Μπορούμε επίσης να κατασκευάσουμε μια λίστα με στοιχεία λίστες. Αυτό μπορεί να μας φανεί χρήσιμο σε προβλήματα τα οποία μοντελοποιούνται με τη χρήση ενός πίνακα δυο διαστάσεων Ένας πίνακας μπορεί να υλοποιηθεί στην Python αν θεωρήσουμε τις γραμμές ως λίστες οι οποίες είναι εμφωλευμένες σε μια μεγάλη κεντρική λίστα.
Για την προσπέλαση του στοιχείο που βρίσκεται στην i-οστή γραμμή και στην j-οστή στήλη, στην Python χρησιμοποιείται ο συμβολισμός a[i][j].

π.χ.
A = [[0, 12, 14], [22, 0, 28], [15, 6, 0]]
Η **print** A [0] *θα μας δώσει* [0, 12, 14]
Η **print** A [0] [2] *θα μας δώσει* 14

Δημιουργούμε έναν πίνακα δύο διαστάσεων με τη χρήση της **append** :

```
for i in range (0,3):
    a.append( [ ] )
    for j in range (0,3):
        a[i].append(0)
```

Χρήσεις
Μια λίστα μπορεί να λειτουργήσει σαν **στοίβα**, αν οι μόνες πράξεις που εφαρμόζουμε σε αυτήν είναι η **append** και η **pop**, δηλαδή η προσθήκη (ώθηση) και αφαίρεση (απώθηση) στοιχείων μόνο από το ένα άκρο. Ομοίως η λίστα μπορεί να λειτουργήσει σαν ουρά αν προσθέτουμε από το ένα άκρο και αφαιρούμε από το άλλο.

Παραδείγματα

```
L = [1, 4, 5, 6]
for number in L :
      print number
```

```
for c in 'hello':
   print(c)
```

```
L = ['Nikos','Mitsos','Anna ']
for onoma in L :
     print onoma
```

```
File =open("words.txt")
for line in File:
         print line
```

Άσκηση
Βρείτε τι κάνει το παρακάτω πρόγραμμα:

```
sqlist=[ ]
for x in range (1,11):
    sqlist.append (x*x)
    print sqlist
```

Απάντηση : δημιουργεί μία λίστα με στοιχεία τα τετράγωνα [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

5.6 Πλειάδες

Μία **πλειάδα** (**tuple**) είναι μια διατεταγμένη ακολουθία τιμών, οι οποίες αντιστοιχίζονται σε δείκτες. Οι τιμές που είναι μέλη μιας πλειάδας ονομάζονται **στοιχεία** (**elements**) και μπορεί να είναι οποιουδήποτε τύπου (αριθμοί, συμβολοσειρές, λίστες, πλειάδες). Οι πλειάδες μοιάζουν με τις λίστες στη χρήση δεικτών, στον τρόπο με τον οποίο διατρέχονται και στη χρήση του τελεστή φέτας.

Δήλωση :
Συντακτικά, μια πλειάδα είναι μια λίστα τιμών που χωρίζονται με κόμμα. Παρόλο που δεν είναι αναγκαίο, είναι σύνηθες να περικλείουμε τις πλειάδες με παρενθέσεις **( )**.

πλειάδα **=** ‘a’, 32, ΄καλημέρα΄, 5
ή καλύτερα
πλειάδα **= (** ‘a’, 32, ΄καλημέρα΄, 5 **)**

Για να δημιουργήσουμε μια άδεια πλειάδα, χρησιμοποιούμε άδειες παρενθέσεις.
t = ()
Για να δημιουργήσουμε πλειάδα με ένα μόνο στοιχείο, πρέπει να προσθέσουμε ένα κόμμα. Χωρίς κόμμα η Python νομίζει ότι πρόκειται για συμβολοσειρά μέσα σε παρενθέσεις.
t = (‘a’,)

Σημείωση :
Η συνάρτηση **tuple** () δέχεται ως όρισμα μια συμβολοσειρά ή μια λίστα και επιστρέφει μια πλειάδα.
t = **tuple** (συμβολοσειρά ή λίστα **)**
Με κενό όρισμα επιστρέφει μια άδεια πλειάδα.

Πρόκειται για ένα σύνθετο τύπο που τον χρησιμοποιούμε όταν θέλουμε να συγκρατήσουμε μαζί πολλαπλά αντικείμενα. Η πλειάδα μοιάζει με μια λίστα.

