Κεφάλαιο 8ο
Δομές Δεδομένων ΙΙ

Εισαγωγή

Μια **δομή δεδομένων** μπορεί να οριστεί ως ένα σχήμα οργάνωσης σχετικών μεταξύ τους στοιχείων δεδομένων. Η επιλογή της κατάλληλης δομής δεδομένων παίζει σημαντικό ρόλο στην ανάπτυξη του αλγορίθμου, για την επίλυση ενός προβλήματος.
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.

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.

8.1 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

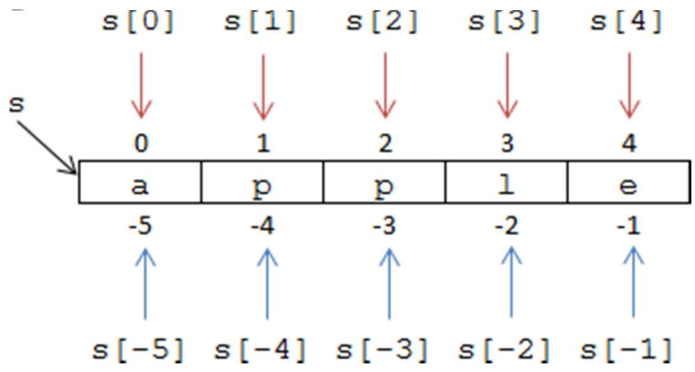
Κατά την εργασία μας με συμβολοσειρές απαιτείται πολλές φορές να προσπελάσουμε ξεχωριστά τους χαρακτήρες από τους οποίους αποτελούνται. Για να γίνει αυτό, χρησιμοποιούμε **δείκτες**. Γενικά ένας δείκτης (index) αναφέρεται σε μέλος ενός διατεταγμένου συνόλου, στη δική μας περίπτωση του συνόλου χαρακτήρων 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 δημιουργείτε μία νέα συμβολοσειρά t που περιέχει το κείμενο 'pp'

Συνάρτηση len (str)

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

Ο τελεστής + όταν εφαρμόζεται σε αντικείμενα τύπου string, έχει σαν αποτέλεσμα τη συνένωσή τους σε μια συμβολοσειρά
Με τη συνάρτηση str μπορούμε να μετατρέψουμε μια αριθμητική τιμή σε συμβολοσειρά
Με τη συνάρτηση int μπορούμε να μετατρέψουμε ένα αλφαριθμητικό που περιέχει νούμερα στον ακέραιο αριθμό που αναπαριστά.

Πέρασμα συμβολοσειράς με βρόγχο while και for

```
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.

Παραδείγματα

1. Σάρωση των χαρακτήρων μιας συμβολοσειράς

Μια συμβολοσειρά είναι μια ακολουθία (sequence) από αντικείμενα, τα οποία, στην προκειμένη περίπτωση, είναι χαρακτήρες. Αν θέλουμε να σαρώσουμε τα αντικείμενα αυτά ένα-ένα, μπορούμε να το κάνουμε με μια εντολή επανάληψης for. Το παρακάτω τμήμα κώδικα δημιουργεί μια νέα συμβολοσειρά η οποία είναι όμοια με την αρχική, αλλά χωρίς τα κενά ανάμεσα στις λέξεις:

```
def trimSpaces (sentence):
     result = ""
     for char in sentence :
          if char != " " :
               result += char
      return result
```

```
phrase = "Houston we have a problem"
trimSpaces (phrase)  # θα εμφανίσει το κείμενο 'Houstonwehaveaproblem'
```

2. Καταμέτρηση φωνηέντων μιας φράσης

Η παρακάτω συνάρτηση υπολογίζει πόσα φωνήεντα έχει η λέξη word.

