ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΙΔΡΥΜΑ ΠΑΤΡΩΝ

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΟΙΚΟΝΟΜΙΑΣ

ΤΜΗΜΑ ΕΠΙΧΕΙΡΗΜΑΤΙΚΟΥ ΣΧΕΔΙΑΣΜΟΥ ΚΑΙ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΜΕΛΕΤΗ ΤΗΣ ΓΛΩΣΣΑΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ PYTHON ΚΑΙ ΤΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΩΝ ΜΟΝΤΕΛΩΝ ΠΟΥ ΥΠΟΣΤΗΡΙΖΕΙ

ΣΠΟΥΔΑΣΤΕΣ:
ΕΥΣΤΑΘΙΟΣ ΚΑΛΟΥΛΗΣ, ΑΜ 1799
ΓΕΩΡΓΙΟΣ ΠΕΤΣΟΥΛΗΣ, ΑΜ 1825

ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ: ΔΡ. ΣΤΑΜΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΠΑΤΡΑ - 2012

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 8ο

ΠΕΡΙΕΧΟΜΕΝΑ ΕΙΚΟΝΩΝ

ΠΕΡΙΛΗΨΗ

Η python είναι μια απλή και εύκολη στη διαχείριση γλώσσα προγραμματισμού και δημιουργήθηκε από τον Guido Van Rossum. Είναι δυναμική γλώσσα και δεν χρειάζεται μεταγλώττιση. Δεν χρησιμοποιεί αποκλειστικά ένα μοντέλο, αλλά υποστηρίζει αρκετά προγραμματιστικά μοντέλα, όπως ο δομημένος και αντικειμενοστραφής προγραμματισμός, που έχουν πλήρη εφαρμογή στην python.

Είναι μια scripting γλώσσα και υπάρχει η δυνατότητα να δημιουργηθούν προγράμματα που θα είναι γραμμένα και σε άλλες γλώσσες όπως η C ή τη java. Δεν χρειάζεται ο προγραμματιστής να παράγει κάθε φορά κώδικα, αφού του δίνεται η δυνατότητα να χρησιμοποιεί συναρτήσεις και modules από την πλούσια βιβλιοθήκη της python και να δημιουργεί τις δικές του να τις προσθέτει στη βιβλιοθήκη.

Επιπλέον, είναι ανοικτού κώδικα γλώσσα με άδεια χρήσης από το Python Software Foundation (PSF) και είναι ελεύθερη η χρήση της. Αυτό επιτρέπει στην ευκολότερη ανάπτυξη της γλώσσας και τη μη χρονοβόρα συγγραφή προγραμμάτων, λόγω της χρήσης κώδικα άλλων προγραμματιστών.

ΠΡΟΛΟΓΟΣ

Το θέμα της πτυχιακής εργασίας είναι η μελέτη της γλώσσας προγραμματισμού python και των προγραμματιστικών μοντέλων που υποστηρίζει. Η παρούσα εργασία εκπονήθηκε από τους σπουδαστές Ευστάθιο Καλούλη και Γεώργιο Πέτσουλη.

Θα θέλαμε να ευχαριστήσουμε τον επιβλέποντα καθηγητή μας, Δρ. Κωνσταντίνο Στάμο για την βοήθεια του και τις πολύτιμες συμβουλές που μας προσέφερε, όλο αυτό το διάστημα.

Με την ευκαιρία αυτή θέλουμε να πούμε και ένα μεγάλο ευχαριστώ στις οικογένειες μας, που μας στήριξαν σε όλη τη διάρκεια των προπτυχιακών σπουδών μας.

ΕΙΣΑΓΩΓΗ

Σε αυτή την εργασία, θα μελετηθεί η προγραμματιστική γλώσσα python και θα αναλυθεί ταυτόχρονα και ο τρόπος λειτουργίας κάθε μοντέλου. Στο πρώτο κεφάλαιο γίνεται μια εισαγωγή στη γλώσσα και παρουσιάζονται, οι λόγοι που δημιουργήθηκε και οι ανάγκες που έπρεπε να καλύψει. Αναφέρονται επίσης τα χαρακτηριστικά της και πως μπορεί να εγκατασταθεί στα βασικά λειτουργικά συστήματα.

Στο κεφάλαιο 2, μελετάται ο τρόπος που εκτελούνται βασικές λειτουργίες στην python, όπως εμφάνιση μηνυμάτων, δημιουργία μεταβλητών και εκτέλεση αριθμητικών πράξεων. Περιγράφεται, ο τρόπος εισαγωγής από τη βιβλιοθήκη διάφορων modules και συναρτήσεων και πως δουλεύουν στην python, αλλά και η δημιουργία καινούργιων.

Στο κεφάλαιο 3 μελετάται η λειτουργία του δομημένου προγραμματισμού και του συναρτησιακού μοντέλου, που έχει κάποια εφαρμογή στην python. Παρουσιάζεται ο τρόπος λειτουργίας των ακολουθιών στην python όπως οι λίστες, τα λεξικά και τα σύνολα. Στο συναρτησιακό μοντέλο, υπάρχουν οι λειτουργίες των συναρτήσεων filter, map και reduce.

Στο κεφάλαιο 4, παρουσιάζεται ο τρόπος λειτουργίας του αντικειμενοστραφούς μοντέλου προγραμματισμού στην python. Μερικά από τα χαρακτηριστικά του μοντέλου αυτού, που αναλύονται σε αυτό το κεφάλαιο είναι οι ειδικές μέθοδοι, και η κληρονομικότητα. Γίνεται μελέτη επίσης, του προσανατολισμένου μοντέλου προγραμματισμού και πιο συγκεκριμένα, της στατικής μεθόδου.

Στο κεφάλαιο 5, γίνεται μελέτη του έλεγχου ροής στην python. Μελετάται η δομή επιλογής αλλά και οι δομές επανάληψης που χρησιμοποιούνται. Στο 6 κεφάλαιο μελετάται ο τρόπος εισαγωγής και εμφάνισης των δεδομένων καθώς και η λειτουργία των αρχείων.

Στο κεφάλαιο 7, παρουσιάζονται οι εξαιρέσεις και μελετώνται οι λειτουργίες που μπορούν να γίνουν. Τέλος, στο κεφάλαιο 8, γίνεται μελέτη της έκφρασης λάμδα από το συναρτησιακό μοντέλο προγραμματισμού.

ΚΕΦΑΛΑΙΟ 1ο
Εισαγωγή στην Python.

Η python είναι μια δυναμική γλώσσα γενικής χρήσης και υψηλού επιπέδου. Είναι ανοικτού κώδικα και αυτή η ιδιότητα, συντελεί στην ύπαρξη μιας πλούσιας βιβλιοθήκης προγραμμάτων. Όλες αυτές οι ιδιότητες επιτρέπουν τον προγραμματιστή να κατασκευάσει ότι θέλει, χωρίς να περιορίζεται. Στα βασικά της χαρακτηριστικά, είναι η ευκολία στην εκμάθηση και στην ανάπτυξη προγραμμάτων, κάτι που επιτρέπει στο χρήστη να δημιουργεί αρκετά απαιτητικές εφαρμογές σε ελάχιστο χρόνο. Για τους παραπάνω λόγους, χρησιμοποιείται και ως γλώσσα προγραμματισμού σεναρίων (scripting language).

1.1 Ιστορική αναδρομή.

Από ιστορικής πλευράς, η ανάπτυξη της ξεκίνησε από τον Ολλανδό Guido Van Rossum, στο τέλος του 1989 στο εθνικό ινστιτούτο έρευνας και πληροφορικής της Ολλανδίας (CWI), που βρίσκεται στο Άμστερνταμ. Η ιδέα της python βασίστηκε πάνω στην ABC γλώσσα προγραμματισμού και δημιουργήθηκε για να καλύψει τα κενά που είχαν άλλες γλώσσες προγραμματισμού της εποχής εκείνης, όπως η ABC. Η ονομασία της προήλθε από μια αγαπημένη εκπομπή του Guido, το «ιπτάμενο τσίρκο» των κωμικών Monty Pythons (Monty Python's Flying Circus), μια εκπομπή του bbc. Η φιλοσοφία της είναι η ίδια με εκείνη των Monty Pythons όταν ξεκίνησαν το ιπτάμενο τσίρκο: «Και τώρα κάτι τελείως διαφορετικό». Μέσα στους πρώτους μήνες του 1990 είχε κατασκευαστεί μια λειτουργική έκδοση της python, ενώ η πρώτη της έκδοση (έκδοση 0.9.1), κυκλοφόρησε στις 20 Φεβρουαρίου του 1991.

Εικόνα 1- Η δημοσίευση της πρώτης έκδοσης το 1991.

Η πρώτη έκδοση δημοσιεύτηκε σε όλο τον κόσμο, μέσα από μια ομάδα συζητήσεων προγραμματιστών (alt.sources newsgroup) [1] και αποτελούταν από 21 πακέτα σε μετάφραση Uuencoding (Unix-to-Unix κωδικοποίηση), τα οποία έπρεπε να ενωθούν για να σχηματιστεί όλη η πρώτη έκδοση της python.[2]

[1] Guido van Rossum (1991), Group: alt.sources, Python 0.9.1 part 01/21, Ανάκτηση στις 5-11-2011 από http://groups.google.com/group/alt.sources/browse_thread/thread/3b6652abb0e23b03/53ee3620bc53c281

[2] Guido van Rossum, (2009), Personal History - part 1 CWI, Ανάκτηση στις 5-11-2011 από http://python-history.blogspot.com/2009/01/personal-history-part-1-cwi.html

Εικόνα 2 – Πρώτη έκδοση από τον Guido van Rossum.

Η python είναι μια σύγχρονη και δυναμική γλώσσα προγραμματισμού και υποστηρίζει πολλά μοντέλα εκτός από το διαδικαστικό προγραμματισμό, όπως είναι ο αντικειμενοστραφής και ο συναρτησιακός.

Όπως αναφέρθηκε, είναι γλώσσα ανοικτού κώδικα και κυκλοφορεί με άδεια χρήσης από το PSF (Python Software Foundation) σε δύο εκδόσεις, που κατά τη συγγραφή της συγκεκριμένης μελέτης υπάρχουν, η 2.7 και την 3.2 οι οποίες αναπτύσσονται παράλληλα. Οι εκδόσεις της 2 είναι με λιγότερα σφάλματα, διότι σε κάθε ενημέρωση διορθώνονται τα σφάλματα των παλαιότερων εκδόσεων. Η έκδοση 3 χρησιμοποιείται από έμπειρους προγραμματιστές και εφαρμόζει κάποια νέα είδη λειτουργίας σε διάφορες εκφράσεις και εντολές. Αυτό έχει ως αποτέλεσμα να μην είναι συμβατή με παλαιότερες εκδόσεις. Όπως είναι λογικό, η τρίτη έκδοση δεν έχει δοκιμαστεί στο επίπεδο της δεύτερης έκδοσης, με αποτέλεσμα να εμφανίζονται διάφορα σφάλματα. Όταν δεν χρειάζεται να γίνει εκτενής χρήση της γλώσσας, αλλά απλές εφαρμογές όπως στην τρέχουσα μελέτη, καλό είναι να χρησιμοποιείται η έκδοση 2, που είναι δοκιμασμένη και ολοκληρωμένη μέχρι ενός σημείου.

1.2 Τα βασικά χαρακτηριστικά της python.

Τα βασικά στοιχεία της python παρουσιάζονται παρακάτω:

1. Αρχικά, είναι μια απλή γλώσσα προγραμματισμού και αυτό, καθιστά την εκμάθηση της εύκολη. Όταν γράφεται κώδικας στην python, δεν θα προκύψουν ποτέ ζητήματα διαχείρισης της μνήμης, επειδή είναι μια γλώσσα υψηλού επιπέδου.
2. Ως γλώσσα ανοιχτού κώδικα, έχει με τον καιρό προσαρμοστεί από τους χρήστες και είναι λειτουργική σε πολλές πλατφόρμες. Μερικές από αυτές είναι τα windows, το linux και τα mac.
3. Η python είναι μια διερμηνευόμενη γλώσσα και δεν χρειάζεται ο κώδικας της να μεταγλωττιστεί, πριν την εκτέλεση ενός προγράμματος σε 0 και 1, όπως γίνεται σε άλλες γλώσσες (C). Το πρόγραμμα θα τρέξει κατευθείαν από τον κώδικα που έχει γραφτεί. Γενικά οι διερμηνευόμενες γλώσσες δίνουν δυνατότητες, όπως οι ακόλουθες:

- Ταχεία υλοποίηση προγραμμάτων,
- Επεξεργασία διαφορετικών δεδομένων από πολλές πηγές και
- Άμεση σύνδεση και συνεργασία ενός προγράμματος με ένα άλλο.

4. Υποστηρίζει το μοντέλο του αντικειμενοστραφούς προγραμματισμού. Κατασκευή δηλαδή προγραμμάτων, που βασίζονται στην αλληλεπίδραση των διάφορων αντικειμένων. Ο τρόπος που λειτουργεί ο αντικειμενοστραφής προγραμματισμός στην python, είναι αρκετά απλούστερος, σε σχέση με άλλες γλώσσες προγραμματισμού (όπως Java και C++), αλλά και ισχυρός ταυτόχρονα.
5. Επίσης, είναι επεκτάσιμη γλώσσα και παρακολουθεί πολύ καλά τις εξελίξεις. Η βασική της λειτουργία είναι η ίδια, αλλά αναπτύσσονται συνέχεια νέα modules από τους χρήστες της. Το γεγονός, ότι είναι ελεύθερη γλώσσα βοηθάει στην περαιτέρω εξέλιξη της. Ήδη η έκδοση 3 που αναπτύσσεται αυτή την περίοδο είναι ένα στοιχείο της εξέλιξης της γλώσσας.
6. Είναι μια scripting γλώσσα, χρησιμοποιείται δηλαδή για να γράφονται εφαρμογές. Υποστηρίζει προγράμματα, που έχουν κατασκευαστεί σε άλλες γλώσσες και ενσωματώνονται στην python. Για τη χρήση κώδικα C και Java, μπορούν να εγκατασταθούν τα πρόσθετα Cython και Jython αντίστοιχα.

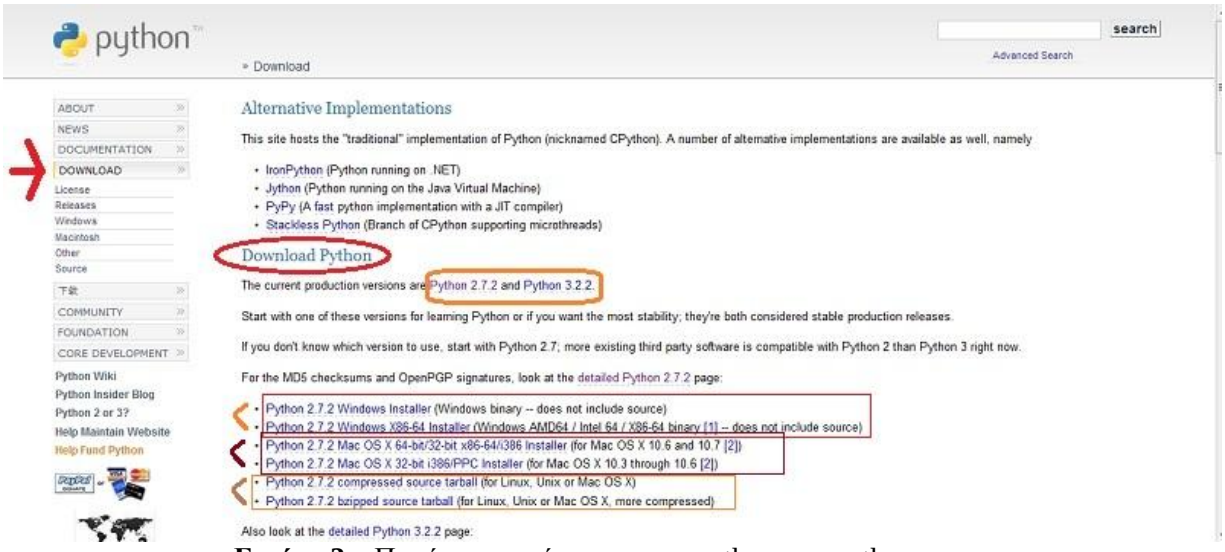
7. Τέλος, ένα ακόμα χαρακτηριστικό της, είναι η πλούσια βιβλιοθήκη που διαθέτει και δίνει τη δυνατότητα στο χρήστη να χρησιμοποιήσει όλα αυτά τα εργαλεία άμεσα.

1.3 Εγκατάσταση της python.

Αρχικά, πρέπει να εγκατασταθεί η python στο λειτουργικό σύστημα. Ενημερωμένη έκδοσης της, υπάρχει στην επίσημη ιστοσελίδα της python (python.org). Αφού γίνει η εγκατάσταση και ανοίξει ο διερμηνευτής της python, θα είναι όλα έτοιμα για την εκτέλεση εντολών.

Η python, κατά την συγγραφή της τρέχουσας μελέτης, διαθέτει δύο εκδόσεις για εγκατάσταση, που είναι η 2.7.2 και η 3.2.2. Στην κατηγορία download, βρίσκονται τα πακέτα εγκατάστασης των βασικότερων λειτουργικών συστημάτων (windows, mac unix και linux). Ανάλογα με το λειτουργικό σύστημα, θα εγκατασταθεί και το αντίστοιχο πακέτο. Για τη μελέτη, αυτή εγκαταστάθηκε η python 2.7.2 σε λειτουργικό σύστημα windows.

Στην ιστοσελίδα της python και στην κατηγορία other, υπάρχουν πακέτα για όλα τα λειτουργικά συστήματα, που υποστηρίζουν την python.



Εικόνα 3 – Πακέτα εγκατάστασης της python στο python.org.

1.3.1 Windows

Αν χρησιμοποιούνται τα windows ως λειτουργικό σύστημα, θα εγκατασταθεί και το αντίστοιχο πακέτο. Υπάρχουν δύο πακέτα, ένα για τα windows των 32-bit και ένα των 64-bit.

1.3.2 Mac, Unix και Linux.

Το ίδιο θα γίνει και στη περίπτωση κάποιου άλλου λειτουργικού συστήματος, όπως mac ή linux. Εδώ όμως το πιθανότερο είναι, να υπάρχει ήδη εγκατεστημένη η python. Για να ελεγχθεί αυτό, θα εκτελεστεί η εντολή: python –V. Η εντολή αυτή θα εμφανίσει την έκδοση, που είναι εγκατεστημένη στο σύστημα. Αν δεν επιστραφεί κάτι, σημαίνει ότι δεν υπάρχει και θα πρέπει να εγκατασταθεί. Στα linux, η εντολή θα τρέξει σε ένα πρόγραμμα κελύφους, όπως είναι το console. Για τα mac, θα χρησιμοποιηθεί το terminal.app και εκεί θα τρέξει η παραπάνω εντολή.[3]

[3] Swaroop CH, «*A Byte of Python*» (Ελληνική μετάφραση), Εγκατάσταση, Ανάκτηση στις 10-11-2011 από http://www.swaroopch.org/notes/Python_el:%CE%A0%CE%B5%CF%81%CE%B9%CE%B5%CF%87%CF%8C%CE%BC%CE%B5%CE%BD%CE%B1

Κατά την εγκατάσταση του πακέτου, δεν πρέπει να αποεπιλεγούν τα προεπιλεγμένα προαιρετικά στοιχεία που εμφανίζονται, διότι είναι ιδιαίτερα χρήσιμα εργαλεία για τον προγραμματισμό με την python.