Οι πράξεις πάνω σε πλειάδες είναι παρόμοιες με τις πράξεις πάνω σε λίστες. Ο τελεστής **[]** επιλέγει ένα στοιχείο από μια πλειάδα. Ο τελεστής ‘’**:**’’ «φέτα» επιλέγει διάστημα τιμών, όπως ακριβώς και στις λίστες. Ο τελεστής **in** ελέγχει εάν μια τιμή ανήκει σε μια πλειάδα. Η συνάρτηση **len** επιστρέφει το μήκος μιας πλειάδας (τον αριθμό των στοιχείων που περιέχει).

Οι πλειάδες είναι αμετάβλητες
Οι πλειάδες, όπως και οι συμβολοσειρές, είναι **αμετάβλητες**. Αν προσπαθήσουμε να αλλάξουμε ένα από τα στοιχεία μιας πλειάδας, θα εμφανιστεί μήνυμα λάθους. Μπορούμε όμως να

αντικαταστήσουμε μια πλειάδα με μία άλλη. Οι πλειάδες αξιοποιούνται συνήθως στις περιπτώσεις όπου πρόκειται να χρησιμοποιηθεί μια ακολουθία τιμών (πλειάδα) που δεν πρόκειται να αλλάξει.

παραδείγματα:
rec = 'a', 'b', 120, 'r'
Η **print rec** *θα μας δώσει* ('a', 'b', 120, 'r')
Η **rec** [1:3] *θα μας δώσει* ('b', 120)
Η **rec** [2] *θα μας δώσει* 120
Η **len(rec)** *θα μας δώσει* 4

rec = ('apple' ,) + **rec** [2:3]
Τώρα η **print rec** *θα μας δώσει* ('apple', 120)
το **del rec** [1] *σβήσει το στοιχείο* 120

Μέθοδοι για πλειάδες
Όπως γίνεται και με τις συμβολοσειρές, η μέθοδος **count** (στοιχείο) επιστρέφει τον αριθμό των εμφανίσεων μιας τιμής σε μια πλειάδα. Η μέθοδος **index** (στοιχείο) επιστρέφει τον πρώτο δείκτη στον οποίο αντιστοιχεί μια τιμή. Αν δεν υπάρχει τιμή, εμφανίζεται μήνυμα λάθους.

Χρήσεις
Συνήθως οι πλειάδες χρησιμοποιούνται για την αναπαράσταση μιας σειράς χαρακτηριστικών/ιδιοτήτων μιας οντότητας, εφόσον τα χαρακτηριστικά αυτά δεν μεταβάλλονται.

Για παράδειγμα, θέλουμε να έχουμε σε μια λίστα τα στοιχεία των υπαλλήλων μιας επιχείρησης, όπως όνομα, επώνυμο, βαθμός, τηλέφωνο, τμήμα, μισθός κ.λπ. Αυτό θα το υλοποιήσουμε με μια λίστα από πλειάδες, όπου κάθε πλειάδα θα αντιστοιχεί σε έναν εργαζόμενο. Μια σημαντική εφαρμογή των πλειάδων είναι ότι μπορούμε να ορίσουμε συναρτήσεις με ορίσματα μεταβλητού μεγέθους.

Μια άλλη γνωστή εφαρμογή τους είναι η **αντιμετάθεση των τιμών δύο μεταβλητών, χωρίς τη χρήση βοηθητικής μεταβλητής,** με την παρακάτω εντολή:
a , b = b, a

5.7 Λεξικά

Το **λεξικό** (**dictionary**) είναι μια δομή δεδομένων για αποθήκευση ζευγαριών τιμών της μορφής *κλειδί : τιμή* (*key : value*). Πρόκειται για έναν σύνθετο τύπο. Κάθε κλειδί αντιστοιχίζεται σε μια τιμή και είναι μοναδικό σε ένα λεξικό. Μπορούμε να χρησιμοποιούμε μόνο αμετάβλητα αντικείμενα (όπως ακέραιους αριθμούς, συμβολοσειρές, πλειάδες) για κλειδιά ενός λεξικού, αλλά μπορούμε να έχουμε είτε αμετάβλητα ή μετατρέψιμα αντικείμενα για τις τιμές του. Τα λεξικά είναι **μετατρέψιμα**, και μπορούμε εύκολα να προσθέσουμε και να διαγράψουμε στοιχεία. Επίσης, ένα λεξικό αποτελεί μια μη διατεταγμένη συλλογή από ζεύγη κλειδιών-τιμών, τα οποία δεν ταξινομούνται με κανένα τρόπο (απροσδιόριστη σειρά). Δεν υπάρχει η έννοια της θέσης δείκτη και έτσι σε ένα λεξικό δεν μπορούμε να κάνουμε πέρασμα ή να χρησιμοποιήσουμε φέτες.