```
def count_vowels (word):
       vowels = "AEIOUaeiou"
       count = 0
       for letter in word :
            if letter in vowels:
                 count += 1
      return count
```

Παραπάνω κάνουμε χρήση του ιδιώματος **for…in** για να επεξεργαστούμε τους χαρακτήρες της συμβολοσειράς, έναν κάθε φορά.

Σημείωση: Παρατηρήστε το διαφορετικό τρόπο με τον οποίο χρησιμοποιείται ο τελεστής **in**, στη μία περίπτωση για τον καθορισμό του εύρους της επανάληψης και στην άλλη για τον έλεγχο ύπαρξης του γράμματος letter στη λέξη vowels.

Σύγκριση συμβολοσειρών

Οι γνωστοί *συγκριτικοί τελεστές* (<, <=, >, >=, ==, !=) ισχύουν και στις συμβολοσειρές, η λειτουργία των οποίων βασίζεται στη λεξικογραφική διάταξη των χαρακτήρων.
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.

Έλεγχος ύπαρξης - 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 μη μεταβαλλόμενες δομές

Οι συμβολοσειρές είναι αντικείμενα μη μεταβαλλόμενα (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’      # δημιουργείτε 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.

8.2 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 (αλλά όχι απαραίτητα). 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 **]**

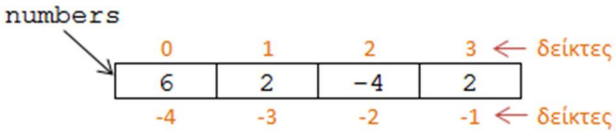
Σημείωση : Η συνάρτηση **list** (s) «σπάει» μια συμβολοσειρά σε χαρακτήρες δημιουργώντας μία λίστα.

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

Δήλωση μιας λίστας χωρίς στοιχεία
Μία λίστα που δεν περιέχει στοιχεία ονομάζεται **άδεια λίστα** και συμβολίζεται με **[ ]**
Για να δηλώσουμε μία λίστα στην οποία θα εισαγάγουμε στοιχεία στη συνέχεια του προγράμματος μας στην αρχή του προγράμματος μας δίνουμε την εντολή :
Λίστα = [ ]

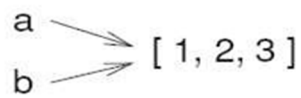
Μπορούμε να δημιουργήσουμε νέα λίστα συνενώνοντας δύο λίστες
Με τον τελεστή **+** μπορούμε να συνενώσουμε λίστες (αλληλουχία) λίστα και με τον τελεστή ***** να επαναλάβουμε μια λίστα.

Οι λίστες - όπως και οι συμβολοσειρές - ανήκουν και σε μια πιο γενική κατηγορία δομών, τις *ακολουθιακές δομές* (sequences) της Python, οπότε ισχύουν οι ίδιοι τελεστές.
Μπορούμε λοιπόν να χρησιμοποιήσουμε τον υπαρξιακό τελεστή **in** και τη συνάρτηση **len (list)**. 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.

Μια ακόμη χρήσιμη συνάρτηση εκτός από τη len που χρησιμοποιείται στις λίστες είναι η **list (string)** η οποία επιστρέφει μια λίστα με στοιχεία τους χαρακτήρες της συμβολοσειράς string. Η συνάρτηση αυτή μπορεί να μετατρέψει και άλλα είδη δομών σε λίστα, όπως είναι οι πλειάδες και τα λεξικά
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**.

Οι λίστες είναι μετατρέψιμες (mutable)
Οι λίστες είναι μία δυναμική δομή και 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.

Ο τελεστής διαμέρισης

Ο τελεστής φέτας δουλεύει και με λίστες.

Υπενθύμιση : Ο τελεστής διαμέρισης μας επιστρέφει ένα τμήμα μιας συμβολοσειράς ή μιας λίστας. Η έκφραση word [αρχή : τέλος] μας επιστρέφει το τμήμα της συμβολοσειράς ή της λίστας από το στοιχείο word [αρχή] μέχρι και το στοιχείο word [τέλος-1]. Μπορούμε να παραλείψουμε την αρχή ή το τέλος αν θέλουμε ένα τμήμα από την αρχή ή το τέλος της λίστας/συμβολοσειράς αντίστοιχα.

Προσοχή : Στις λίστες η χρήση του τελεστή διαμέρισης word [:] - σε αντίθεση με ότι συμβαίνει στις τις συμβολοσειρές - δημιουργεί ένα αντίγραφο της λίστας πχ list = word [:]. Αντίθετα η εντολή list = word κάνει τη λίστα list να δείχνει στην λίστα word, οπότε όποια αλλαγή γίνει στη μία από αυτές θα επηρεάσει και την άλλη.

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*

Μπορούμε να δημιουργήσουμε νέα λίστα λοιπόν :
α) συνενώνοντας δύο υπάρχουσες λίστες με τον τελεστή **+**
β) επαναλαμβάνοντας τα στοιχεία μιας υπάρχουσας λίστας με τον τελεστή *****

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

Προσθήκη στοιχείου σε μια Λίστα

Αν θέλουμε να προσθέσουμε ένα στοιχείο
α) στο τέλος μιας λίστας, γράφουμε: Λίστα = Λίστα + [στοιχείο]
β) ενώ στην αρχή της λίστας : Λίστα = [στοιχείο] + Λίστα

Στην πραγματικότητα όμως οι παραπάνω εντολές δεν προσθέτουν το στοιχείο στην ήδη υπάρχουσα λίστα αλλά δημιουργούν μια νέα λίστα κάθε φορά. Η λειτουργία αυτή έχει σημαντικό υπολογιστικό κόστος.
Για αυτό αν θέλουμε να προσθέσουμε ένα στοιχείο στο τέλος της λίστας προτιμούμε τον τελεστή **+=** , όπως φαίνεται παρακάτω:
Λίστα += [στοιχείο]
ή προτιμότερο καταφεύγουμε στη χρήση κάποιας μεθόδου όπως η Λίστα.append (στοιχείο)

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 (**index**)** : αφαιρεί και επιστρέφει το στοιχείο x, το οποίο βρίσκεται στη θέση index της λίστας l. Αν καλέσουμε την **pop** χωρίς όρισμα αφαιρεί το τελευταίο στοιχείο της λίστας l.

l.**append (**x**)** : προσθέτει το στοιχείο x στο τέλος της λίστας l.

l.**insert (**index, x**)** : προσθέτει το στοιχείο x στη θέση index της λίστας l μετακινώντας όλα τα στοιχεία από τη θέση index και μετά, κατά μία θέση δεξιά.
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.

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]
```