ΚΕΦΑΛΑΙΟ 2^{ο}
Η βασική λειτουργία της Python.

Για να εκτελεστούν εντολές στην python, πρέπει να ανοίξει ο διερμηνευτής ή το διαδραστικό κέλυφος (Python GUI). Οι εντολές γράφονται μετά την τη χαρακτηριστική σήμανση (>>>) και στην περίπτωση, που υπάρχει block κώδικα χρησιμοποιούνται τρείς τελείες (…). Με την εντολή print θα τυπώσει το μήνυμα που παρουσιάζεται.

```
>>> print'Καλώς ορίσατε στην python!'
Καλώς ορίσατε στην python!
```

Η μελέτη της python γίνεται με βάση τις οδηγίες χρήσης της, που είναι διαθέσιμες στα έγγραφα της επίσημης ιστοσελίδας.[4]

2.1 Εισαγωγή σχολίων στον κώδικα.

Όπως και σε άλλες γλώσσες προγραμματισμού, πολλές φορές χρειάζεται να προστεθούν κάποια σχόλια, κατά τη συγγραφή των προγραμμάτων, για να είναι κατανοητή η λειτουργία του κώδικα. Για την εισαγωγή ενός σχόλιου στην python, πρέπει να χρησιμοποιηθεί το σύμβολο της δίεσης # πριν από το σχόλιο. Έτσι ο διερμηνέας, κατά την εκτέλεση του προγράμματος, θα αγνοήσει όλους του χαρακτήρες μετά από την δίεση.

2.2 Αριθμοί και εκφράσεις.

Στην python μπορούν να εκτελεστούν αριθμητικές πράξεις, αφού υποστηρίζει τους κλασικούς αριθμητικούς τελεστές όπως:

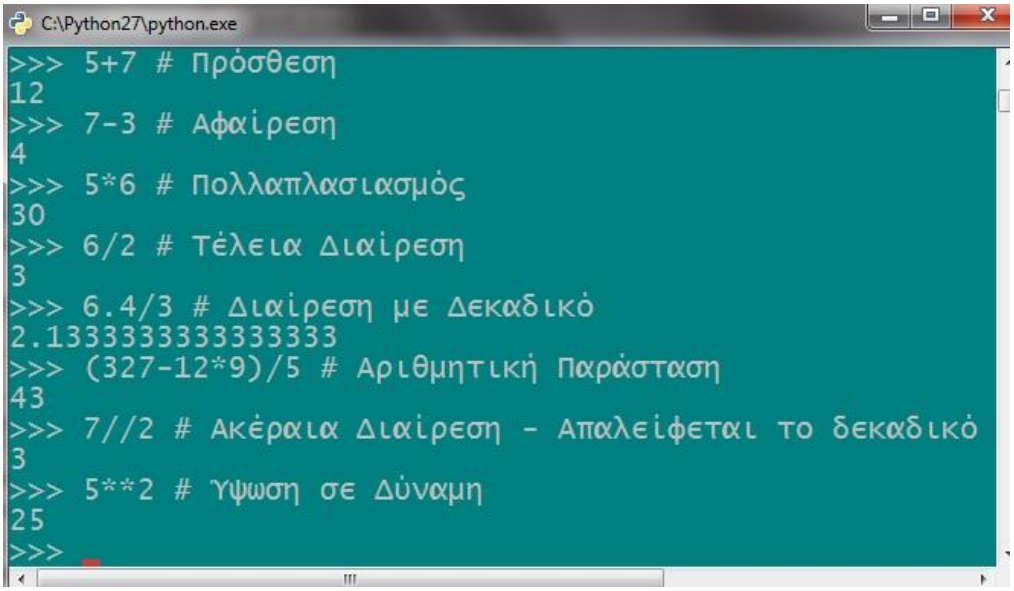
- Πρόσθεση
- Αφαίρεση
- Πολλαπλασιασμός
- Τέλεια διαίρεση
- Διαίρεση με δεκαδικό
- Αριθμητική παράσταση
- Ακέραια διαίρεση (απαλείφεται το δεκαδικό)
- Ύψωση σε δύναμη

Εκτελούνται ενδεικτικά παρακάτω, μερικές αριθμητικές πράξεις στη γραμμή εντολών της python:

```
>>> 5+7 # Πρόσθεση
12
>>> 7-3 # Αφαίρεση
4
>>> 5*6 # Πολλαπλασιασμός
30
>>> 6/2 # Τέλεια Διαίρεση
3
>>> 6.4/3 # Διαίρεση με Δεκαδικό
2.1333333333333333
>>> (327-12*9)/5 # Αριθμητική Παράσταση
```

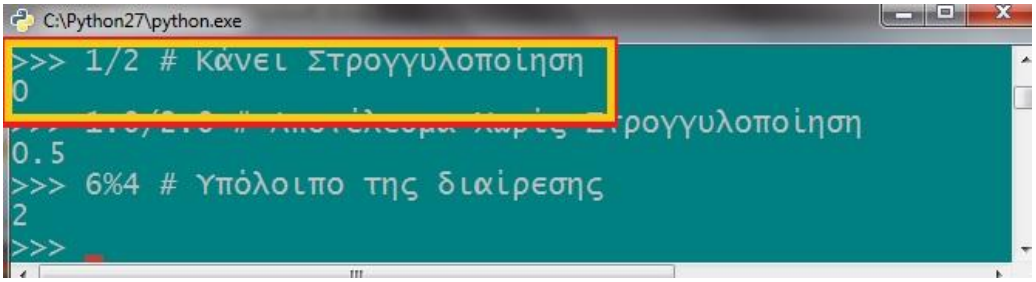
[4] Ανάλογα με την έκδοση που χρησιμοποιείται, υπάρχουν και οι αντίστοιχες οδηγίες στο documentation της python, στη διεύθυνση http://docs.python.org/.

43
>>> 7//2 # Ακέραια Διαίρεση - Απαλείφεται το δεκαδικό
3
>>> 5**2 # Ύψωση σε Δύναμη
25



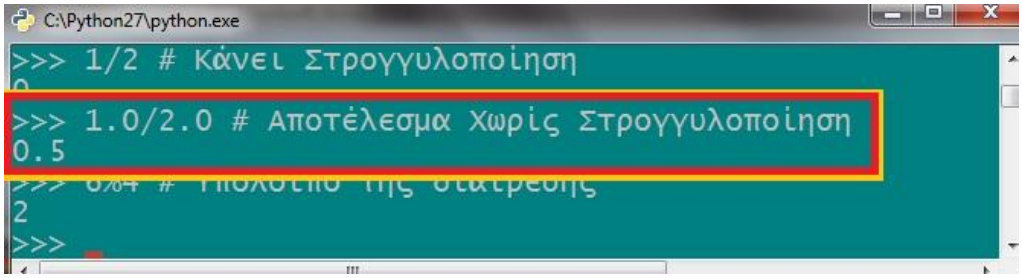
Εικόνα 4 – Εκτέλεση αριθμητικών πράξεων στη γραμμή εντολών.

Οι περισσότεροι αριθμητικοί τελεστές λειτουργούν κανονικά, με εξαίρεση την διαίρεση με ακέραιο. Σε εκδόσεις παλαιότερες της 3, γίνεται στρογγυλοποίηση του αποτελέσματος, ώστε αυτό να αποτελεί ακέραιο αριθμό:
>>> 1/2 # Κάνει Στρογγυλοποίηση
0



Εικόνα 5 – Στρογγυλοποίηση σε διαίρεση.

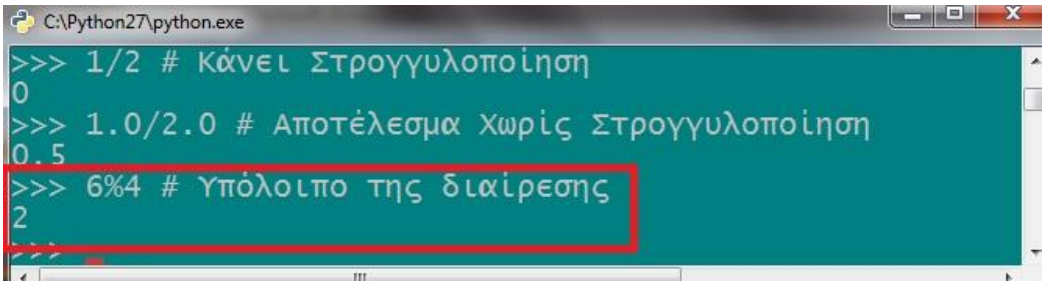
Για αποτέλεσμα χωρίς στρογγυλοποίηση, πρέπει να χρησιμοποιηθούν πραγματικοί αριθμοί (floats):
>>> 1.0/2.0 # Αποτέλεσμα Χωρίς Στρογγυλοποίηση
0.5



Εικόνα 6 – Αποτέλεσμα διαίρεσης χωρίς στρογγυλοποίηση.

Ένας ακόμα χρήσιμος τελεστής είναι ο %, που δίνει το υπόλοιπο διαίρεσης:

```
>>> 6 % 4 # Υπόλοιπο της διαίρεσης
2
```



Εικόνα 7 – Υπόλοιπο διαίρεσης.

Ο διερμηνευτής τηρεί αυστηρή προτεραιότητα στην εκτέλεση των πράξεων, όπως διαπιστώθηκε στην παραπάνω αριθμητική παράσταση. Για παράδειγμα, οι παραστάσεις 3+6*5 και (3+6)*5 θα δώσουν διαφορετικό αποτέλεσμα.

```
>>> 3+6*5
33
>>> (3+6)*5
45
```

2.3 Εκχώρηση τιμών στις μεταβλητές.

Οι μεταβλητές δεν έχουν τύπους στην python και αυτό επιτρέπει την εκχώρηση τιμών, χωρίς την υποχρέωση της δήλωσης του τύπου της μεταβλητής. Για να δηλωθεί οτιδήποτε (ακέραιος ή πραγματικός αριθμός, λίστα, αντικείμενο κ.α) αρκεί, το όνομα της μεταβλητής και η τιμή της, μετά από το ίσον (=). Θα εκχωρηθεί στη μεταβλητή z η τιμή 4 με τον εξής τρόπο:

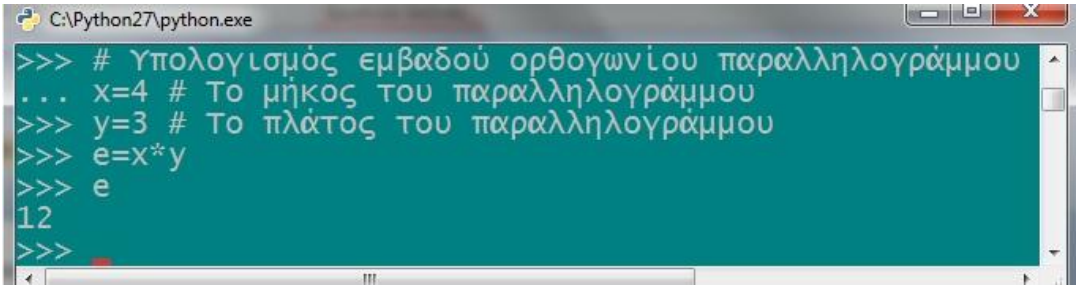
```
>>>z = 4
```

Μπορεί να χρησιμοποιηθεί επίσης, η μεταβλητή z και σε εκφράσεις:

```
>>>z*4
16
```

Παράδειγμα υπολογισμού του εμβαδού ενός ορθογωνίου παραλληλογράμμου:

```
>>> # Υπολογισμός εμβαδού ορθογωνίου παραλληλογράμμου
... x=4 # Το μήκος του παραλληλογράμμου
>>> y=3 # Το πλάτος του παραλληλογράμμου
>>> e=x*y
>>> e
12
```



Εικόνα 8 - Παράδειγμα υπολογισμού εμβαδού παραλληλογράμμου.

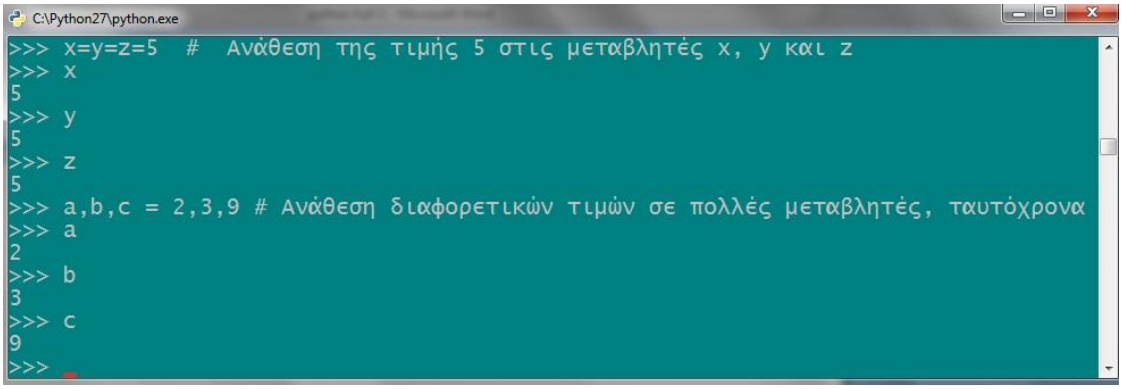
2.3.1 Ανάθεση τιμών σε πολλές μεταβλητές ταυτόχρονα.

Για να εξοικονομηθεί πολύτιμος χρόνος στην python, επιτρέπεται η ανάθεση τιμών σε πολλές μεταβλητές ταυτόχρονα, όπως φαίνεται στα παρακάτω παραδείγματα: [5]

```
>>> x=y=z=5  #  Ανάθεση της τιμής 5 στις μεταβλητές x, y και z
>>> x
5
>>> y
5
>>> z
5
```

Για τον ορισμό διαφορετικής τιμής σε κάθε μεταβλητή, κάθε μια χωρίζεται με κόμμα και μετά το ίσον, μπαίνουν οι αντίστοιχες τιμές των μεταβλητών. Σε αυτό το παράδειγμά, η μεταβλητή a θα πάρει την τιμή 2, η b την τιμή 3 και η c το 9.

```
>>> a,b,c = 2,3,9 # Ανάθεση διαφορετικών τιμών σε πολλές μεταβλητές, ταυτόχρονα
>>> a
2
>>> b
3
>>> c
9
```



Εικόνα 9 – Ανάθεση τιμών σε πολλές μεταβλητές ταυτόχρονα.

2.4 Μιγαδικοί αριθμοί.

Πολλές γλώσσες προγραμματισμού δεν υποστηρίζουν τους μιγαδικούς αριθμούς, αλλά στην python υπάρχει αυτή η δυνατότητα. Ένας μιγαδικός αριθμός ορίζεται, χωρίζοντάς

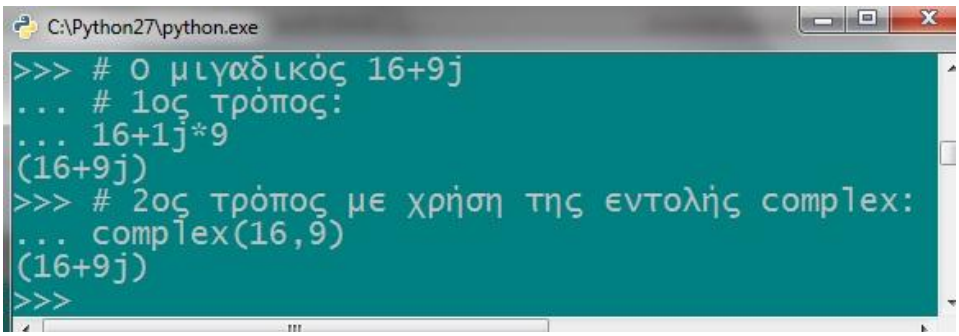
[5] Greektuts(2009), Μαθαίνοντας python – Μέρος 2, Ανάκτηση στις 17-4-2011 από http://greektuts.net/python-part-2

το πραγματικό και φανταστικό του μέρος. Έτσι, μπορεί να υπολογιστεί το μέτρο του. Επιπλέον, δίνεται η δυνατότητα για πράξεις μεταξύ μιγαδικών αριθμών.

2.4.1 *Τρόπος δήλωσης ενός μιγαδικού.*

Οι φανταστικοί αριθμοί δηλώνονται με 'j' ή 'J'. Όταν το πραγματικό μέρος του μιγαδικού, δεν είναι μηδέν, τότε γράφεται ως (real+imagj). Μπορεί να δημιουργηθεί και αμέσως, με τη λειτουργία complex(real, imag).

```
>>> # Ο μιγαδικός 16+9j
... # 1ος τρόπος:
... 16+1j*9
(16+9j)
>>> # 2ος τρόπος με χρήση της εντολής complex:
... complex(16,9)
(16+9j)
```



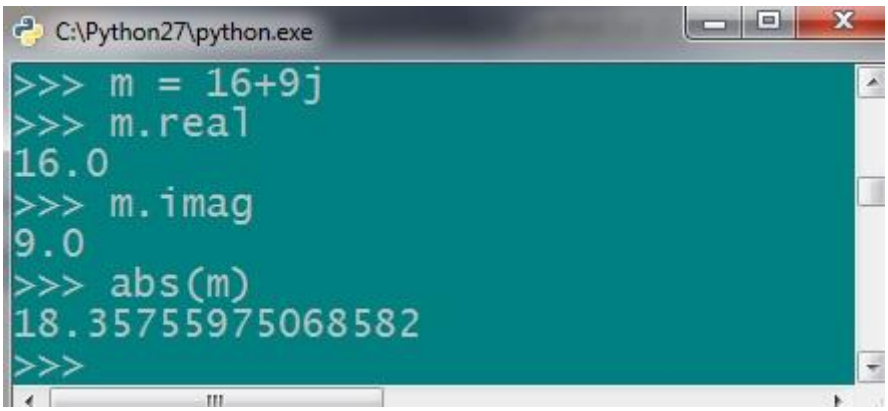
Εικόνα 10 – Παράδειγμα δήλωσης μιγαδικού.