Δήλωση :
Μπορούμε να ορίσουμε ένα λεξικό ως μία σειρά από ζεύγη κλειδιών : τιμών που χωρίζονται με κόμμα. Τα ξεχωριστά στοιχεία της που είναι τα ζεύγη τα περικλείουμε με αγκύλες **{ }** :
Λεξικό = {'one':1, 'two':2, 'three':3}

Η **print** λεξικό *θα μας δώσει* {'one':1, 'two':2, 'three':3}

Το λεξικό είναι μια εξαιρετικά χρήσιμη ενσωματωμένη (built-in) δομή της Python.
Φανταστείτε το σαν έναν τηλεφωνικό κατάλογο ο οποίος μας δίνει τη δυνατότητα να βρούμε πολύ γρήγορα τα στοιχεία κάποιου, μόνο από το όνομά του. Προγραμματιστικά, φανταστείτε το σαν μια λίστα που έχει ως δείκτες αλφαριθμητικά και όχι ακέραιους αριθμούς.

Όπως σε έναν πίνακα, η θέση κάθε στοιχείου είναι μοναδική και δεν μπορεί να περιέχει ταυτόχρονα δυο τιμές, έτσι και στο λεξικό, η λέξη-κλειδί, που χρησιμοποιούμε μέσα στις αγκύλες, έχει μοναδική τιμή και εμφανίζεται το πολύ μια φορά. Είναι το λεγόμενο **κλειδί**.
Έτσι ένα λεξικό είναι ουσιαστικά ένα σύνολο ζευγών κλειδιών-τιμών, όπου κάθε κλειδί δεν εμφανίζεται δεύτερη φορά.
Σημείωση : Μπορούμε να δημιουργήσουμε ένα λεξικό και με τη χρήσης της ενσωματωμένης συνάρτησης **dict** ().
π.χ. d = **dict** (one=1, two=2, three=3)

Θα πρέπει να σημειωθεί ότι κάθε κλειδί έχει μοναδική τιμή, ενώ για να προσθέσουμε ένα νέο ζεύγος σε ένα λεξικό αυτό γίνεται με την εκχώρηση **Λεξικό [κλειδί] = τιμή** οπότε και δημιουργείται το νέο ζεύγος {κλειδί : τιμή}. Αν θέλουμε να αντιστοιχήσουμε σε ένα κλειδί περισσότερες από μια τιμές, τότε η τιμή του κλειδιού είναι μια λίστα από τις τιμές αυτές.

Ο τελεστής **del** αφαιρεί ένα ζευγάρι κλειδιού-τιμής από ένα λεξικό.
π.χ. dictionary = {'one':1, 'two':2, 'three':3}
το **del** dictionary ['two'] *σβήσει το ζεύγος* 'two':2

Η συνάρτηση **len** χρησιμοποιείται και στα λεξικά και επιστρέφει το πλήθος των ζευγαριών κλειδιών-τιμών. Επίσης, ο τελεστής **in** ελέγχει το αν υπάρχει ένα κλειδί (και όχι μια τιμή) σε ένα λεξικό.

Μέθοδοι για λεξικά
Μπορούμε να επικαλεστούμε πάνω σε ένα λεξικό τις μεθόδους **keys**, **values**, **items** για να επιστρέψουμε μια **εικόνα** (**view**) των κλειδιών, των τιμών και των ζευγαριών κλειδιών-τιμών αντίστοιχα.