Υπενθύμιση :
Η συνάρτηση **range** (αρχή, τέλος, βήμα) επιστρέφει, αν δώσουμε την αρχική, την τελική τιμή και το βήμα, μια λίστα από αριθμούς. Συγκεκριμένα επιστρέφει μια λίστα αριθμών ξεκινώντας με τον αριθμό ‘αρχή’ μέχρι τον αριθμό ‘τέλος’ με βήμα τον αριθμό ‘βήμα’. Ο αριθμός ‘τέλος’ δεν συμπεριλαμβάνεται στη λίστα.

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

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

Διάσχιση Λίστας

Μπορούμε να επεξεργαστούμε τα στοιχεία μιας λίστας, ένα κάθε φορά, κάνοντας χρήση του παρακάτω ιδιώματος της δομής επανάληψης for:

```
for item in List :
    <Εντολές Επεξεργασίας του αντικειμένου item>
```

Παραδείγματα

1. Δημιουργία και εμφάνιση στοιχείων λίστας

Οι παρακάτω εντολές αρχικά κατασκευάζουν μια λίστα η οποία περιέχει όλους τους αριθμούς από το 1 έως και το 6 και στη συνέχεια εμφανίζουν κάθε αριθμό σε διαφορετική γραμμή.

```
L = [1, 2, 3, 4, 5, 6]
for number in L :
    print number
```

2. Μέσος όρων των στοιχείων μιας λίστας

Για να υπολογίσουμε το μέσο όρο των στοιχείων μιας λίστας, πρώτα χρειάζεται να υπολογίσουμε το άθροισμα των στοιχείων, χρησιμοποιώντας μια μεταβλητή στην οποία προσθέτουμε, κάθε φορά, το επόμενο στοιχείο της λίστας:

```
sum = 0.0                       # το sum είναι πραγματικός (float)
for number in L :
    sum = sum + number
average = sum / len( L )        # δεν θα γίνει ακέραια διαίρεση
print average
```

3. Μέγιστη τιμή σε μια λίστα

```
maximum = L[0]
for number in L :
    if number > maximum :
        maximum = number
print maximum
```

4. Ρέστα

Καλούμαστε να σχεδιάσουμε το λογισμικό μιας ταμειακής μηχανής, το οποίο θα διαβάζει το ποσό που έδωσε ο πελάτης και το κόστος των αγορών του και θα εμφανίζει το ελάχιστο πλήθος κερμάτων ή χαρτονομισμάτων που θα δοθούν για ρέστα. Θεωρήστε ότι όλες οι τιμές είναι σε ακέραια πολλαπλάσια του ευρώ:

```
values = [100, 50, 20, 10, 5, 2, 1]
cost = input( “Δώσε το κόστος των αγορών” )
payment = input( “Δώσε το ποσό της πληρωμής” )
change = payment – cost
counter = 0
for value in values :
    counter = counter + ( change / value )   # το στοιχείο value σε κάθε επανάληψη παίρνει
                                             # διαδοχικά τις τιμές 100, 50, 20, 10, 5, 2, 1
change = change % value                      # αφαιρούμε την προηγούμενη τάξη μεγέθους value
print counter
```

Εφαρμογές

1. Διαχωρισμός λίστας

Η λειτουργία του διαχωρισμού μιας λίστας σε δύο μέρη με βάση κάποια κριτήρια, αποτελεί μια τυπική επεξεργασία των λιστών.

```
positives = [ ]
negatives = [ ]
for number in numbers :                  # για κάθε αριθμό της λίστας
    if number > 0 :                      # αν είναι θετικός
        positives.append( number )       # πρόσθεσέ τον στη λίστα με τους θετικούς
    else :
        negatives.append( number )       # αλλιώς στη λίστα με τους αρνητικούς
print positives
print negatives
```

Το παραπάνω πρόγραμμα διαχωρίζει τους αριθμούς μιας λίστας σε αρνητικούς και θετικούς. Υποθέτουμε ότι όλοι οι αριθμοί είναι διάφοροι του μηδενός. Να σημειωθεί ότι η αρχική λίστα παραμένει ανέπαφη.