Για να χωριστεί ο μιγαδικός σε πραγματικό και φανταστικό μέρος, γίνεται η ακόλουθη διαδικασία. Αν ο μιγαδικός είναι ο m, τότε m.real είναι το πραγματικό μέρος και το m.imag, είναι το φανταστικό μέρος του μιγαδικού.

```
>>> m=16+9j
>>> m.real
16.0
>>> m.imag
9.0
```

Με την εντολή abs(m) υπολογίζεται το μέτρο του μιγαδικού αριθμού.

```
>>> abs(m)
18.357559750685819
```



Εικόνα 11 – Διαχωρισμός του μιγαδικού και υπολογισμός του μέτρου του.

2.5 Συναρτήσεις.

Για να λυθεί ένα μεγάλο πρόβλημα, πρέπει πρώτα να αναλυθεί σε μια σειρά από απλούστερα προβλήματα. Η python για την επίλυση μικρών προβλημάτων, παρέχει στον προγραμματιστή μια σειρά από συναρτήσεις, μέσω της πλούσιας βιβλιοθήκης της. Οι συναρτήσεις είναι διάφορα υποπρογράμματα, τα οποία καλούνται μέσα στο κυρίως πρόγραμμα, για να υλοποιηθεί μια συγκεκριμένη διαδικασία.

Μια συνάρτηση, καλείται με το όνομα της για να εφαρμοστεί και μέσα σε παρένθεση μπαίνουν οι τιμές των παραμέτρων της. Οι συναρτήσεις μπορούν να χρησιμοποιηθούν σε εκφράσεις.

2.5.1 Παραδείγματα συναρτήσεων.

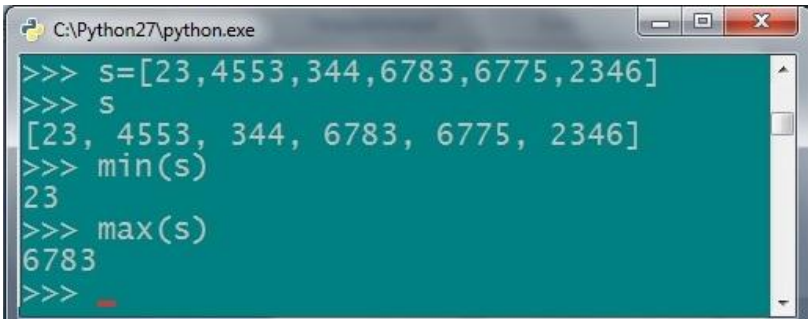
Ένα απλό παράδειγμα συνάρτησης, είναι η range(). Με αυτή τη συνάρτηση δημιουργείται μια λίστα αριθμών.

```
>>> range(8)
[0, 1, 2, 3, 4, 5, 6, 7]
```

Εμφανίζεται μια λίστα με 8 τιμές, από το 0 μέχρι το 7.

Η λίστα μπορεί να μην ξεκινάει από το 0 και έχει βήμα μεγαλύτερο από το 1. Στο παρακάτω παράδειγμα, το 4 είναι η αρχή της λίστας, το 80 είναι το τέλος της και το 15 είναι το βήμα που κινείται. Ενδεχομένως, να έχει και αρνητικές τιμές, με αρνητικό βήμα.

```
>>> range(4,80,15)
[4, 19, 34, 49, 64, 79]
>>> range(5,-35,-10)
[5, -5, -15, -25]
```



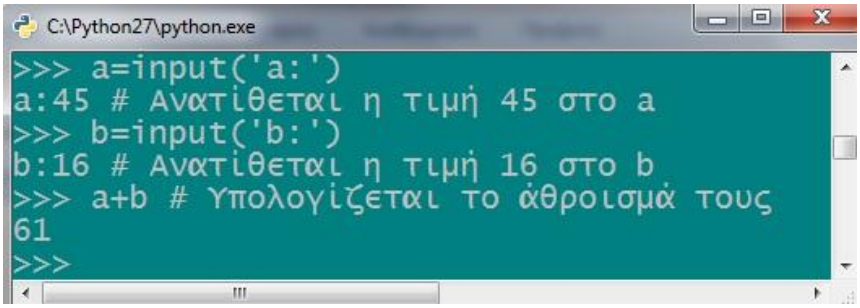
Εικόνα 12 – Παράδειγμα με τη συνάρτηση range.

Η συνάρτηση pow(), υψώνει έναν αριθμό σε μια δύναμη:

```
>>> pow(6,2)
36
```

Μια άλλη χρήσιμη συνάρτηση είναι η input(), η οποία διαβάζει μια τιμή από το πληκτρολόγιο του χρήστη και να την αναθέτει σε κάποια μεταβλητή.

```
>>> a=input('a:')
a:45 # Ανατίθεται η τιμή 45 στο a
>>> b=input('b:')
b:16 # Ανατίθεται η τιμή 16 στο b
>>> a+b # Υπολογίζεται το άθροισμά τους
61
```



Εικόνα 13 – Παράδειγμα ανάθεσης τιμής με το input.

Η Python διαθέτει αρκετές ενσωματωμένες συναρτήσεις και φυσικά μπορεί ο προγραμματιστής να ορίσει τις δικές του.

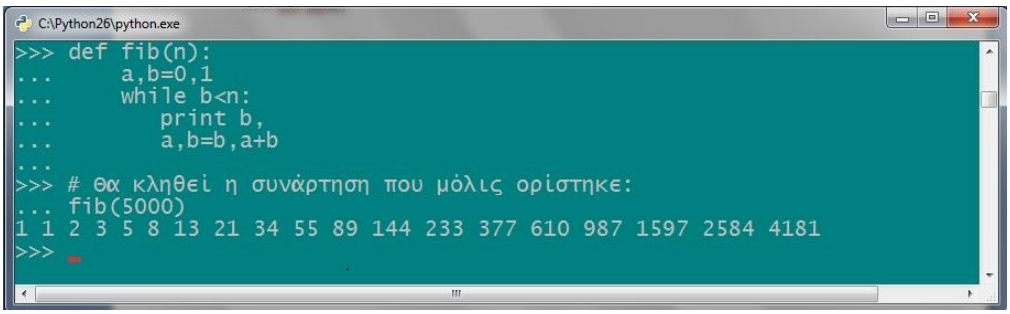
2.5.2 Ορίζοντας μια συνάρτηση.

Για να οριστεί μια νέα συνάρτηση πρέπει να μπει το def ακολουθούμενο από το όνομα που θα έχει η συνάρτηση. Θα δημιουργηθεί μια συνάρτηση η οποία θα υπολογίζει την ακολουθία Fibonacci για n αριθμούς. Η ακολουθία Fibonacci παίρνει δύο αριθμούς, βρίσκει το άθροισμα τους και στη συνέχεια τους προσθέτει και το αποτέλεσμα, μπαίνει στο τέλος της ακολουθίας. Θα προσθέτει τους δύο τελευταίους αριθμούς της ακολουθίας, μέχρι να φτάσει στον αριθμό, που ορίστηκε κατά την κλήση της συνάρτησης. Στη διαδικασία του υπολογισμού της ακολουθίας Fibonacci, θα χρησιμοποιηθεί η δομή επανάληψης που θα αναλυθεί στο κεφάλαιο των δομών ελέγχου.

```
>>> def fib(n):
...         a,b=0,1
...         while b<n:
...             print b,
...             a,b=b,a+b
...
```

Αφού οριστεί η συνάρτηση, θα κληθεί για τον υπολογισμό της ακολουθίας Fibonacci μέχρι το 5000.

```
>>> # Θα κληθεί η συνάρτηση που μόλις ορίστηκε:
... fib(5000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```



Εικόνα 14 – Παράδειγμα ορισμού συνάρτησης.

2.6 Modules

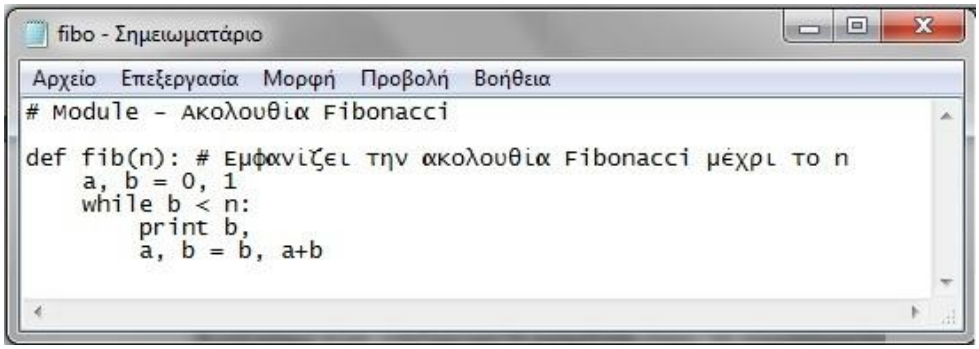
Όταν κλείσει ο διερμηνευτής της python, χάνονται οι μεταβλητές που έχουν χρησιμοποιηθεί και όλες οι συναρτήσεις που ορίστηκαν. Για να ξαναχρησιμοποιηθεί μια συνάρτηση που είχε οριστεί, πρέπει να ξαναγίνει η διαδικασία από την αρχή.

Για την εξοικονόμηση χρόνου, μπορεί να δημιουργηθεί ένα νέο module που θα περιλαμβάνει όλες τις διαδικασίες, για την εκτέλεση μιας λειτουργίας. Τα modules είναι κάτι σαν επεκτάσεις που εισάγονται στην Python, για να επεκτείνονται οι δυνατότητές της. Ένα module, πρέπει να γράφεται σε επεξεργαστή κειμένου και να αποθηκεύεται με συγκεκριμένο όνομα, που θα χρησιμοποιείται κατά την κλήση του. Είναι όπως όταν δημιουργείται ένα script. Ένα module μπορεί να αποτελείται και από δευτερεύοντα modules, που εκτελούν συγκεκριμένες διαδικασίες.

Όλα τα modules πρέπει να τελειώνουν σε κατάληξη .py και το όνομα τους να αποτελείται από χαρακτήρες. Κάθε module, είναι διαθέσιμο ως παγκόσμια μεταβλητή __name__.

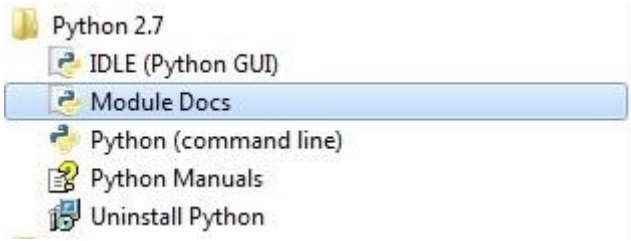
Σαν εφαρμογή των παραπάνω, θα κατασκευαστεί ένα module με την ακολουθία Fibonacci. Σε έναν επεξεργαστή κειμένου, όπως το σημειωματάριο, ορίζεται η ακολουθία και αποθηκεύεται το module ως fibo.py.

```
# Module - Ακολουθία Fibonacci
def fib(n): # Εμφανίζει την ακολουθία Fibonacci μέχρι το n
   a, b = 0, 1
   while b < n:
      print b,
      a, b = b, a+b
```



Εικόνα 15 – Δημιουργία module στο σημειωματάριο.

Το παραπάνω module αποθηκεύεται στην βιβλιοθήκη της python, μαζί με τα υπόλοιπα modules. Η βασική βιβλιοθήκη της Python διαθέτει πάρα πολλά, που αφορούν κάθε είδους λειτουργία. Υπάρχουν ακόμα περισσότερα modules που μπορούν να εγκατασταθούν, άλλων προγραμματιστών και σε διάφορες on-line πηγές. Η πρόσβαση στη βιβλιοθήκη γίνεται από το εικονίδιο module docs του φακέλου της python.



Εικόνα 16 – Μετάβαση στη βιβλιοθήκη της python.

Το module που δημιουργείται, εισάγεται κατευθείαν από τον διερμηνευτή της python, χρησιμοποιώντας την εντολή import και στη συνέχεια καλείται η συνάρτηση ως εξής:

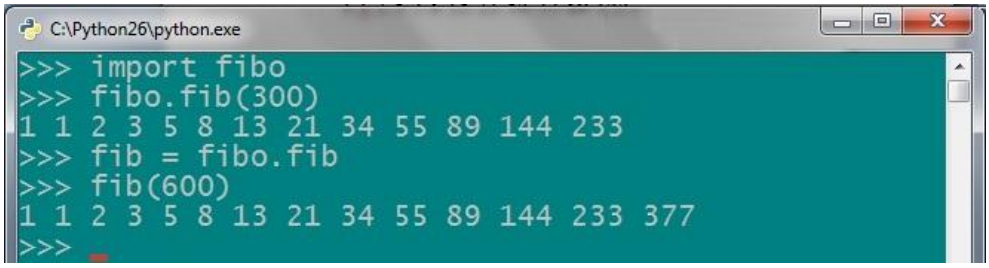
```
>>> import fibo
```

Βάζοντας μπροστά τη λέξη fibo, καλείται η συνάρτηση που ορίστηκε στο module. Αυτό συμβαίνει, επειδή η fib είναι συνάρτηση του module fibo.

```
>>> fibo.fib(300)
1 1 2 3 5 8 13 21 34 55 89 144 233
```

Υπάρχει η δυνατότητα να εκχωρηθεί η συνάρτηση σε μια νέα μεταβλητή, όπως γίνεται στη συνέχεια.

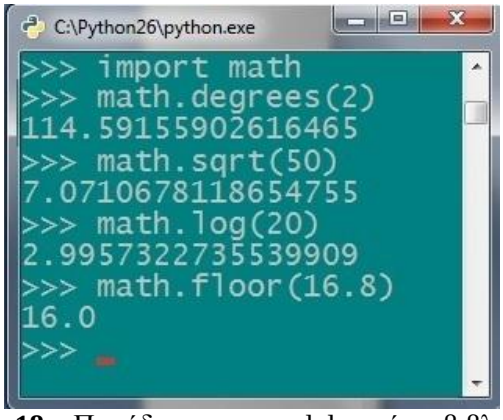
```
>>> fib = fibo.fib
>>> fib(600)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```



Εικόνα 17 - Εισαγωγή του module.

Όπως αναφέρθηκε, η python έχει στην βιβλιοθήκη της πολλά modules, όπως για παράδειγμα το module math, που διαθέτει πολλές χρήσιμες συναρτήσεις. Μερικές από αυτές είναι η degrees, που μετατρέπει τα ακτίνια σε μοίρες, η sqrt που βρίσκει την τετραγωνική ρίζα ενός αριθμού και η log, που βρίσκει το λογάριθμο. Υπάρχει τέλος και η floor, η οποία θα δεχθεί έναν αριθμό και θα επιστρέψει, τη μικρότερη δυνατή ακέραια τιμή.

```
>>> import math
>>> math.degrees(2)
114.59155902616465
>>> math.sqrt(50)
7.0710678118654755
>>> math.log(20)
2.9957322735539909
>>> math.floor(16.8)
16.0
```

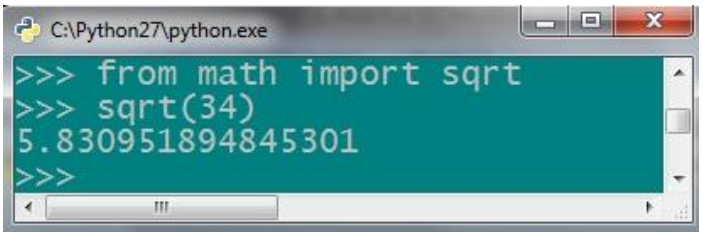


Εικόνα 18 – Παράδειγμα με module από τη βιβλιοθήκη.

Είναι δυνατή επίσης, η εισαγωγή μόνο μιας συνάρτησης, από το module. Μετά την εισαγωγή της συνάρτησης, θα καλείται άμεσα και χωρίς τη μεσολάβηση του module. Ενδεικτικό είναι το ακόλουθο παράδειγμα με τη συνάρτηση της τετραγωνικής ρίζας:

```
>>> from math import sqrt
>>> sqrt(34)
```

5.8309518948453007



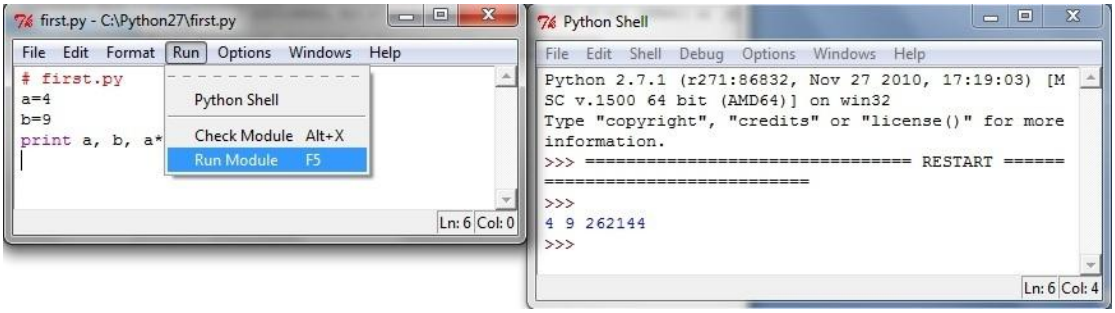
Εικόνα 19 - Εισαγωγή μιας συνάρτησης από το module.

2.7 Αποθήκευση και εκτέλεση των προγραμμάτων.

Παρουσιάστηκε πιο πριν, ο τρόπος για να δημιουργηθεί ένα νέο module στην python. Όταν ένα πρόγραμμα είναι αποθηκευμένο στον υπολογιστή του προγραμματιστή, δεν ελλοχεύει ο κίνδυνος να χαθεί μόλις κλείσει ο διερμηνευτής.

Για τη δημιουργία ενός προγράμματος, πρέπει να ανοίξει ο προγραμματιστής ένα νέο παράθυρο, μέσα στο διαδραστικό κέλυφος της python – IDLE (Python GUI). Αυτό πετυχαίνεται, πατώντας file→ New Window. Στο παράθυρο που εμφανίζεται, συντάσσεται ο κώδικας και αφού αποθηκευτεί, πατώντας το run, τρέχει το module (βλ. εικόνα 20). Για παράδειγμα, τοποθετήθηκαν οι παρακάτω εντολές και αποθηκεύτηκε το αρχείο ως first.py. Το αρχείο θα αποθηκευθεί στο φάκελο της εγκατάστασης. Έτσι, έχει δημιουργηθεί ένα πρόγραμμα στην python.

```
# first.py
a=4
b=9
print a,b,a**b
```

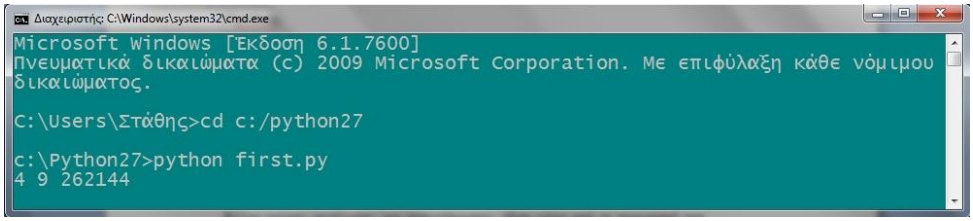


Εικόνα 20 – Δημιουργία προγράμματος.