π.χ.
```
dictionary = {'one':1, 'two':2, 'three':3}
keys_view =  dictionary.keys()      #  θα εκχωρήσει τη λίστα  ['one', 'two', 'three']
values_view = dictionary.values()  #  θα εκχωρήσει τη λίστα  [1, 2, 3]
items_view = dictionary.items()    # θα εκχωρήσει τη λίστα  [ ('one', 1), ('two', 2), ('three', 3) ]

for key in keys_view:
   print key,          # θα τυπωθούν τα κλειδιά three two one
for value in values_view:
   print value,        # θα τυπωθούν οι τιμές 3 2 1
for key, value in items_view:
   print (key, value),    # θα τυπωθούν οι πλειάδες κλειδιών τιμών ('three', 3)  ('two', 2) ('one', 1)
```
Με τη βοήθεια της συνάρτησης **list** μπορούμε να αποθηκεύσουμε τα κλειδιά, τις τιμές και τα

ζευγάρια κλειδιών-τιμών ενός λεξικού σε αντίστοιχες λίστες

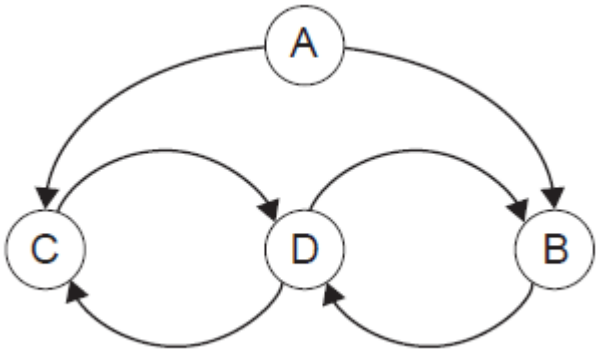
```
keys_view = list (dictionary.keys())        #  θα μας δώσει τη λίστα  ['one', 'two', 'three']
values_view = list (dictionary.values())    #  θα μας δώσει τη λίστα  [1, 2, 3]
items_view = list (dictionary.items())      #  θα μας δώσει τη λίστα των πλειάδων
                                                 [ ('one', 1), ('two', 2), ('three', 3) ]
```

παραδείγματα

```
dictionary = { 'A':2, 'B':4, 'Ω':8, 'M':16 }
list (dictionary.keys() )  θα μας δώσει τη λίστα          ['A', 'B', 'Ω', 'M']
dictionary ['Ω']           θα μας δώσει την τιμή          8
len (dictionary )          θα μας δώσει την τιμή          4
del dictionary ['Ω']
del dictionary ['A']
print dictionary           θα μας δώσει το λεξικό         {'B':4, 'M':16}
dictionary ['Λ'] = 32
print dictionary           θα μας δώσει το λεξικό         {'B':4, 'Λ':32, 'M':16}
'Λ' in dictionary          θα μας δώσει τη λογική τιμή    True
```

Χρήσεις

Ο συνδυασμός λεξικού και πλειάδας ή λίστας, έχει πολλές εφαρμογές όπως για παράδειγμα στην αναπαράσταση **γράφων**. Η δομή του γραφήματος (Λίστα γειτνίασης) στην Python υλοποιείται με ένα λεξικό, με κλειδιά τις κορυφές του γράφου και τιμές τη λίστα ακμών κάθε κλειδιού, δηλαδή τη λίστα των γειτονικών κορυφών.



graph = { 'A': ['B', 'C'], 'B': ['D'], 'C': ['D'], 'D': ['B', 'C'] }

5.5 Σύνολα

Τα **σύνολα** αποτελούν μία από τις ενσωματωμένες δομές δεδομένων της Python και υλοποιούν τα ομώνυμα μαθηματικά αντικείμενα. Ένα σύνολο είναι μια ομάδα από ***μη*** *διατεταγμένα αντικείμενα*, με κάθε αντικείμενο να εμφανίζεται μία φορά. Τα σύνολα υποστηρίζουν τις γνωστές *πράξεις συνόλων*, δηλαδή την ένωση, την τομή και το συμπλήρωμα.

Δήλωση :

```
Σύνολο = set ([ στοιχείο1, στοιχείο2, στοιχείο3, στοιχείο4, στοιχείο5 ])
```

Για τη δημιουργία κενού συνόλου :

mySet = **set ()**

π.χ.
myset = set([1 ,2, 3, 1, 2, 3, 1, 2, 3, 8])
print myset *θα μας δώσει το σύνολο* set([8, 1, 2, 3])
myset.i**ntersection** (set([1, 2, 90])) *θα μας σώσει την τομή* set([1, 2])