2. Συγχώνευση διατεταγμένων λιστών

Ένας από τους πιο γνωστούς και χρήσιμους αλγορίθμους της Πληροφορικής είναι ο αλγόριθμος της συγχώνευσης των στοιχείων δυο ταξινομημένων λιστών σε μία νέα, επίσης ταξινομημένη, λίστα. Ο αλγόριθμος αξιοποιεί το γεγονός ότι οι αρχικές λίστες είναι ήδη ταξινομημένες, ώστε να μη χρειαστεί να ταξινομήσει από την αρχή την τελική λίστα, κάτι το οποίο έχει σημαντικό υπολογιστικό κόστος για πολύ μεγάλες λίστες.

```
def merge( A, B ) :
    L = [ ]
    while A != [ ] and B != [ ] :          # όσο οι δυο λίστες έχουν στοιχεία
       if A[0] < B[0] :                    # Αν το 1ο στοιχείο της A είναι το μικρότερο
                                             # από το 1ο στοιχείο της B
            L.append( A.pop(0) )           # μεταφέρουμε το πρώτο στοιχείο
                                             # της A στο τέλος της L
        else :
            L.append( B.pop(0) )           # αλλιώς μεταφέρουμε το πρώτο στοιχείο
                                             # της B στην L
                                        # Αφού αφαιρούμε το στοιχείο[0] μιας λίστα το στοιχείο[1]
                                          # έρχεται αριστερά και γίνεται το νέο στοιχείο [0]
        return L + A + B                   # στο τέλος προσθέτουμε τα στοιχεία που έχουν μείνει
```

Αφού οι δύο λίστες είναι ταξινομημένες σε αύξουσα σειρά το ελάχιστο στοιχείο και των δύο λιστών είναι το μικρότερο από τα A[0], B[0]. Στη συνέχεια μεταφέρουμε αυτό το στοιχείο στην τελική λίστα.

Κάποια στιγμή, μία από τις δυο λίστες θα αδειάσει, οπότε η επανάληψη θα σταματήσει. Η άλλη λίστα όμως θα έχει κάποια στοιχεία τα οποία πρέπει να προστεθούν στο τέλος της τρίτης λίστας. Έτσι θα έπρεπε να γράψουμε:

```
if A == [ ] :
    L = L + B
else :
    L = L + A
```

κάτι που είναι ισοδύναμο με την εντολή L = L + B + A } αφού αν η A είναι κενή το αποτελέσματα είναι L + B + [ ], ενώ αν η B είναι κενή το αποτέλεσμα είναι L + [ ] + A.

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)
```

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
```

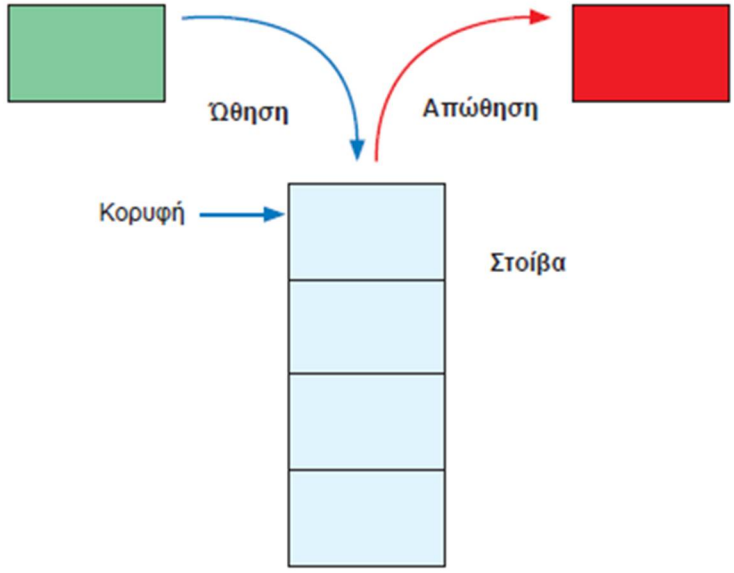
Απάντηση : δημιουργεί μία λίστα με στοιχεία τα τετράγωνα [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Οι λίστες στην Python:
● Δεν έχουν σταθερό μέγεθος, δηλαδή μπορούν να αυξάνονται και να μειώνονται κατά την εκτέλεση του προγράμματος.
● Η αρίθμηση των δεικτών ξεκινάει από το 0, όπως ακριβώς στις συμβολοσειρές.
● Είναι *δυναμικές δομές*, και χαρακτηρίζονται από μεγάλη ευελιξία. Έτσι για παράδειγμα, μπορούμε να έχουμε σε μια λίστα ακόμα και στοιχεία διαφορετικού τύπου.

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.