Ένα πρόγραμμα μπορεί να εκτελεστεί και μέσα από το τερματικό (cmd) του λειτουργικού συστήματος, αρκεί να προσδιοριστεί ο φάκελος που είναι αποθηκευμένο. Ανοίγοντας τη γραμμή εντολών των windows, εκτελείται η ακόλουθη εντολή για να προσδιοριστεί ο φάκελος:

cd c:/python27. Αφού προσδιοριστεί ο φάκελος που είναι αποθηκευμένο, εκτελείται το πρόγραμμα.

```
c:\Python27>python first.py
4 9 262144
```



Εικόνα 21 – Εκτέλεση προγράμματος από το cmd.

Μπορεί να παραλειφθεί η λέξη python και να μείνει μόνο το first.py, δεδομένου ότι ο τρέχων φάκελος είναι αυτός που το περιέχει.

Εικόνα 22 – Δεύτερη εκτέλεση από το cmd.

2.8 Άλλοι τελεστές.

Εκτός από τους αριθμητικούς τελεστές που αναφέρθηκαν μέχρι τώρα, υπάρχουν οι τελεστές σύγκρισης και οι λογικοί τελεστές, οι οποίοι παρατίθενται στον πίνακα που ακολουθεί:[6]

Πινάκας 1: Λογικοί τελεστές και τελεστές σύγκρισης.	
<, >, <= και >=	Μικρότερο, μεγαλύτερο, μικρότερο ή ίσο και μεγαλύτερο ή ίσο.
a == b	Η τιμή του a να ισοδυναμεί με την τιμή του b.
a != b	Η τιμή του a να είναι διαφορετική από αυτή του b.
not	Λογικό όχι - Να μην ισχύει η συνθήκη. Αν ισχύει, θα επιστρέψει το false και αν δεν ισχύει η συνθήκη, θα επιστρέψει true.
and	Λογικό και
or	Λογικό ή

2.9 Ασκήσεις στα βασικά.

Α. Με την βοήθεια της γλωσσάς προγραμματισμού python να υπολογισθούν οι παρακάτω αριθμητικές πράξεις:

- 3+1
- 20-5
- 3*7
- 12/4

[6] Swaroop CH, «*A Byte of Python*» (Ελληνική μετάφραση), Τελεστές και εκφράσεις, Ανάκτηση στις 25-4-2011 απόHttp://www.swaroopch.org/notes/Python_el:%CE%A0%CE%B5%CF%81%CE%B9%CE%B5%CF%87%CF%8C%CE%BC%CE%B5%CE%BD%CE%B1

- 9.4/2
- 8/5
- 10**2
- 1/2
- 5.0/9.0
- 7%5

B. Ομοίως και οι παρακάτω αριθμητικές παραστάσεις:
- (345-50)/4+5*4-7
- (5**2)/(2*3)
- (350-240)*3)/5**2

C. Να αποδειχθεί το πυθαγόρειο θεώρημα σε ένα τρίγωνο. Οι 2 κάθετες πλευρές του έχουν μήκη 5 και 12, ενώ η υποτείνουσα έχει μήκος 13.

D. Να δημιουργηθούν 2 μιγαδικοί αριθμοί, να χωρισθούν σε πραγματικό και φανταστικό μέρος και να βρεθούν τα μετρά τους.

E. Να δημιουργηθεί και να εμφανιστεί μια λίστα. Στη συνέχεια να δημιουργηθούν άλλες δύο λίστες που θα χρησιμοποιούν σταθερό βήμα για την εμφάνιση των στοιχείων. Η τελευταία λίστα θα έχει αρνητικό βήμα.

F. Παρακάτω δίνονται οι μέρες που πήραν άδεια 5 υπάλληλοι τον προηγούμενο χρόνο. Να ανατεθούν οι τιμές από το χρήστη και να υπολογισθεί ο μέσος όρος των ημερών που πήραν άδεια οι υπάλληλοι. Α:17, Β:20, Γ:11, Δ:19 και Ε:23

G. Να υπολογισθούν τα παρακάτω:
a. Η τετραγωνική ρίζα του 25.
b. Ο λογάριθμος του 10.
c. Τα 2,5 ακτίνια, σε πόσες μοίρες αντιστοιχούν.
d. Να βρεθεί η μικρότερη δυνατή ακέραια τιμή, για τον αριθμό 17,8.

Οι λύσεις των ασκήσεων βρίσκονται στο παράρτημα.

ΚΕΦΑΛΑΙΟ 3^{ο}
Δομές δεδομένων.

Οι δομές δεδομένων υποστηρίζονται από την python και είναι ακολουθίες, που περιέχουν συλλογές στοιχείων δεδομένων και συνεισφέρουν στην ταχύτερη διεκπεραίωση των απαιτούμενων υπολογισμών ενός προγράμματος. Η βασική βιβλιοθήκη της python περιέχει μια πλούσια συλλογή από δομές δεδομένων. Στη συνέχεια θα αναλυθούν οι βασικότερες δομές δεδομένων που συναντώνται σε ένα πρόγραμμα.

3.1 Οι σημαντικότερες λειτουργίες των ακολουθιών.

Υπάρχουν μερικές λειτουργίες που εφαρμόζονται σε όλα τα ειδή των ακολουθιών όπως:

- Ευρετηρίαση στοιχείου (indexing)
- Τεμαχισμός (Slicing)
- Πρόσθεση στοιχείου (adding)
- Πολλαπλασιασμός ακολουθίας (multiplying)
- Έλεγχος μέλους (Membership)
- Μήκος ακολουθίας
- Μέγιστο και Μικρότερο στοιχείο

Για την ώρα θα γίνει μία απλή αναφορά και θα αναλυθούν μαζί με τις δομές.

- Indexing

Σε κάθε στοιχείο της ακολουθίας αντιστοιχεί και ένας αριθμός ευρετηρίου (index). Η αρίθμηση στο ευρετήριο ξεκινάει πάντα από 0, το δεύτερο στοιχείο έχει το 1 και ούτω καθεξής. Επίσης, είναι δυνατή η αντίστροφη αρίθμηση από το τέλος της ακολουθίας προς την αρχή της, δηλαδή το τελευταίο στοιχείο να ξεκινάει με τον αριθμό ευρετηρίου -1, το επόμενο το -2 και λοιπά.

- Slicing

Με τον τεμαχισμό (slicing), κόβονται τα στοιχεία μιας ακολουθίας χρησιμοποιώντας δύο δείκτες χωρισμένους με άνω κάτω τελεία.

```
>>> a=[1,2,3,4,5,6,7,8,9]
>>> a[3:6]
[4, 5, 6]
```

Το slicing είναι ιδανικό για να αποσπασθούν συγκεκριμένα κομμάτια από μια ακολουθία. Ο πρώτος δείκτης είναι ο αριθμός ευρετηρίου του στοιχείου που θα είναι πρώτο στη νέα λίστα, ενώ ο δεύτερος δείχνει το τέλος, χωρίς όμως αυτός να περιλαμβάνεται στη νέα λίστα.

- Adding

Με την πρόσθεση (adding) συνενώνονται ακολουθίες, χρησιμοποιώντας τον τελεστή της πρόσθεσης.

```
>>> [1,2,3,4]+[5,6,7]+[8,9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Multiplication

Ο πολλαπλασιασμός (multiplication) δημιουργεί μία καινούρια ακολουθία. Η νέα ακολουθία θα είναι η αρχική επαναλαμβανόμενη κατά n φορές, όπου n θα είναι η τιμή με την οποία πολλαπλασιάστηκε την ακολουθία.

```
>>> b='Patra'
>>> b
'Patra'
>>> b*4
'PatraPatraPatraPatra'
```

- Membership

Για να εντοπιστεί μία τιμή σε μια ακολουθία, χρησιμοποιείται ο τελεστής ‘in’. Επίσης με το ‘not in’ θα ελεγχθεί αν η τιμή δεν υπάρχει στην ακολουθία. Οι τελεστές αυτοί ονομάζονται Boolean και επιστρέφουν την ένδειξη True ή False, ανάλογα αν υπάρχει ή όχι η τιμή στη λίστα.

```
>>> 6 in a
True
>>> 10 not in a
True
>>> 10 in a
False
>>> 7 not in a
False
```

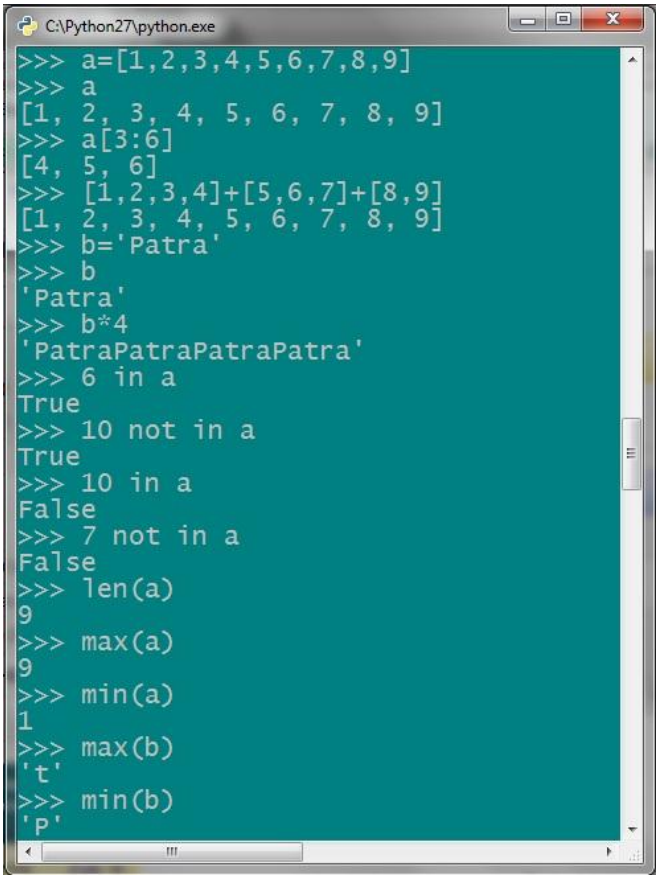
- Μήκος ακολουθίας, μέγιστο και ελάχιστο.

Με την βοήθεια των ενσωματωμένων συναρτήσεων που διαθέτει η Python, ενημερώνεται ο χρήστης για το μήκος μιας ακολουθίας, το μεγαλύτερο στοιχείο της ή το μικρότερο. Οι ενσωματωμένες συναρτήσεις της Python είναι οι len, max και min αντίστοιχα, αποτελούν αρκετά χρήσιμα εργαλεία στις ακολουθίες:

```
>>> len(a)
9
>>> max(a)
9
>>> min(a)
1
```

Τα min και max, μπορούν να εφαρμοστούν και σε αλφαριθμητικά, αλλά θα βρεθεί το μέγιστο και το ελάχιστο αφού μετατραπούν οι χαρακτήρες σε αρίθμηση ASCII.

```
>>> max(b)
't'
>>> min(b)
'P'
```



Εικόνα 23 – Εφαρμογή των len, max και min.

Οι βασικότερες από τις δομές δεδομένων της python είναι οι εξής:

1) Αλφαριθμητικά (strings) και Unicode Αλφαριθμητικά (Unicode Strings)
2) Λίστες
3) Tuples (πλειάδες)
4) Λεξικά
5) Σύνολα (Sets)

Υπάρχουν και άλλες δευτερεύοντες δομές, όπως τα xrange αντικείμενα και οι buffers που χρησιμοποιούνται σπανιότερα.

3.2 Αλφαριθμητικά.

Τα αλφαριθμητικά (strings) μπαίνουν πάντα σε αποστρόφους και δεν υπάρχει η δυνατότητα να μεταβληθούν, αλλά μόνο να δημιουργηθούν καινούρια. Τα αλφαριθμητικά είναι σε κωδικοποίηση UTF-8.

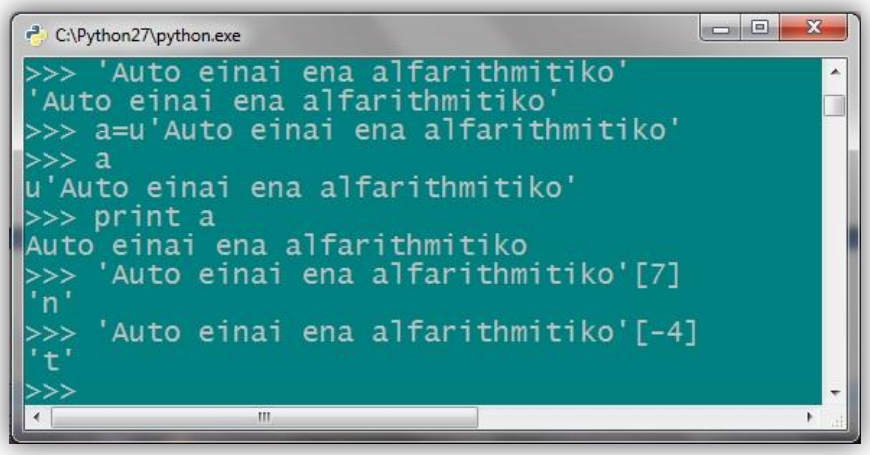
```
>>> 'Auto einai ena alfarithmitiko'
'Auto einai ena alfarithmitiko'
```

Υπάρχουν και τα unicode αλφαριθμητικά, που είναι οι αλφαριθμητικοί χαρακτήρες και μετατρέπονται σε μορφή ASCII. Αφορά μόνο κυριλλικές γλώσσες όπως τα Ρώσικα κλπ.).

```
>>> a=u'Auto einai ena alfarithmitiko'
>>> a
u'Auto einai ena alfarithmitiko'
>>> print a
Auto einai ena alfarithmitiko
```

Τα strings βέβαια αποτελούν ακολουθίες και αυτό σημαίνει ότι είναι προσπελάσιμα τα στοιχεία τους, ακόμα και χωρίς τη χρήση κάποιας μεταβλητής:

```
>>> 'Auto einai ena alfarithmitiko'[7]
'n'
>>> 'Auto einai ena alfarithmitiko'[-4]
't'
```



Εικόνα 24 – Αλφαριθμητικά.

3.3 Λίστες

Η λίστα είναι μια χρήσιμη δομή δεδομένων της python και βασικό χαρακτηριστικό της είναι, ότι μπορεί να μεταβληθεί το περιεχόμενο της. Στα αλφαριθμητικά και τα tuples, δεν συμβαίνει αυτό. Είναι μια συλλογή από ομαδοποιημένες τιμές, βρίσκονται δηλαδή μέσα σε μια κατάσταση (λίστα) και χωρίζονται με κόμμα. Η λίστα περιβάλλεται από αγκύλες και τα στοιχεία της λίστας δεν είναι απαραίτητο να είναι όλα του ίδιου τύπου αλλά μπορεί να είναι strings, αριθμοί κλπ. Τα αλφαριθμητικά δηλώνονται μέσα σε αποστρόφους για να είναι δυνατή η διαχείριση τους. Υπάρχει και η συνάρτηση list(), που δημιουργεί μία λίστα από τα στοιχεία του string. Ένα παράδειγμα για τις λίστες:

```
>>> lista=['Patra','Kalamata',64,327]
>>> lista
['Patra', 'Kalamata', 64, 327]
```

3.3.1 Λειτουργίες στις λίστες.

Κάθε στοιχείο της λίστας έχει για όνομα, το όνομα της λίστας και την αντίστοιχη θέση (index). Η θέση του πρώτου στοιχείου έχει την τιμή 0.

```
>>> lista[0]
'Patra'
>>> lista[3]
327
```

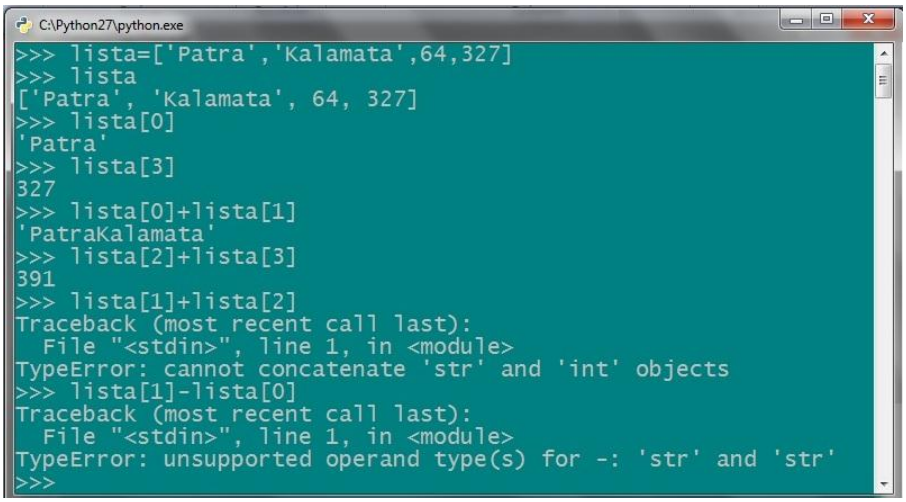
Τα στοιχεία μπορεί να είναι διαφορετικού τύπου το ένα από το άλλο. Η python αντιμετωπίζει κάθε στοιχείο της λίστας, ανάλογα με το τον τύπο του.

```
>>> lista[0]+lista[1]
'PatraKalamata'
>>> lista[2]+lista[3]
391
```

Δεν γίνονται πράξεις (όπως πρόσθεση και αφαίρεση) με χαρακτήρες και αριθμούς, σε τέτοια περίπτωση θα εμφανίσει σφάλμα.

```
>>> lista[1]+lista[2]
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> lista[1]-lista[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

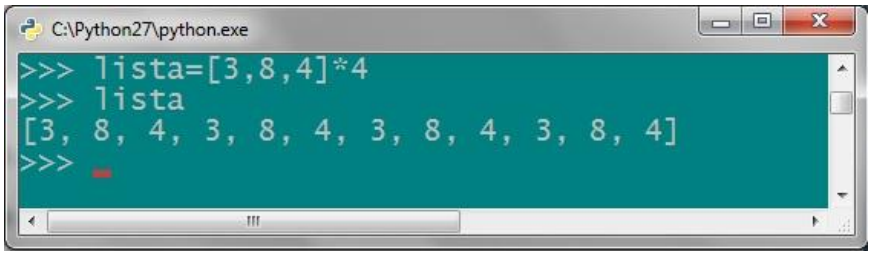


Εικόνα 25 – Εντολές στις λίστες.

Κλείνοντας το κομμάτι με τους αριθμητικούς τελεστές, υπάρχει και το επί (*), που χρησιμοποιείται για την επανάληψη των στοιχείων. Για παράδειγμα:

```
>>> lista=[3,8,4]*4
>>> lista
[3, 8, 4, 3, 8, 4, 3, 8, 4, 3, 8, 4]
```

Το lista=[3,8,4]*4 ισοδυναμεί με το lista=[3,8,4] + [3,8,4] + [3,8,4] + [3,8,4] και συνδέει τις παραπάνω λίστες σε μία.