8.3 Στοίβα

Στοίβα ονομάζουμε τη δομή στην οποία οι εισαγωγές και οι διαγραφές στοιχείων γίνονται από το πάνω άκρο μόνο. Υλοποιείται με τη δομή της λίστας. Στη λίστα αυτή το στοιχείο που προστέθηκε τελευταίο είναι και το πρώτο που θα εξαχθεί, έχουμε δηλαδή μια λειτουργία τύπου LIFO (**L**ast **I**n **F**irst **O**ut), δηλαδή ο τελευταίος που εισέρχεται στη λίστα, είναι και ο πρώτος που θα εξαχθεί. Η συγκεκριμένη δομή δεδομένων ονομάζεται **Στοίβα** και οι λειτουργίες εισαγωγής και εξαγωγής είναι γνωστές ως **ώθηση** και **απώθηση**.



Η στοίβα αποτελεί μια από τις σημαντικότερες δομές δεδομένων της επιστήμης της Πληροφορικής και χρησιμοποιείται σε πολλά πεδία της, όπως είναι η θεωρία αλγορίθμων, η ανάπτυξη μεταγλωττιστών, η τεχνητή νοημοσύνη κ.ά.
Όταν η στοίβα είναι άδεια, είναι προφανές ότι δεν μπορεί να γίνει απώθηση.
Άρα, όταν απωθούμε ένα στοιχείο από τη στοίβα, θα πρέπει προηγουμένως να έχουμε εξασφαλίσει ότι η στοίβα δεν είναι κενή. Για αυτό το λόγο, εκτός από την ώθηση και την απώθηση, πρέπει να υλοποιήσουμε και τον έλεγχο, αν η στοίβα είναι κενή. Άρα οι βασικές λειτουργίες που πρέπει να υποστηρίζει η υλοποίηση μιας στοίβας είναι:
- Δημιουργία μιας κενής στοίβας.
- Έλεγχος, αν η στοίβα είναι κενή.
- Ώθηση ενός στοιχείου στη στοίβα.
- Απώθηση ενός στοιχείου από τη στοίβα.

Η δομή της στοίβας μπορεί να υλοποιηθεί στην Python με μια λίστα στην οποία οι εισαγωγές και οι εξαγωγές στοιχείων γίνονται μόνο από το ένα άκρο.

Υλοποίηση Στοίβας σε Python με δύο τρόπους
Στην πρώτη περίπτωση, οι εισαγωγές/διαγραφές στοιχείων γίνονται στο τέλος της λίστας,

```
def push(stack, item) :              # ώθηση στοιχείου
    stack.append( item )

def pop(stack) :                     # απώθηση στοιχείου
    return stack.pop( )

def isEmpty(stack) :                 # έλεγχος για ύπαρξη στοιχείων
    return len(stack) == 0

def createStack( ) :                 # δημιουργία στοίβας(λίστας)
     return [ ]
```

ενώ στη δεύτερη περίπτωση, οι εισαγωγές/διαγραφές στοιχείων γίνονται στην αρχή της.

```
def push(stack, item) :
    stack.insert(0, item)

def pop(stack) :
    return stack.pop( 0 )

def isEmpty(stack) :
   return len(stack) == 0

def createStack( ) :
    return [ ]
```

Οι δύο υλοποιήσεις που δίνονται παραπάνω, διαφέρουν μόνο ως προς το σημείο της λίστας στο οποίο γίνονται οι εισαγωγές/διαγραφές των στοιχείων.

Εφαρμογή Στοίβας.
Αντιστροφή αριθμών
Το παρακάτω πρόγραμμα δέχεται από το χρήστη αριθμούς μέχρι να δοθεί το 0 και τους εμφανίζει σε αντίστροφη σειρά από αυτή με την οποία δόθηκαν.
Θέλουμε κάθε φορά να εμφανίσουμε πρώτο τον αριθμό που δόθηκε τελευταίος. Χρειαζόμαστε μια δομή δεδομένων που να υποστηρίζει τη λειτουργία LIFO, όπως η στοίβα.

```
stack = createStack()                # Δημιουργία της στοίβας stack
number = int( raw_input( ) )         # Διάβασε τον πρώτο αριθμό
while number != 0 :                  # Όσο δεν δίνεται 0
      push(stack, number)            # Σπρώξε τον αριθμό στη στοίβα
      number = int( raw_input( ) )   # Διάβασε τον επόμενο αριθμό
while not isEmpty( stack ) :         # Μέχρι να αδειάσει η στοίβα
       number = pop(stack)           # βγάλε τον επόμενο αριθμό
       print number                  # και εκτύπωσέ τον
```

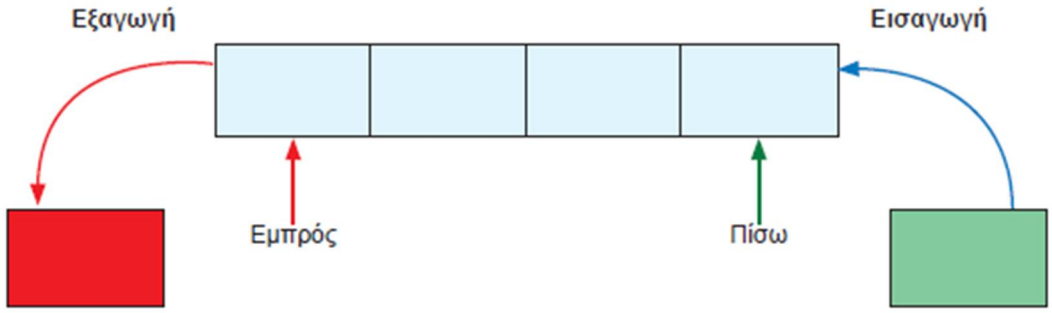
Παρατηρήστε ότι στο παραπάνω πρόγραμμα δεν φαίνεται ποια υλοποίησηχρησιμοποιείται. Αν τροποποιήσουμε την υλοποίηση της ώθησης, δε θα χρειαστεί να αλλάξουμε τίποτα στο πρόγραμμα. Αυτό είναι γνωστό ως *διαχωρισμόςδιεπαφής (διασύνδεσης) – υλοποίησης (interface /*

implementation), αφού οιεφαρμογές που χρησιμοποιούν δομές δεδομένων όπως η στοίβα, είναι απολύτως ανεξάρτητες από την υλοποίηση.
Ουσιαστικά δε μας ενδιαφέρει πώς έχει υλοποιηθεί η στοίβα, αφού εμείς θέλουμε μόνο να χρησιμοποιήσουμε τις συναρτήσεις που έχουμε ορίσει παραπάνω.
Το σύνολο των επικεφαλίδων των συναρτήσεων το οποίο είναι διαθέσιμο στον προγραμματιστή που χρησιμοποιεί τη στοίβα, το ονομάζουμε διεπαφή ή διασύνδεση (interface) της δομής αυτής. Η διεπαφή μιας δομής ορίζει τι μπορεί να κάνει η δομή και όχι τον τρόπο με τον οποίο το κάνει. Το τελευταίο είναι ο ρόλος της υλοποίησης.

Διεπαφή της δομής δεδομένων Στοίβα
def push(stack, item)
def pop(stack)
def isEmpty(stack)
def createStack( )

8.4 Ουρά

Μια δομή δεδομένων που χρησιμοποιείται για την μοντελοποίηση και προσομοίωση πραγματικών φαινομένων, είναι η δομή της ουράς. Τα φαινόμενα που μοντελοποιούνται αναφέρονται στην εξυπηρέτηση ανθρώπων, αντικειμένων ή προγραμμάτων.
Σε αντίθεση με τη στοίβα, που η λειτουργία της χαρακτηρίζεται ως LIFO (**L**ast **I**n **F**irst **O**ut), η λειτουργία της ουράς είναι γνωστή στη βιβλιογραφία ως FIFO (First In First Out), αφού το κάθε στοιχείο της ουράς εξυπηρετείται με τη σειρά που έφτασε στην ουρά.



Τα φαινόμενα αυτά μελετώνται από διάφορους κλάδους των Μαθηματικών και της Πληροφορικής, όπως είναι η Θεωρία Ουρών (Queueing theory) και η Επιχειρησιακή 'Ερευνα (Operations Research).

Δύο είναι οι βασικές λειτουργίες μιας ουράς:
- Εισαγωγή στοιχείου, η οποία γίνεται στο πίσω μέρος της ουράς.
- Εξαγωγή στοιχείου, η οποία γίνεται από το εμπρός μέρος της ουράς.

Υλοποίηση Ουράς σε Python

```
def enqueue(queue, item) :          # εισαγωγή στοιχείου
queue = queue.append( item )

def dequeue(queue) :                # εξαγωγή στοιχείου
return queue.pop( 0 )

def isEmpty(queue) :                # έλεγχος για ύπαρξη στοιχείων
    return len(stack) == 0

def createQueue( ) :                # δημιουργία ουράς (λίστας)
return [ ]
```

8.5 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 :
Στις πλειάδες, όπως συμβαίνει και με τις λίστες τα στοιχεία χωρίζονται με κόμματα, όπως στις λίστες, αλλά, αντί να περικλείονται από αγκύλες, περικλείονται σε παρενθέσεις, οι οποίες όμως δεν είναι υποχρεωτικές. 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’,)

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).

Συνάρτηση tuple (αντικείμενο)
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.

Οι πλειάδες είναι αμετάβλητες
Οι πλειάδες, όπως και οι συμβολοσειρές, είναι **μη τροποιήσιμες (immutables)**. 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’
print rec # *θα μας δώσει* (‘a’, ‘b’, 120, ‘r’)
rec [1:3] # *θα μας δώσει* (‘b’, 120)
rec [2] **#** *θα μας δώσει* 120
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 (swap)** δηλαδή η αμοιβαία αλλαγή των τιμών δυο μεταβλητών, χωρίς τη χρήση βοηθητικής μεταβλητής**,** με την παρακάτω εντολή:
a , b = b, a

8.6 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}

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.