Εικόνα 26 – Επανάληψη λίστας.

Μπορούν να χρησιμοποιηθούν και αρνητικοί δείκτες, για να εντοπισθεί ένα στοιχείο της λίστας. Σε αυτήν την περίπτωση μετράει από το τέλος της λίστας προς την αρχή.

```
>>> lista[-1]
327
>>> lista[-3]
'Kalamata'
```

Με τη συνάρτηση in, ελέγχει αν υπάρχει ένα στοιχείο στη λίστα. Το not in αντίστοιχα, ενημερώνει αν το στοιχείο δεν υπάρχει στη λίστα.

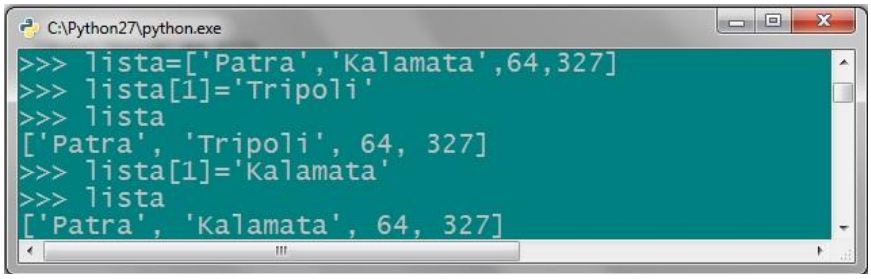
```
>>> 34 in lista
False
>>> 327 in lista
True
```

```
>>> 100 not in lista
True
```

- Μεταβολή μιας λίστας.

Η μεταβολή των στοιχείων μιας λίστας, είναι πολύ απλή διαδικασία και μοιάζει με εκείνη της ανάθεσης τιμής σε μεταβλητή. Όμως εδώ, χρησιμοποιείται και ο δείκτης ευρετηρίου, για να δηλώσει ο χρήστης ποιο από τα στοιχεία της λίστας επιθυμεί να μεταβάλλει:

```
>>> lista=['Patra','Kalamata',64,327]
>>> lista[1]='Tripoli'
>>> lista
['Patra', 'Tripoli', 64, 327]
>>> lista[1]='Kalamata'
>>> lista
['Patra', 'Kalamata', 64, 327]
```



Εικόνα 27 – Μεταβολή στοιχείων μιας λίστας.

- Τεμαχισμός μιας λίστας (slicing).

Για να κοπεί ένα κομμάτι της λίστας, χρησιμοποιείται η παρακάτω εντολή:

```
>>> lista[1:3]
['Kalamata', 64]
```

Το lista[1:3] σημαίνει, πως η νέα λίστα αποτελείται από το στοιχείο lista[1] και όλα τα επόμενα μέχρι το lista[3], χωρίς όμως να συμπεριλαμβάνεται το συγκεκριμένο στοιχείο. Ο πρώτος αριθμός, θα δηλώνει πάντα τη θέση ευρετηρίασης (index) του στοιχείου, με το οποίο αρχίζει το νέο slice και ο δεύτερος δείχνει το τέλος του κομματιού, δηλαδή μέχρι που θα φτάσει το slice χωρίς όμως να το περιέχει. Υπάρχει η δυνατότητα, να χρησιμοποιηθούν και αρνητικοί αριθμοί χωρίς να δημιουργείται πρόβλημα.

```
>>> lista[2:-1]
[64]
>>> lista[1:-1]
['Kalamata', 64]
```

Μπορεί επίσης να παραλειφθεί ο πρώτος αριθμός, όταν εννοείται η αρχή της λίστας, δηλαδή τον αριθμό 0.

```
>>> lista[:2]
['Patra', 'Kalamata']
```

Για να συμπεριληφθεί και το τελευταίο στοιχείο της λίστας στο κομμάτι, δεν θα μπει δεύτερος αριθμός, ή θα δηλωθεί μεγαλύτερος αριθμός, πχ το 4 (δεν υπάρχει στοιχείο σε αυτή τη θέση της λίστας). Σε αμφότερες τις περιπτώσεις, το αποτέλεσμα θα είναι το ίδιο.

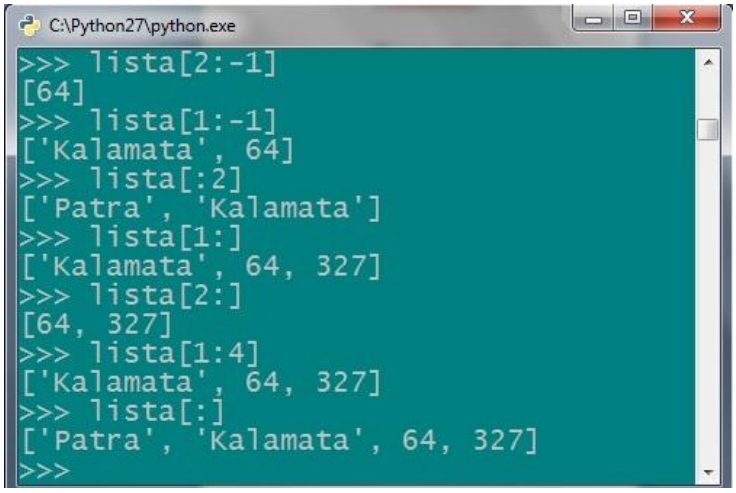
```
>>> lista[1:]
['Kalamata', 64, 327]
>>> lista[2:]
[64, 327]
```

```
>>> lista[1:4]
['Kalamata', 64, 327]
```

Τέλος, με τον παρακάτω τρόπο περιλαμβάνεται ολόκληρη η λίστα στο κομμάτι:

```
>>> lista[:]
['Patra', 'Kalamata', 64, 327]
```

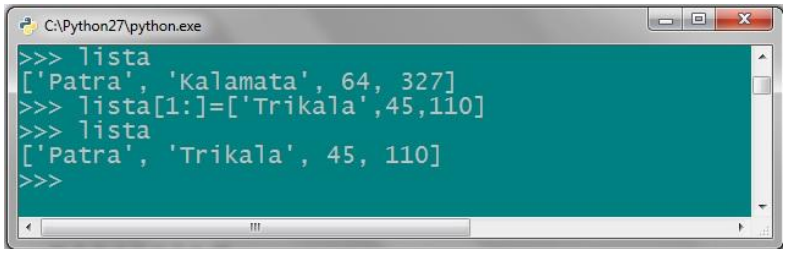
Το lista[:] είναι μια νέα λίστα, που απλά τυχαίνει να έχει τα ίδια στοιχεία με την πρωταρχική λίστα. Έτσι, δημιουργείται ένα αντίγραφο της λίστας.



Εικόνα 28 – Τεμαχισμός λίστας.

Αυτή η μέθοδος επίσης, δίνει τη δυνατότητα να μεταβληθεί ένα συγκεκριμένο κομμάτι της λίστας:

```
>>> lista
['Patra', 'Kalamata', 64, 327]
>>> lista[1:]=['Trikala',45,110]
>>> lista
['Patra', 'Trikala', 45, 110]
```



Εικόνα 29 – Μεταβολή στοιχείων με τεμαχισμό.

Κλείνοντας με τον τεμαχισμό, η χρήση ενός τρίτου δείκτη θα αντιστοιχεί στο βήμα, που θα τεμαχιστεί η λίστα.

```
>>> a=[1,2,3,4,5,6,7,8,9]
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a[1:6:2]
[2, 4, 6]
```

Εικόνα 30 – Τεμαχισμός με βήμα.

- Άλλες λειτουργίες στις λίστες.

Όπως αναφέρθηκε και στην εισαγωγή νωρίτερα, η συνάρτηση len() βρίσκει το μήκος της λίστας:

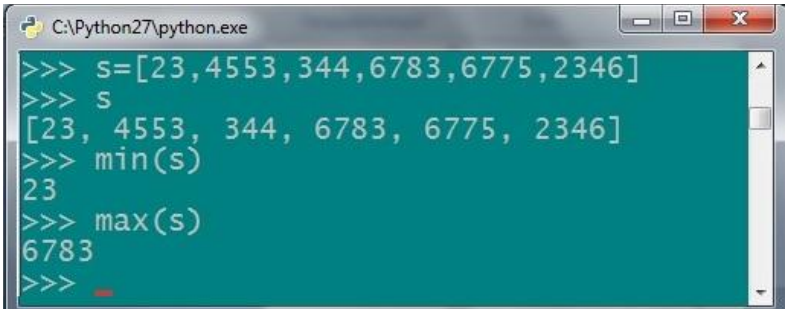
```
>>> len(lista)
4
>>> bpis='teipat'
>>> len(bpis)
6
```

Η εντολή list() μετατρέπει μια ακολουθία σε λίστα, είτε είναι αλφαριθμητική είτε οτιδήποτε άλλο.

```
>>> list(bpis)
['t', 'e', 'i', 'p', 'a', 't']
```

Με τα min() και max() επιστρέφεται αντίστοιχα, η μικρότερη και η μεγαλύτερη τιμή που υπάρχει σε μία λίστα.

```
>>> s=[23,4553,344,6783,6775,2346]
>>> s
[23, 4553, 344, 6783, 6775, 2346]
>>> min(s)
23
>>> max(s)
6783
```

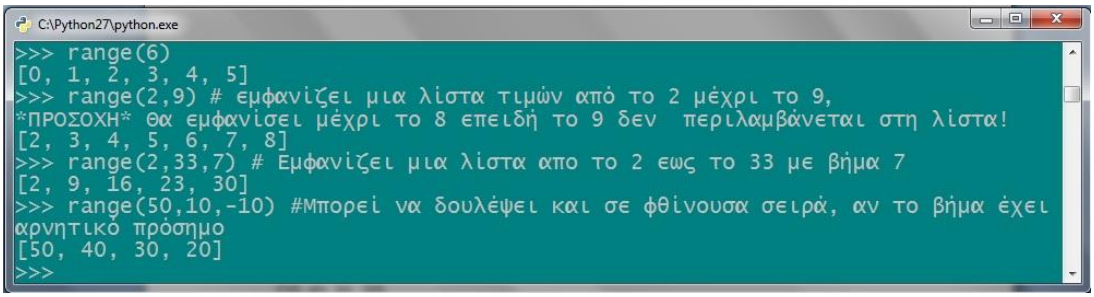


Εικόνα 31 – Διάφορες λειτουργίες στις λίστες.

Άλλη ενδιαφέρουσα συνάρτηση της python που αφορά τις λίστες είναι η range(), που επιστρέφει μια λίστα τιμών.

```
>>> range(6)
[0, 1, 2, 3, 4, 5]
>>> range(2,9) # εμφανίζει μια λίστα τιμών από το 2 μέχρι το 9.
*ΠΡΟΣΟΧΗ* Θα εμφανίσει μέχρι το 8 επειδή το 9 δεν  περιλαμβάνεται στη λίστα!
[2, 3, 4, 5, 6, 7, 8]
>>> range(2,33,7) # Εμφανίζει μια λίστα από το 2 έως το 33 με βήμα 7
[2, 9, 16, 23, 30]
```

>>> range(50,10,-10) #Μπορεί να δουλέψει και σε φθίνουσα σειρά, αν το βήμα έχει αρνητικό πρόσημο.
[50, 40, 30, 20]



Εικόνα 32 - Συνάρτηση range().

- Διαγραφή των στοιχείων μιας λίστας.

Για να διαγραφεί ένα στοιχείο ή μια ομάδα στοιχείων από μια λίστα, χρησιμοποιείται η εντολή del μαζί με τον δείκτη ευρετηρίου:

>>> a=range(4)
>>> a
[0, 1, 2, 3]
>>> del a[2:]
>>> a
[0, 1]

Εικόνα 33 – Διαγραφή στοιχείων από τη λίστα.

3.3.2 *Μέθοδοι που εκτελούνται στις λίστες.*

Οι μέθοδοι διαφέρουν από τις συναρτήσεις επειδή έχουν διαφορετικό τρόπο εκτέλεσης. Από τη μία, οι συναρτήσεις είναι στενά συνδεδεμένες με κάποιο αντικείμενο, όπως είναι μια λίστα, ένα αλφαριθμητικό ή οποιοδήποτε άλλο, όπως η max(a), που είναι η συνάρτηση max για το αντικείμενο a. Οι μέθοδοι από την άλλη πλευρά, καλούνται όπως οι συναρτήσεις αλλά μπαίνει μπροστά το όνομα του αντικειμένου και στη συνέχεια μετά από τελεία το όνομα της μεθόδου. Οι λίστες έχουν αρκετές μεθόδους που δίνουν τη δυνατότητα να εξεταστούν και να τροποποιηθούν τα στοιχεία τους. Μερικές αρκετά χρήσιμες από αυτές είναι:

- s.append() - Η μέθοδος append προσθέτει ένα καινούργιο στοιχείο στο τέλος της λίστας s.

>>> s =[23, 4553, 344, 6783, 6775, 2346]
>>> s
[23, 4553, 344, 6783, 6775, 2346]
>>> s.append(34)
>>> s
[23, 4553, 344, 6783, 6775, 2346, 34]

◦ s.count() - Μετράει πόσες φορές υπάρχει ένα στοιχείο σε μια λίστα.

```
>>> s.append(4553)
>>> s.count(4553)
2
>>> s.count(23)
1
```

◦ s.index() – Βρίσκει και επιστρέφει τον δείκτη, στον οποίο πρωτοεμφανίζεται ένα στοιχείο σε μια λίστα.

```
>>> s
[23, 4553, 344, 6783, 6775, 2346, 34, 4553]
>>> s.index(23)
0
>>> s.index(6775)
4
```

◦ s.remove() – Χρησιμοποιείται για την αφαίρεση της πρώτης εμφάνισης ενός αντικειμένου μιας λίστας:

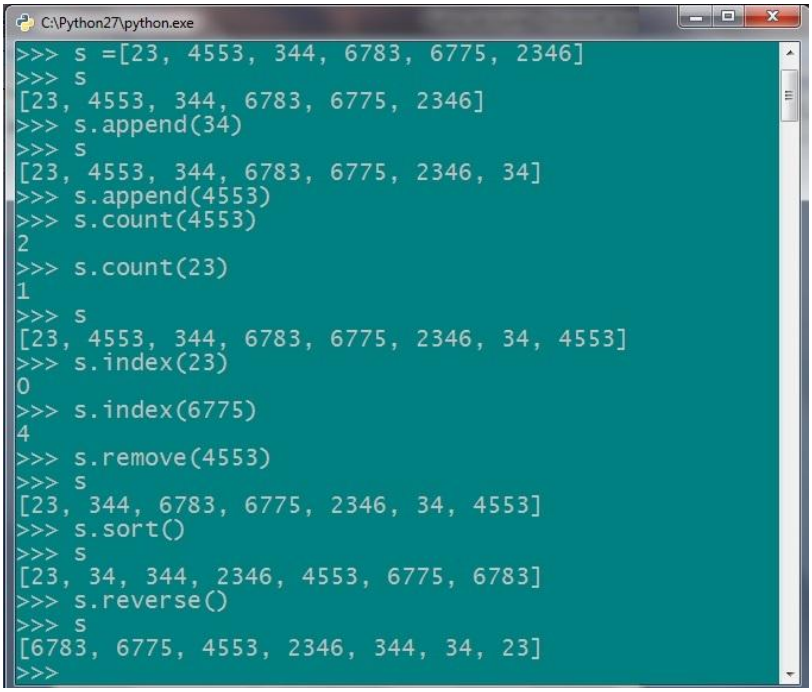
```
>>> s.remove(4553)
>>> s
[23, 344, 6783, 6775, 2346, 34, 4553]
```

◦ s.sort() - Η μέθοδος sort ταξινομεί τα στοιχεία μιας λίστας, μεταβάλλοντας την ίδια τη λίστα:

```
>>> s.sort()
>>> s
[23, 34, 344, 2346, 4553, 6775, 6783]
```

◦ s.reverse() – Η μέθοδος reverse αντιστρέφει τη σειρά εμφάνισης της λίστας.

```
>>> s.reverse()
>>> s
[6783, 6775, 4553, 2346, 344, 34, 23]
```



Εικόνα 34 – Μέθοδοι στις λίστες.

◦ s.insert() - Προσθέτει ένα καινούργιο στοιχείο σε συγκεκριμένη θέση της λίστας που ορίζει ο προγραμματιστής.

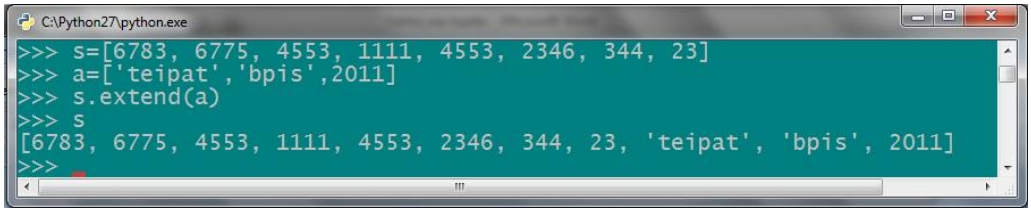
```
>>> s.insert(3,1111)
>>> s
[6783, 6775, 4553, 1111, 4553, 2346, 344, 23]
```

Προστέθηκε ο αριθμός 1111 στην τέταρτη θέση της λίστας, δηλαδή στο 3.

Εικόνα 35 – Μέθοδος insert.

◦ s.extend() - Προσθέτει στο τέλος της λίστας όσα στοιχεία υπάρχουν στην παρένθεση. Ουσιαστικά, επεκτείνεται η λίστα με στοιχεία από άλλη λίστα.

```
>>> s=[6783, 6775, 4553, 1111, 4553, 2346, 344, 23]
>>> a=['teipat','bpis',2011]
>>> s.extend(a)
>>> s
[6783, 6775, 4553, 1111, 4553, 2346, 344, 23, 'teipat', 'bpis', 2011]
```



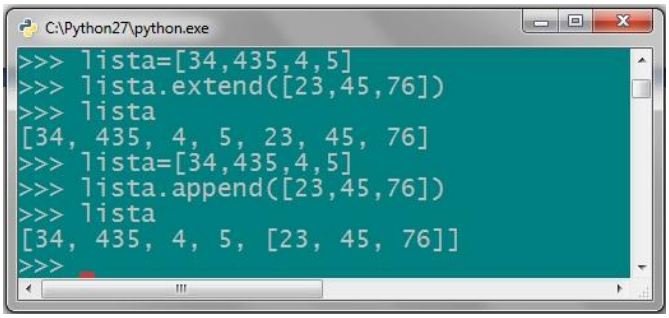
Εικόνα 36- Μέθοδος extend.