Συνάρτηση dict
Μπορούμε να δημιουργήσουμε ένα λεξικό και με τη χρήσης της ενσωματωμένης συνάρτησης **dict** (). π.χ. d = **dict** (one=1, two=2, three=3)

Προθήκη νέου ζεύγους κλειδιού : τιμής
Θα πρέπει να σημειωθεί ότι κάθε κλειδί έχει μοναδική τιμή
Για να προσθέσουμε ένα νέο ζεύγος σε ένα λεξικό χρησιμοποιούμε την εκχώρηση
Λεξικό [κλειδί] = τιμή οπότε και δημιουργείται το νέο ζεύγος {κλειδί : τιμή}.

Παρατήρηση : Αν το κλειδί υπάρχει ήδη στο λεξικό, η τιμή που είχε ενημερώνεται και στη θέση της μπαίνει η νέα τιμή, *Τιμή*, ενώ αν δεν υπάρχει, τότε δημιουργείται ένα 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 πλειάδες

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 μας δώσει τη λίστα  ['one', 'two', 'three']
for value in values_view:
   print (value)               θα μας δώσει τη λίστα  [1, 2, 3]
for key, value in items_view:
   print (key, value)       θα μας δώσει τη λίστα των πλειάδων  [ (‘one', 1), ('two', 2), ('three', 3) ]
```

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

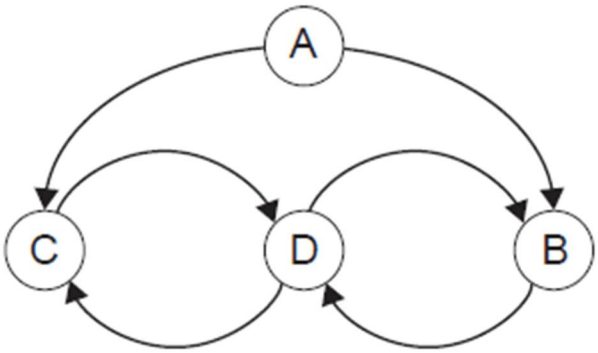
Ο συνδυασμός λεξικού και πλειάδας ή λίστας, έχει πολλές εφαρμογές όπως για παράδειγμα στην αναπαράσταση **γράφων**. Η δομή του γραφήματος (Λίστα γειτνίασης) στην Python υλοποιείται με ένα λεξικό, με κλειδιά τις κορυφές του γράφου και τιμές τη λίστα ακμών κάθε κλειδιού, δηλαδή τη λίστα των γειτονικών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