Υπάρχουν όμως, σημαντικές διαφορές μεταξύ των μεθόδων extend και append. Οι λίστες έχουν αυτές τις δύο μεθόδους οι οποίες φαινομενικά είναι ίδιες αλλά στην πραγματικότητα είναι τελείως διαφορετικές μεταξύ τους. Η μέθοδος extend όπως λέει και το όνομά της επεκτείνει την λίστα. Αν εξεταστεί για παράδειγμα μια λίστα με τέσσερα στοιχεία, με αυτή τη μέθοδο προστίθεται μια άλλη με περισσότερα στοιχεία, έστω τρία. Μετά την εκτέλεση της μεθόδου extend, θα είναι η αρχική μαζί με τη δεύτερη λίστα και ως εκ τούτου, η νέα λίστα θα αποτελείται από επτά στοιχεία.

```
>>> lista=[34,435,4,5]
>>> lista.extend([23,45,76])
>>> lista
[34, 435, 4, 5, 23, 45, 76]
```

Με την μέθοδο append από την άλλη μεριά, επισυνάπτεται η νέα λίστα στο τέλος της αρχικής λίστας. Στην περίπτωση δηλαδή, που η λίστα έχει τέσσερα στοιχεία και προστεθεί κάποια άλλη με τρία στοιχεία, τότε η λίστα θα έχει τελικά πέντε στοιχεία. Η λίστα που ενώθηκε με την αρχική, θα θεωρείται ως ένα στοιχείο ανεξάρτητα από το πόσα στοιχεία είχε.

```
>>> lista=[34,435,4,5]
>>> lista.append([23,45,76])
>>> lista
[34, 435, 4, 5, [23, 45, 76]]
```



Εικόνα 37: Διαφορά μεταξύ των extend και append.

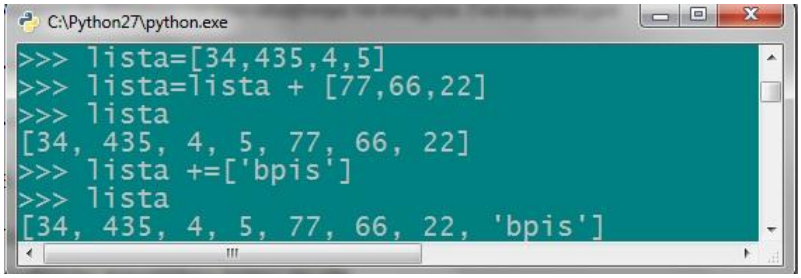
Μπορούν να χρησιμοποιηθούν και αριθμητικές εκφράσεις στις λίστες. Να εκτελεστεί η πρόσθεση αντί της εντολής extend: lista = lista + deuteri lista. Έχει την ίδια λειτουργία με το extend.

```
 >>> lista=lista + [77,66,22]
>>> lista
[34, 435, 4, 5, 77, 66, 22]
```

Επίσης η python υποστηρίζει και το +=.

```
>>> lista +=['bpis']
>>> lista
[34, 435, 4, 5, 77, 66, 22, 'bpis']
```

Το lista +=['bpis'] αντιστοιχεί στο lista.extend([‘bpis’]). Η έκφραση += μπορεί να χρησιμοποιηθεί σε λίστες, αλφαριθμητικά και ακέραιους. Τέλος, υπάρχει η δυνατότητα να φορτωθεί για να δουλέψει και σε κλάσεις που ορίζονται από το χρήστη της python (θα αναλυθεί σε παρακάτω κεφάλαιο).

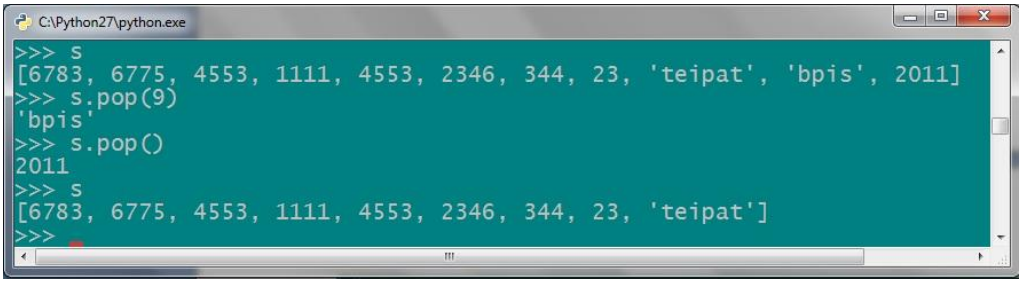


Εικόνα 38 – Αριθμητικές εκφράσεις σε λίστες.

◦ s.pop()[7] – Αυτή η μέθοδος αφαιρεί ένα στοιχείο από τη λίστα και το εμφανίζει στην οθόνη. Αν δεν οριστεί συγκεκριμένη θέση, τότε θα διώξει το τελευταίο στοιχείο της λίστας. Είναι η μοναδική μέθοδος στις λίστες που κάνει μεταβολή στη λίστα και ταυτόχρονα επιστρέφει μια τιμή.

```
>>> s.pop(9)
'bpis'
>>> s.pop()
2011
```

[7] Το pop() διαφέρει από το del() που αναφέρθηκε παραπάνω, επειδή το pop() εμφανίζει το στοιχείο που διαγράφεται. Το del() επίσης μπορεί να χρησιμοποιηθεί για να διαγράψει ολόκληρα κομμάτια από τη λίστα.

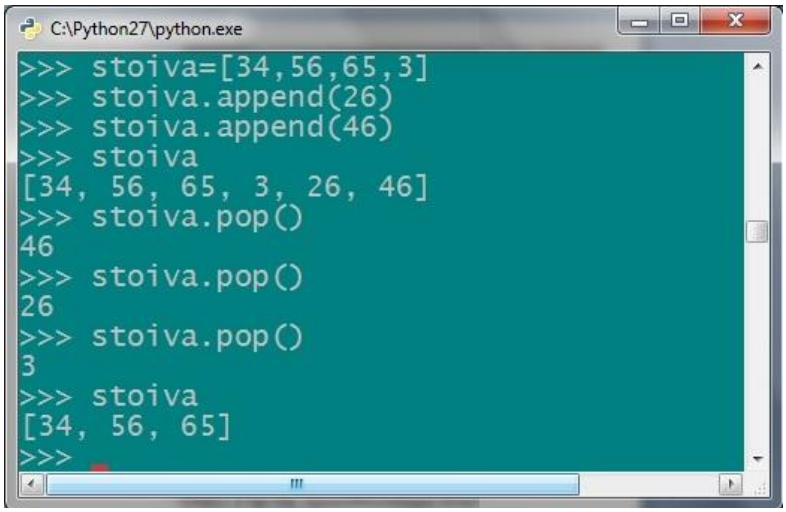


Εικόνα 39 – Μέθοδος pop.

3.3.3 *Χρησιμοποιώντας τις λίστες ως στοίβες.*

Μια στοίβα, μπορεί να υλοποιηθεί με μορφή λίστας και η κατασκευή της είναι πολύ εύκολη, όπως θα φανεί παρακάτω. Η στοίβα χρησιμοποιεί τη μέθοδο LiFo (Last in First Out). Για να προστεθεί ένα νέο μέλος στη στοίβα χρησιμοποιείται το append(). Η αφαίρεση του στοιχείου, επιτυγχάνεται με το pop() χωρίς δείκτη στην παρένθεση, ώστε να φύγει το τελευταίο στοιχείο της λίστας.

```
>>> stoiva=[34,56,65,3]
>>> stoiva.append(26)
>>> stoiva.append(46)
>>> stoiva
[34, 56, 65, 3, 26, 46]
>>> stoiva.pop()
46
>>> stoiva.pop()
26
>>> stoiva.pop()
3
>>> stoiva
[34, 56, 65]
```



Εικόνα 40 – Υλοποίηση στοίβας.

3.3.4 *Λίστες για δημιουργία μιας ουράς.*

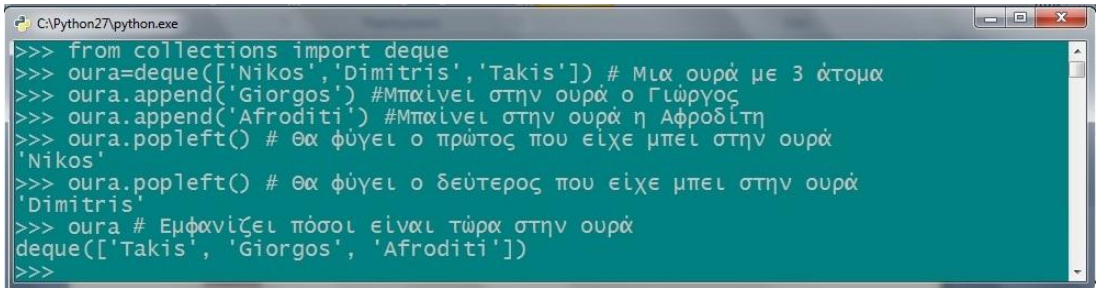
Εκτός από τις στοίβες, δίνεται η δυνατότητα να δημιουργηθούν και ουρές. Μια ουρά χρησιμοποιεί την μέθοδο FiFo (First in First out), αλλά αυτή η διαδικασία δεν είναι τόσο εύκολη στην υλοποίηση της όσο εκείνη της ουράς, που απλά προσθέτει και αφαιρεί στοιχεία ο χειριστής στο τέλος της στοίβας.

Εδώ, πρέπει να εισαχθεί το collections.deque από τη βιβλιοθήκη της python, που επιτρέπει την εισαγωγή και την αφαίρεση στοιχείων και από τις δύο άκρες της λίστας. Το collections είναι μια συλλογή από δομές δεδομένων που έχουν δημιουργηθεί για την python. Αυτός ο όρος εμφανίζεται σε πολλές γλώσσες προγραμματισμού.

Με την παρακάτω εντολή εισάγεται η deque και στη συνέχεια εκτελούνται οι απαιτούμενες εντολές για την υλοποίηση της ουράς:

```
>>> from collections import deque
>>> oura=deque(['Nikos','Dimitris','Takis']) # Μια ουρά με 3 άτομα
>>> oura.append('Giorgos') # Μπαίνει στην ουρά ο Γιώργος
>>> oura.append('Afroditi') # Μπαίνει στην ουρά η Αφροδίτη
>>> oura.popleft() # Θα φύγει ο πρώτος που είχε μπει στην ουρά
'Nikos'
>>> oura.popleft() # Θα φύγει ο δεύτερος που είχε μπει στην ουρά
'Dimitris'
>>> oura # Εμφανίζει πόσοι είναι τώρα στην ουρά
deque(['Takis', 'Giorgos', 'Afroditi'])
```

Για να βγει κάποιος από την ουρά, εκτελείται η εντολή popleft() αντί του pop() και αφήνεται κενή η παρένθεση, για να φύγει ο πρώτος της λίστας.



Εικόνα 41 – Η λειτουργία της ουράς.

3.3.5 *Συναρτησιακό μοντέλο στις δομές δεδομένων.*

Όπως αναφέρθηκε στην αρχή της μελέτης, η python υποστηρίζει το συναρτησιακό μοντέλο προγραμματισμού. Το μοντέλο αυτό έχει εφαρμογή στις δομές δεδομένων και συγκεκριμένα στις λίστες, Οι συναρτήσεις αυτές είναι τα filter(), map() και reduce().[8]

◦ Το Filter(function, sequence) επιστρέφει μια ακολουθία αποτελούμενη από εκείνα τα στοιχεία της ακολουθίας για τα οποία το function(item) είναι αληθές. Όπως λέει και το όνομα του είναι σαν να φιλτράρονται τα δεδομένα με βάση κάποιο κριτήριο, που βάζει ο προγραμματιστής και να παίρνει αυτό που θέλει. Στην περίπτωση που η ακολουθία είναι χαρακτήρας ή tuple, τότε το αποτέλεσμα θα είναι του ίδιου τύπου, διαφορετικά θα είναι πάντα μια λίστα. Για να βρεθούν για παράδειγμα, οι άρτιοι αριθμοί από το 10 μέχρι και το 50 γίνει η εξής διαδικασία:

Αρχικά ορίζεται συνάρτηση, που θα επιστρέφει τους αριθμούς που είναι άρτιοι.

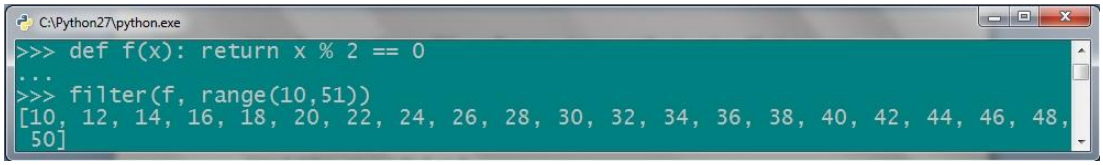
```
>>> def f(x): return x % 2 == 0
...
```

Ορίσθηκε συνάρτηση f η οποία επιστρέφει τιμή μόνο στην περίπτωση που το υπόλοιπο της διαίρεσης του x με το 2 ισοδυναμεί με το 0. Τώρα θα χρησιμοποιηθεί η εντολή filter(f, range(10,51)), όπου f το όνομα της συνάρτησης και range(10,51) η ακολουθία για την οποία επιθυμείται να υπολογιστούν οι άρτιοι.

```
>>> filter(f, range(10,51))
```

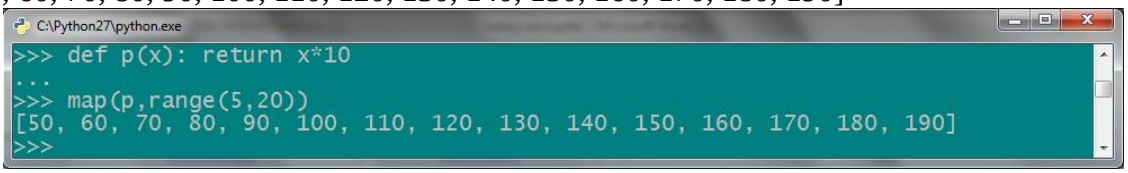
[8] Python documentration, Data Structures – Functional programming tools, Ανάκτηση στις 26-6-2011 από http://docs.python.org/tutorial/

[10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50]
Εμφανίζονται όλοι οι άρτιοι αριθμοί από το 10 έως και το 50.



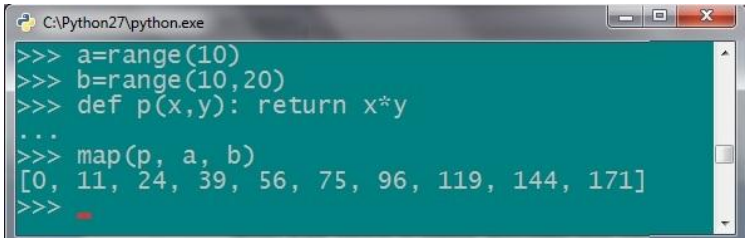
Εικόνα 42 – Εντολή filter.

◦ Το map(function, sequence) καλεί το function(item) για κάθε στοιχείο της ακολουθίας και εμφανίζει μια λίστα με τις επιστρεφόμενες τιμές. Για παράδειγμα, να πολλαπλασιαστούν με το 10, τα στοιχεία μιας ακολουθίας από το 5 μέχρι το 20.
>>> def p(x): return x*10
...
>>> map(p, range(5,20))
[50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190]



Εικόνα 43 – Εντολή map.

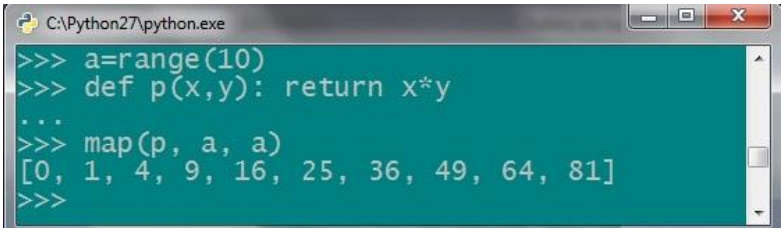
Μπορούν να υπάρχουν και περισσότερες από μία ακολουθίες. Σε τέτοια περίπτωση, η συνάρτηση πρέπει να έχει τόσες παραμέτρους, όσες είναι και οι ακολουθίες. Κάθε ακολουθία, θα είναι επακόλουθη της προηγούμενης, αφού η μία καλεί την άλλη (ή και κανένα αν κάποια ακολουθία είναι μικρότερη από μια άλλη).
>>> a=range(10)
>>> b=range(10,20)
>>> def p(x,y): return x*y
...
>>> map(p, a, b)
[0, 11, 24, 39, 56, 75, 96, 119, 144, 171]
Το αποτέλεσμα προκύπτει κάθε φορά, από τον πολλαπλασιασμό ενός στοιχείου της πρώτη ακολουθίας με το αντίστοιχο στοιχείο της δεύτερης ακολουθίας.



Εικόνα 44 – Χρήση της map, με πολλές ακολουθίες.

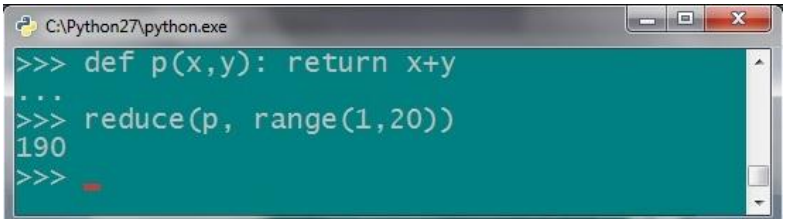
Η χρησιμοποίηση της ίδιας ακολουθίας δυο φορές, θα γίνει φαίνεται παρακάτω. Εμφανίζει το αποτέλεσμα κάνοντας πράξεις στην ίδια λίστα. Είναι ουσιαστικά σαν να είχε οριστεί το x*x αντί του x*y.
>>> a=range(10)
>>> def p(x,y): return x*y
...

>>> map(p, a, a)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]



Εικόνα 45 – Map, σε ίδιες ακολουθίες.

◦ Το reduce (function, sequence) επιστρέφει μια τιμή, η οποία προκύπτει από το κάλεσμα μιας συνάρτησης και μίας ακολουθίας. Η συνάρτηση μπορεί να είναι από τη βιβλιοθήκη της python, ή κάποια που έχει ορίσει ο προγραμματιστής. Στην ακολουθία, x και y εννοούνται τα δύο πρώτα στοιχεία της ακολουθίας. Στη συνέχεια το επόμενο στοιχείο και ούτω καθεξής. Θα υπολογιστεί το άθροισμα των αριθμών από το 1 μέχρι το 20.
>>> def p(x,y): return x+y
...
>>> reduce(p, range(1,20))
190



Εικόνα 46 – Εντολή reduce.

3.4 Tuples

Στις ακολουθίες εκτός από τις λίστες υπάρχουν και τα tuples. Η βασική τους διαφορά είναι ότι, ενώ στις λίστες ο χειριστής μπορεί να μεταβάλει τα στοιχεία τους, να προσθέσει και να αφαιρέσει στοιχεία, στα tuples δεν ισχύει αυτό. Δεν μπορούν να μεταβληθούν τα στοιχεία ενός tuple. Είναι ιδιαίτερα χρήσιμα σε περιπτώσεις που υπάρχουν ευαίσθητα δεδομένα και πρέπει να προστατευτούν από αλλαγές.

Ο ορισμός ενός tuple γίνεται εύκολα, χωρίζοντας μερικές τιμές με κόμμα. Η δήλωση γίνεται κατευθείαν και δεν είναι απαραίτητο να χρησιμοποιηθούν παρενθέσεις.
t=”teipat”,”bpis”,”patra”

Τα στοιχεία του tuple παίρνουν μια αρίθμηση από το 0 όπως ακριβώς και στις λίστες. Με τη χρήση αρνητικών τιμών στο δείκτη, μετράει τα στοιχεία αντίστροφα.
>>> t[0]
'teipat'
>>> t[2]
'patra'
>>> t[-1]
'patra'
>>> t[-3]
'teipat'

Επίσης στα tuples, ισχύει και η μέθοδος slice. Δηλαδή όπως με το slice δημιουργείται μια καινούρια λίστα, έτσι και στα tuples, θα γίνει ένα νέο tuple.

```
>>> t[1:2]
('bpis',)
>>> t[1:3]
('bpis', 'patra')
```

Εικόνα 47 – Χαρακτηριστικά των tuples.

Δεν εφαρμόζονται μέθοδοι όπως τα append, extend, index κλπ. αλλά με τον τελεστή in ελέγχεται αν υπάρχει ένα στοιχείο στο tuple και με το not in, αν δεν βρίσκεται το στοιχείο στο tuple.

```
>>> 'bpis' in t
True
>>> 'sdo' not in t
True
```

Εικόνα 48 – Έλεγχος στοιχείου.

3.4.1 Χρησιμότητα των tuples.

Όπως αναφέρθηκε παραπάνω, τα tuples είναι απλές δομές δεδομένων και το μόνο που μπορεί ο προγραμματιστής να κάνει, είναι να τις δημιουργήσει και να προσπελάσει τα στοιχεία τους. Οι λόγοι που εξυπηρετεί η χρήση ενός tuple αντί κάποιας άλλης δομής, είναι οι παρακάτω:

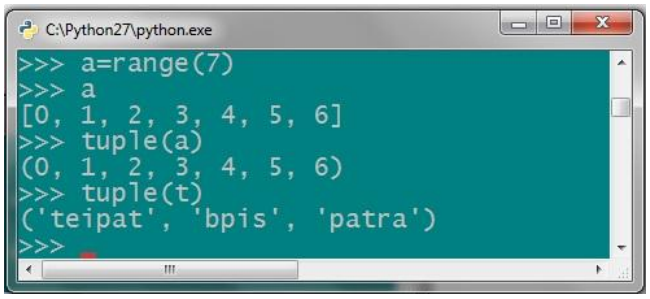
- Επιλέγεται για τη δημιουργία μιας συλλογής από δεδομένα. Αν δεν επιθυμείται η μελλοντική μετατροπή των δεδομένων που δημιουργούνται, είναι καλύτερο να χρησιμοποιηθεί το tuple αντί της λίστας.
- Τα tuples είναι γρηγορότερα από τις λίστες.
- Είναι επίσης ασφαλή, όταν υπάρχουν ευαίσθητα δεδομένα, που δεν θα μεταβληθούν στο μέλλον.
- Εκτός των άλλων, τα tuples μπορούν να χρησιμοποιηθούν και ως λέξεις κλειδιά σε ένα λεξικό ενώ με τις λίστες δεν συμβαίνει κάτι τέτοιο. Οι λέξεις κλειδιά πρέπει να είναι αμετάβλητες και έτσι δεν είναι ασφαλές να μπεί μια λίστα σαν λέξη κλειδί.

- Τέλος, ένας βασικός λόγος ύπαρξής τους είναι ότι επιστρέφονται από ορισμένες ενσωματωμένες συναρτήσεις της Python.

Τα tuples μπορούν να μετατραπούν σε λίστες και αντίστροφα. Η ενσωματωμένη συνάρτηση *tuple* παίρνει μια λίστα ή μία ακολουθία γενικά και επιστρέφει ένα tuple με τα ίδια στοιχεία. Η συνάρτηση *list* παίρνει ένα tuple και επιστρέφει μια λίστα. Με λίγα λόγια, το *tuple* παγώνει μια λίστα και το *list* ξεπαγώνει το tuple.

Αν εκτελεστεί η ενσωματωμένη συνάρτηση tuple(), θα επιστρέψει μια ακολουθία που θα έχει μετατραπεί σε tuple. Εάν η ακολουθία είναι ήδη tuple επιστρέφεται ως έχει:

```
>>> a=range(7)
>>> a
[0, 1, 2, 3, 4, 5, 6]
>>> tuple(a)
(0, 1, 2, 3, 4, 5, 6)
>>> tuple(t)
('teipat', 'bpis', 'patra')
```



Εικόνα 49 – Μετατροπή λίστας σε tuple.

3.5 Λεξικά

Στα λεξικά υπάρχει σχέση ένα προς ένα, ανάμεσα σε κλειδιά και τιμές. Το κλειδί πρέπει να είναι μοναδικό, για μην επιστρέφονται περισσότερα από ένα δεδομένα όταν καλείται. Επίσης το κλειδί μπορεί να είναι χαρακτήρας, αριθμός, ή ακόμα και ένα tuple. Όπως αναφέρθηκε παραπάνω, τα κλειδιά πρέπει να είναι αμετάβλητα, αλλά δεν ισχύει το ίδιο για το περιεχόμενο των κλειδιών.

Για να οριστεί ένα λεξικό, γράφεται το όνομα του λεξικού ακολουθούμενο από τα κλειδιά και την αντίστοιχη τιμή τους, μέσα σε άγκιστρα.

```
>>> lexiko ={'k1':'timi1','k2':'timi2','k3':'timi3'}
>>> lexiko
{'k3': 'timi3', 'k2': 'timi2', 'k1': 'timi1'}
```

Με τις παραπάνω εντολές, ορίστηκε ένα λεξικό με όνομα lexiko. Μέσα σε άγκιστρα προστέθηκαν τα κλειδιά k1 με τιμή timi1, k2 με τιμή timi2 και k3 με τιμή timi3. Κάθε στοιχείο του λεξικού είναι ένα ζευγάρι κλειδιού – τιμής.

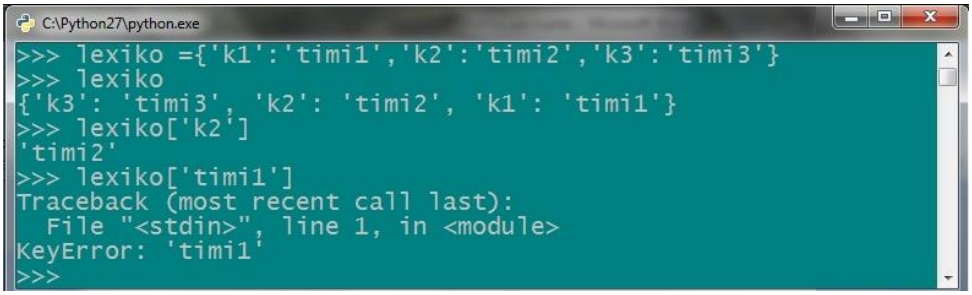
Οι βασικές λειτουργίες των λεξικών είναι σε πολλές περιπτώσεις οι ίδιες με εκείνες που αναφέρθηκαν στις ακολουθίες. Στη συνέχεια θα αναλυθούν όλες οι λειτουργίες των λεξικών.

Στο παραπάνω παράδειγμα, το k2 είναι κλειδί του λεξικού, έτσι αν εκτελεστεί το όνομα του, θα επιστραφεί στην οθόνη η αντίστοιχη τιμή του:

```
>>> lexiko['k2']
'timi2'
```

Αν στη θέση του όμως μπει η τιμή, αντί για το κλειδί, θα εμφανίσει σφάλμα επειδή πρέπει υποχρεωτικά να δηλωθεί κλειδί.

```
>>> lexiko['timi1']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'timi1'
```



Εικόνα 50 – Δημιουργία και εμφάνιση λεξικού.

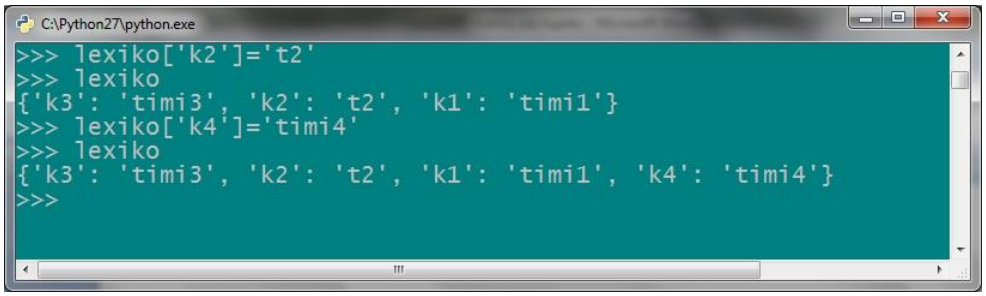
3.5.1 Τροποποιώντας ένα λεξικό.

Σε περίπτωση που ο προγραμματιστής θέλει να μεταβάλει τη τιμή ενός κλειδιού σε ένα λεξικό, τότε μέσα σε αγκύλη θα βάλει το όνομα του κλειδιού και μετά το ίσον, τη νέα επιθυμητή τιμή του κλειδιού.

```
>>> lexiko['k2']='t2'
>>> lexiko
{'k3': 'timi3', 'k2': 't2', 'k1': 'timi1'}
```

Δεν γίνεται να υπάρχουν πολλά κλειδιά με το ίδιο όνομα. Βάζοντας νέα τιμή σε ένα κλειδί, αυτομάτως θα αντικαταστήσει την παλιά του τιμή. Επίσης, είναι δυνατή η πρόσθεση ενός καινούριου κλειδιού στο λεξικό μαζί με την αντίστοιχη τιμή του.

```
>>> lexiko['k4']='timi4'
>>> lexiko
{'k3': 'timi3', 'k2': 't2', 'k1': 'timi1', 'k4': 'timi4'}
```



Εικόνα 51 – Αλλαγές στα λεξικά.

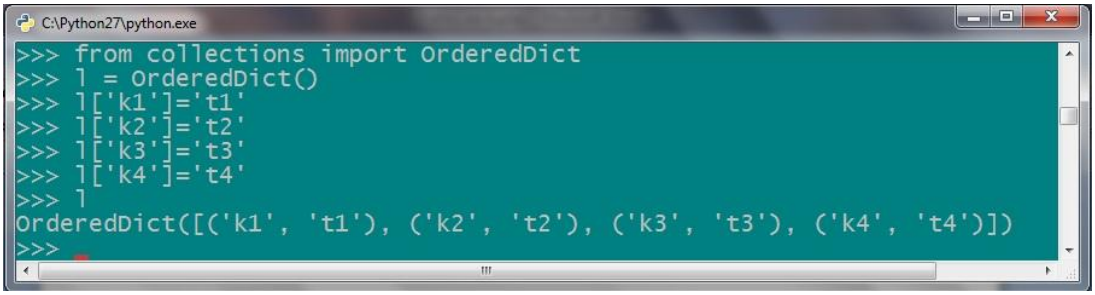
3.5.2 Λειτουργίες στα λεξικά.

Στα λεξικά δεν υπάρχει συγκεκριμένος τρόπος ταξινόμησης. Η σειρά που επιστρέφονται τα δεδομένα εξαρτάται μόνο από το χρόνο που εισήχθησαν τα ζευγάρια και δεν τους εξασφαλίζεται κάποια σειρά εμφάνισης. Τα κλειδιά επίσης, δεν αλλάζουν τη θέση που εμφανίζονται, αν για παράδειγμα αλλάξει η τιμή τους, κάτι που σημαίνει ότι δεν μπορούν να ταξινομηθούν αυτόματα.

Σε περίπτωση όμως, που χρειάζεται να υπάρχουν ταξινομημένα τα ζευγάρια κλειδιών/τιμών, πρέπει να εισαχθεί το OrderedDict από το collections της python βιβλιοθήκης και να ταξινομηθούν από τον προγραμματιστή. Με το OrderedDict, θα εισαχθούν τα δεδομένα με την επιθυμητή ταξινομημένη σειρά και θα εμφανίζονται πλέον τα κλειδιά/τιμές με τη συγκεκριμένη σειρά.

```
>>>from collections import OrderedDict
```

```
>>> l = OrderedDict()
>>> l['k1']='t1'
>>> l['k2']='t2'
>>> l['k3']='t3'
>>> l['k4']='t4'
>>> l
OrderedDict([('k1', 't1'), ('k2', 't2'), ('k3', 't3'), ('k4', 't4')])
```



Εικόνα 52 – Ταξινόμηση στα λεξικά.

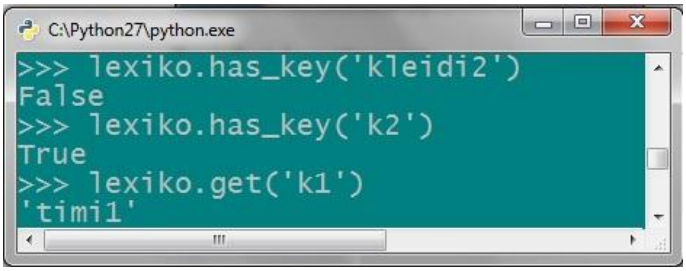
Τα λεξικά υποστηρίζουν διάφορες λειτουργίες και οι βασικότερες από αυτές είναι οι ακόλουθες:

◦ Η συνάρτηση has_key() είναι Boolean και κάνει έλεγχο για το αν υπάρχει μια συγκεκριμένη λέξη κλειδί σε ένα λεξικό.

```
 >>> lexiko.has_key('kleidi2')
False
>>> lexiko.has_key('k2')
True
```

◦ Το get() επιστρέφει την τιμή ενός κλειδιού, εφόσον αυτή υπάρχει στο λεξικό.

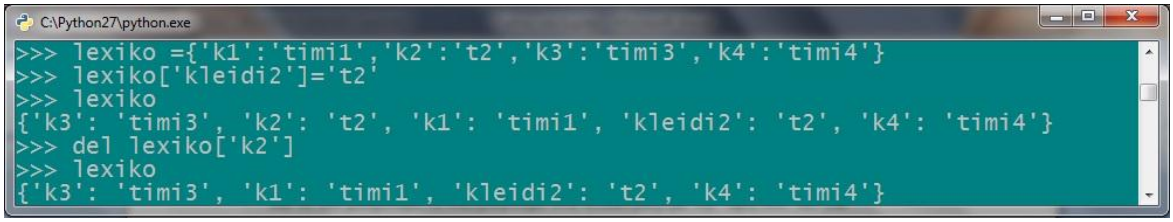
```
>>> lexiko.get('k1')
'timi1'
```



Εικόνα 53 – Συναρτήσεις των λεξικών.

◦ Με το del διαγράφεται ένα κλειδί μαζί με την τιμή του. Για παράδειγμα, η μετονομασία του κλειδιού k2 σε kleidi2, επιτυγχάνεται με την πρόσθεση του νέου κλειδιού και την διαγραφή του παλιού.

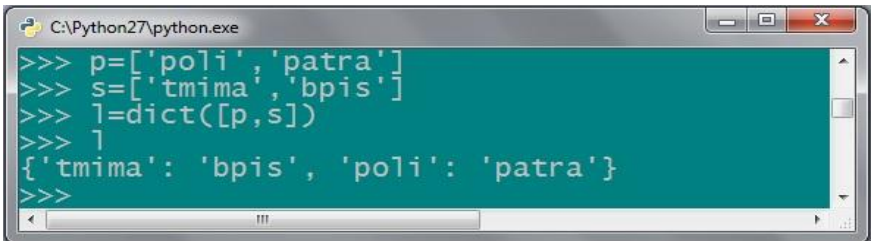
```
>>> lexiko['kleidi2']='t2'
>>> lexiko
{'k3': 'timi3', 'k2': 't2', 'k1': 'timi1', 'kleidi2': 't2', 'k4': 'timi4'}
>>> del lexiko['k2']
>>> lexiko
{'k3': 'timi3', 'k1': 'timi1', 'kleidi2': 't2', 'k4': 'timi4'}
```



Εικόνα 54 – Διαγραφή κλειδιού στο λεξικό.

◦ Με την ενσωματωμένη συνάρτηση dict(), κατασκευάζεται ένα λεξικό χρησιμοποιώντας άλλα λεξικά, tuples και από ακολουθίες με ζεύγος κλειδιού – τιμής. Σε δύο ακολουθίες p και s, καλείται η συνάρτηση dict() και μετατρέπονται σε ένα λεξικό.

```
>>> p=['poli','patra']
>>> s=['tmima','bpis']
>>> l=dict([p,s])
>>> l
{'tmima': 'bpis', 'poli': 'patra'}
```



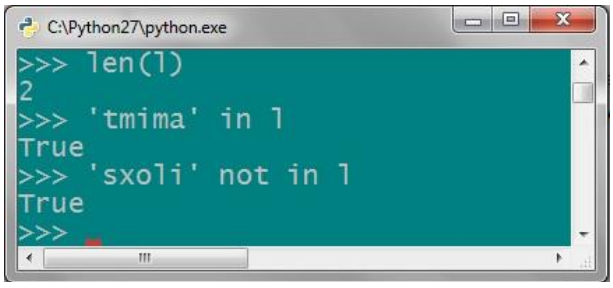
Εικόνα 55 – Μετατροπή ακολουθίας σε λεξικό.

◦ Υπάρχει ακόμα και η εντολή len, η οποία αναφέρει πόσα ζευγάρια υπάρχουν σε ένα λεξικό.

```
>>> len(l)
2
```

Για τον έλεγχο ενός κλειδιού, αν δηλαδή υπάρχει σε ένα λεξικό, χρησιμοποιείται ο τελεστής in όπως έχει γίνει και στις προηγούμενες ενότητες. Με το not in ελέγχεται, αν το συγκεκριμένο κλειδί δεν είναι στο λεξικό.

```
>>> 'tmima' in l
True
>>> 'sxoli' not in l
True
```

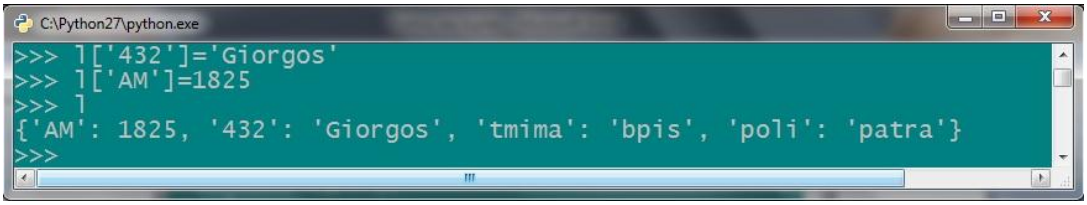


Εικόνα 56 – Έλεγχος κλειδιού στα λεξικά.