```
graph = { ‘A’: [‘B’, ‘C’, ‘D’],
‘B’: [‘D’]
‘C’: [‘D’]
‘D’: [‘B’, ‘C’] }
```

8.7 Εισαγωγή στους Γράφους και τα δέντρα

8.7 Εισαγωγή στους γράφους και τα δέντρα

Ένας **Γράφος** είναι μια δομή που αποτελείται από ένα σύνολο *κορυφών* ή *κόμβων* και ένα σύνολο *ακμών* μεταξύ των κορυφών. Οι κορυφές θα μπορούσαν να είναι οι πόλεις μιας χώρας και οι ακμές οι δρόμοι που τις ενώνουν. Όταν οι δρόμοι είναι διπλής κυκλοφορίας, λέμε ότι ο γράφος είναι **μη κατευθυνόμενος**, ενώ, όταν κάποιοι από τους δρόμους είναι μονόδρομοι, λέμε ότι ο γράφος είναι *κατευθυνόμενος*. Παρακάτω δίνεται ένας κατευθυνόμενος γράφος με 4 κόμβους
A, B, C, D.



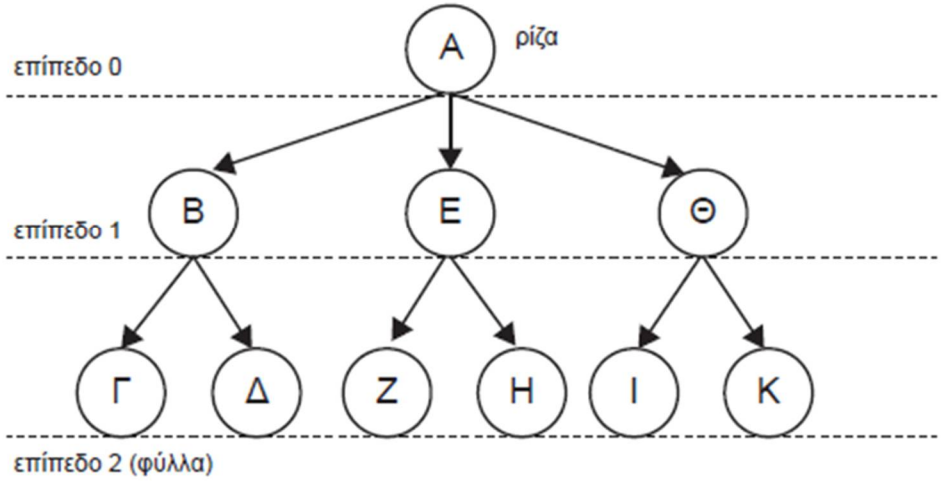
Για να μεταβούμε από την κορυφή C στη B δεν υπάρχει απευθείας ακμή. Πρέπει πρώτα να περάσουμε από τη D. Η ακολουθία των ακμών από τις οποίες περνάμε για να φτάσουμε στον τελικό προορισμό, λέγεται *μονοπάτι*. Το μονοπάτι από τη C στη B συμβολίζεται C → D → B και έχει μήκος 2 αφού αποτελείται από 2 ακμές. Δηλαδή, το *μήκος ενός μονοπατιού* είναι το πλήθος των ακμών του μονοπατιού. Για τους γράφους που εξετάζουμε θα θεωρήσουμε ότι όλες οι ακμές έχουν το ίδιο μήκος ή κόστος.
Παρατηρήστε ότι δεν υπάρχει κανένα μονοπάτι από κάποια κορυφή που να καταλήγει στην A. Ωστόσο, αν ακολουθήσουμε το μονοπάτι B → D → C, μπορούμε να επιστρέψουμε πίσω στη B, αφού υπάρχει δρόμος C → D → B. Αυτό ονομάζεται *κύκλος* ή *κυκλικό μονοπάτι*, επειδή, αν το ακολουθήσουμε επιστρέφουμε στο σημείο από το οποίο ξεκινήσαμε.

Η δομή ενός γράφου συναντάται σε πολλά προβλήματα της επιστήμης της Πληροφορικής, όπως για παράδειγμα στην αναπαράσταση του παγκόσμιου ιστού, την κατάταξη των ιστοσελίδων (page ranking), την εύρεση ενός μονοπατιού ή του συντομότερου μονοπατιού σε ένα δίκτυο πόλεων και άλλα.

Μια ειδική περίπτωση γράφων είναι τα δέντρα.
Ένας γράφος στον οποίο υπάρχει μονοπάτι μεταξύ δυο οποιωνδήποτε κορυφών, και δεν περιέχει κυκλικές διαδρομές, λέγεται **Δέντρο**.
Το *δέντρο* συνιστά μια ιεραρχική δομή η οποία χρησιμοποιείται για τη μοντελοποίηση διάφορων καταστάσεων, όπως για παράδειγμα ένα γενεαλογικό δέντρο ή η οργανωτική δομή μιας επιχείρησης. Αναφερόμαστε στα στοιχεία του δέντρου, όπως και του γράφου, ως *κόμβους*. Κάθε κόμβος συνδέεται με έναν ή περισσότερους κόμβους στους οποίους αναφερόμαστε, ως *παιδιά* του ή **απόγονούς** του. Οι κόμβοι που δεν έχουν απογόνους και βρίσκονται στο τελευταίο επίπεδο λέγονται *φύλλα* του δέντρου. Ο αριθμός των παιδιών που έχει ένας κόμβος ονομάζεται *βαθμός* του κόμβου. Ο μέγιστος βαθμός μεταξύ των κόμβων ενός δέντρου είναι ο βαθμός του *δέντρου*.

Κάθε κόμβος έχει ακριβώς έναν πρόγονο, εκτός από τη **ρίζα**, που βρίσκεται συνήθως στην κορυφή του δέντρου. Η απόσταση ενός κόμβου από τη ρίζα είναι το *επίπεδο* του κόμβου. Προφανώς, η ρίζα βρίσκεται στο μηδενικό (0) επίπεδο.
Το *ύψος* ενός δέντρου είναι η μέγιστη απόσταση κάποιου κόμβου από τη ρίζα.

Όπως φαίνεται στο παρακάτω σχήμα, μέσω των επιπέδων, διακρίνεται πιο εύκολα η ιεραρχική δομή του δέντρου. Το δέντρο του σχήματος έχει ύψος 2, αφού το τελευταίο επίπεδο είναι το 2ο.



Σύνοψη

Οι βασικές δομές δεδομένων της Python που είναι ενσωματωμένες στη γλώσσα χωρίζονται σε δυο κατηγορίες. Αυτές που επιτρέπουν την τροποποίηση των περιεχομένων τους (immutables), όπως οι λίστες και τα λεξικά και αυτές που δεν το επιτρέπουν, όπως οι συμβολοσειρές και οι πλειάδες.