Τα κλειδιά και οι τιμές στα λεξικά, όπως αναφέρθηκε παραπάνω μπορεί να είναι και αριθμοί:

```
>>> l['432']='Giorgos'
>>> l['AM']=1825
```

```
>>> l
{'AM': 1825, '432': 'Giorgos', 'tmima': 'bpis', 'poli': 'patra'}
```



Εικόνα 57 – Αριθμοί ως κλειδιά και τιμές στα λεξικά.

Υπάρχει επίσης η δυνατότητα να κατασκευαστεί κάποιο λεξικό απευθείας, με τυχαίες ακολουθίες από ζευγάρια τιμών, όπως στο παράδειγμα παρακάτω:

```
>>> t=dict(ddff='sw',sefd=34,erg='srt')
>>> t
{'ddff': 'sw', 'erg': 'srt', 'sefd': 34}
```

Μολονότι υπάρχουν αρκετά κοινά στοιχεία ανάμεσα στα λεξικά και τις λίστες, έχουν ορισμένες σημαντικές διαφορές:

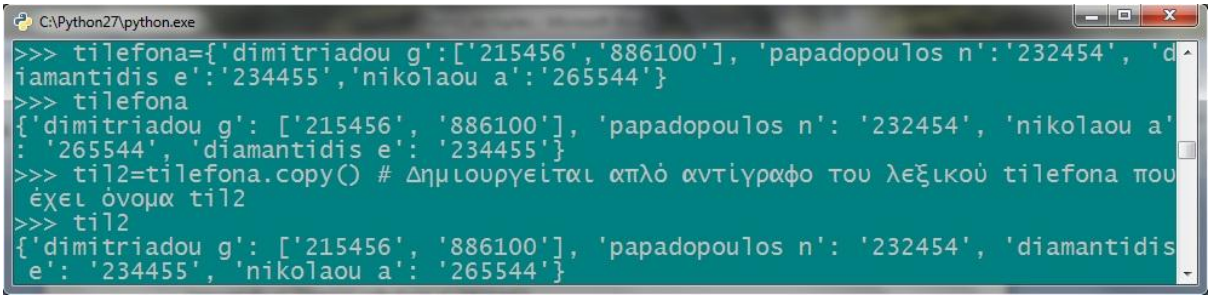
- Οι τύποι των κλειδιών ενός λεξικού δεν απαιτείται να είναι μόνο ακέραιοι αριθμοί, αλλά μπορεί να είναι και πραγματικοί αριθμοί, χαρακτήρες ή και tuples.
- Στις λίστες χρησιμοποιούνται οι τελεστές in και not in για τον έλεγχο μιας τιμής, αν υπάρχει δηλαδή ή όχι, η συγκεκριμένη τιμή στη λίστα. Στα λεξικά όμως αυτή τη διαδικασία δεν γίνεται για να βρεθεί μια τιμή, αλλά για να ενημερωθεί ο προγραμματιστής, για την ύπαρξη του συγκεκριμένου κλειδιού στο λεξικό.
- Για να την πρόσθεση μιας νέας τιμής στη λίστα, χρησιμοποιείται η μέθοδος append. Στα λεξικά όμως, προστίθεται ένα νέο ζευγάρι κλειδιού και τιμής και μπαίνει κατευθείαν στο λεξικό.

3.5.3 Μέθοδοι λεξικών.

Τα λεξικά έχουν μερικές μεθόδους, που βοηθούν στη καλύτερη και ευκολότερη διαχείριση τους. Θα αναλυθούν παρακάτω, χρησιμοποιώντας το παράδειγμα του τηλεφωνικού καταλόγου.

◦ Η μέθοδος .copy() αντιγράφει ένα λεξικό και κατασκευάζει ένα καινούργιο με τα ίδια ζευγάρια κλειδιών - τιμών. Το νέο λεξικό θα είναι ακριβές αντίγραφο του αρχικού, αλλά μόνο ως προς τα κλειδιά. Όσο για τις τιμές του, θα είναι οι ίδιες με το πρώτο λεξικό και όχι αντίγραφά τους.

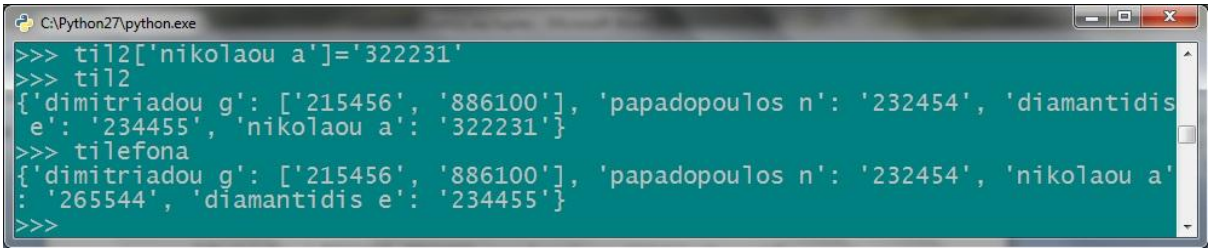
```
>>> tilefona={'dimitriadou g':['215456','886100'], 'papadopoulos n':'232454', 'diamantidis e':'234455','nikolaou a':'265544'}
>>> tilefona
{'dimitriadou g': ['215456', '886100'], 'papadopoulos n': '232454', 'nikolaou a'
: '265544', 'diamantidis e': '234455'}
>>> til2=tilefona.copy() # Δημιουργείται απλό αντίγραφο του λεξικού tilefona που
 έχει όνομα til2
>>> til2
{'dimitriadou g': ['215456', '886100'], 'papadopoulos n': '232454', 'diamantidis
 e': '234455', 'nikolaou a': '265544'}
```



Εικόνα 58 – Αντιγραφή λεξικού.

Εάν τώρα αντικατασταθεί κάποια μια τιμή στο λεξικό til2, το αρχικό λεξικό tilefona θα μείνει ανεπηρέαστο. Το ίδιο ισχύει και αντίστροφα.

```
>>> til2['nikolaou a']='322231'
>>> til2
{'dimitriadou g': ['215456', '886100'], 'papadopoulos n': '232454', 'diamantidis
 e': '234455', 'nikolaou a': '322231'}
>>> tilefona
{'dimitriadou g': ['215456', '886100'], 'papadopoulos n': '232454', 'nikolaou a'
: '265544', 'diamantidis e': '234455'}
```

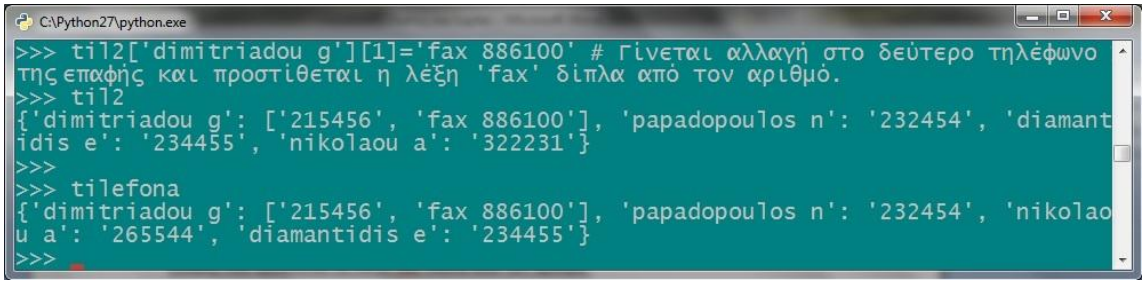


Εικόνα 59 – Αλλαγή τιμών σε αντίγραφο λεξικού.

Σε περίπτωση όμως που δεν αντικατασταθεί η τιμή του κλειδιού και απλά τροποποιηθεί ένα μέρος της, τότε η αλλαγή αυτή θα αντικατοπτριστεί και στα δυο λεξικά.

```
>>> til2['dimitriadou g'][1]='fax 886100' # Γίνεται αλλαγή στο δεύτερο τηλέφωνο της επαφής και προστίθεται η λέξη 'fax' δίπλα από τον αριθμό.
>>> til2
{'dimitriadou g': ['215456', 'fax 886100'], 'papadopoulos n': '232454', 'diamantidis e': '234455', 'nikolaou a': '322231'}
>>> tilefona
{'dimitriadou g': ['215456', 'fax 886100'], 'papadopoulos n': '232454', 'nikolaou a': '265544', 'diamantidis e': '234455'}
```

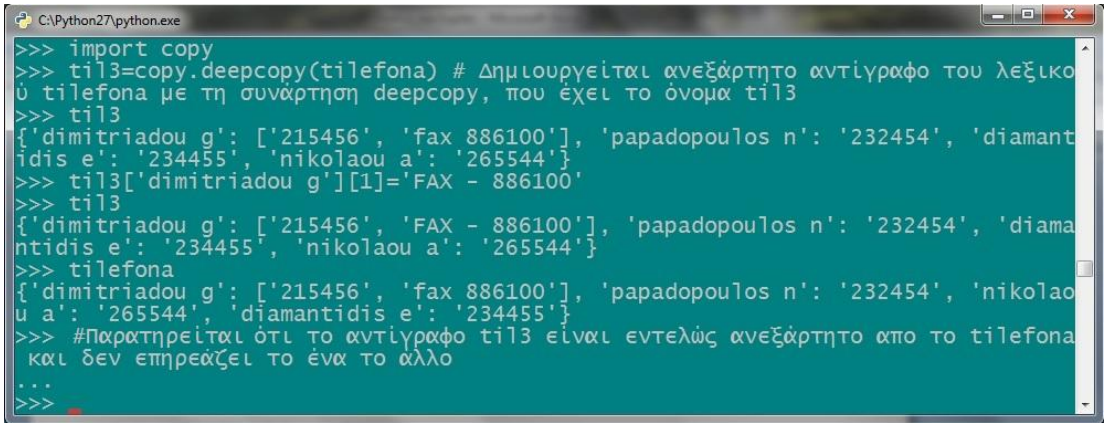
Παρατηρείται ότι η επεξεργασία σε μια από τις τιμές του λεξικού til2 επηρέασε και το λεξικό tilefona. Το ίδιο αποτέλεσμα θα εμφανιζόταν, αν η αλλαγή αυτή γινόταν στο λεξικό tilefona.



Εικόνα 60 – Τροποποίηση τιμής σε αντίγραφο λεξικό.

Το αντίγραφο που κατασκευάστηκε παραπάνω, δεν ήταν και τόσο ανεξάρτητο με το αρχικό λεξικό. Για ένα εντελώς ανεξάρτητο αντίγραφο, πρέπει πρώτα να εισαχτεί το module copy από την βιβλιοθήκη της python και να εκτελεστεί η συνάρτηση deepcopy.[9] Το deepcopy θα κατασκευάσει ένα αντίγραφο, που θα περιέχει τα πλήρη δεδομένα του αρχικού λεξικού, επιτρέποντάς το να χρησιμοποιηθεί ανεξάρτητα από το πρωτότυπο.[10]

```
>>> import copy
>>> til3=copy.deepcopy(tilefona) # Δημιουργείται ανεξάρτητο αντίγραφο του λεξικού tilefona με τη συνάρτηση deepcopy, που έχει το όνομα til3
>>> til3
{'dimitriadou g': ['215456', 'fax 886100'], 'papadopoulos n': '232454', 'diamantidis e': '234455', 'nikolaou a': '265544'}
>>> til3['dimitriadou g'][1]='FAX - 886100'
>>> til3
{'dimitriadou g': ['215456', 'FAX - 886100'], 'papadopoulos n': '232454', 'diamantidis e': '234455', 'nikolaou a': '265544'}
>>> tilefona
{'dimitriadou g': ['215456', 'fax 886100'], 'papadopoulos n': '232454', 'nikolaou a': '265544', 'diamantidis e': '234455'}
>>> # Παρατηρείται ότι το αντίγραφο til3 είναι εντελώς ανεξάρτητο από το tilefona
και δεν επηρεάζει το ένα το άλλο.
```



Εικόνα 61 – Αντίγραφο με χρήση του deepcopy.

[9] Λεβεντέας Δ. (2010), «Εκμάθηση python βήμα βήμα», (σελ. 88), Ελληνική κοινότητα Προγραμματιστών python – Ανάκτηση στις 5-7-2011 από http://python.org.gr/index.php?option=com_phocadownload&view=category&id=3:tutorials&Itemid=58

[10] Το σημείο που υστερεί το copy και κάποιες φορές πρέπει να εφαρμοστεί το deepcopy, είναι όταν το απλό αντίγραφο και το πρωτότυπο αλληλοδανείζονται δεδομένα και μια αλλαγή στο ένα, θα επηρεάσει το άλλο.

◦ Η μέθοδος .keys() εμφανίζει όλα τα κλειδιά, που είναι δηλωμένα στο λεξικό. Τα κλειδιά για το παραπάνω παράδειγμα είναι:

```
>>> tilefona.keys()
['dimitriadou g', 'papadopoulos n', 'nikolaou a', 'diamantidis e']
```

Εικόνα 62 - Η μέθοδος .keys().

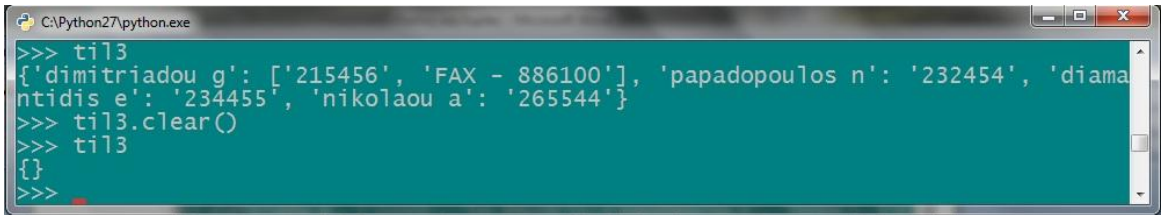
◦ Η μέθοδος .items() σπάει όλα τα στοιχεία ενός λεξικού και τα επιστρέφει σε ζευγάρια (κλειδί, τιμή) μέσα σε μια λίστα. Τα στοιχεία επιστρέφονται με τυχαία σειρά:

```
>>> tilefona.items()
[('dimitriadou g', ['215456', 'fax 886100']), ('papadopoulos n', '232454'), ('nikolaou a', '265544'), ('diamantidis e', '234455')]
```

Εικόνα 63 - Η μέθοδος .items().

◦ Η μέθοδος .clear(), θα διαγράψει όλα τα στοιχεία που έχει το λεξικό και θα το επιστρέψει άδειο:

```
>>> til3
{'dimitriadou g': ['215456', 'FAX - 886100'], 'papadopoulos n': '232454', 'diamantidis e': '234455', 'nikolaou a': '265544'}
>>> til3.clear()
>>> til3
{}
```



Εικόνα 64- Η μέθοδος .clear().

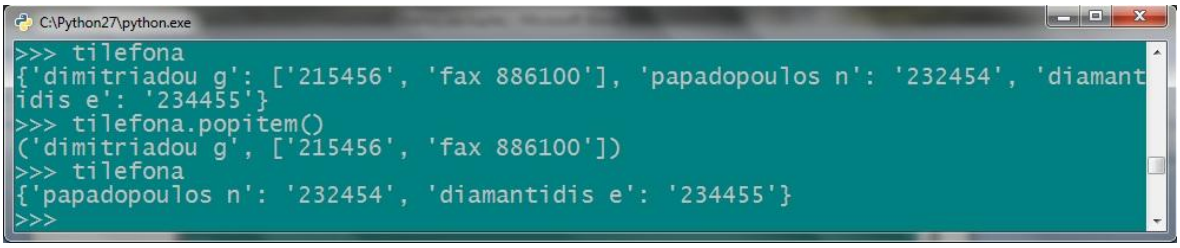
◦ Η μέθοδος .pop('κλειδί') όπως και στις λίστες, επιστρέφει την τιμή ενός συγκεκριμένου κλειδιού και ταυτόχρονα το αφαιρεί μαζί με την τιμή του από το λεξικό.

```
>>> tilefona.pop('nikolaou a')
'265544'
>>> tilefona
{'dimitriadou g': ['215456', 'fax 886100'], 'papadopoulos n': '232454', 'diamantidis e': '234455'}
```

Εικόνα 65 - Η μέθοδος .pop('κλειδί').

◦ Η μέθοδος .popitem() είναι παρόμοια με την .pop, αλλά η .popitem παίρνει αυθαίρετα ένα τυχαίο κλειδί, επιστρέφει την τιμή του και στην συνέχεια διαγράφει το ζευγάρι:

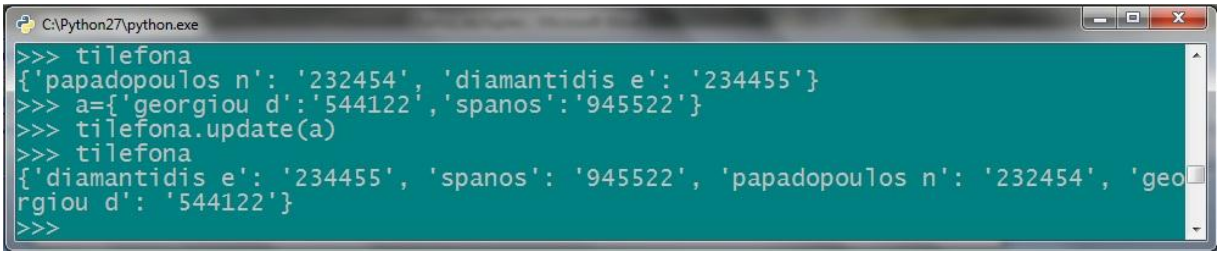
```
>>> tilefona.popitem()
('dimitriadou g', ['215456', 'fax 886100'])
>>> tilefona
{'papadopoulos n': '232454', 'diamantidis e': '234455'}
```



Εικόνα 66 - Η μέθοδος .popitem().

◦ Η μέθοδος .update(a) ενημερώνει ένα λεξικό με νέες εγγραφές από κάποιο άλλο λεξικό, έστω a:

```
>>> tilefona
{'papadopoulos n': '232454', 'diamantidis e': '234455'}
>>> a={'georgiou d':'544122','spanos':'945522'}
>>> tilefona.update(a)
>>> tilefona
{'diamantidis e': '234455', 'spanos': '945522', 'papadopoulos n': '232454', 'georgiou d': '544122'}
```



Εικόνα 67 - Η μέθοδος .update().

◦ Η μέθοδος .values() επιστρέφει μια λίστα που έχει όλες τις τιμές του λεξικού:

```
>>> tilefona.values()
['234455', '945522', '232454', '544122']
```

Εικόνα 68 - Η μέθοδος .values().

3.6 Σύνολα (Sets)

Η python περιλαμβάνει και μια δομή δεδομένων για σύνολα (sets). Ένα set είναι μια μη ταξινομημένη συλλογή από δεδομένα και δεν περιέχει διπλά στοιχεία. Στη βασική του λειτουργία κάνει εξάλειψη των διπλό-εγγραφών και περιλαμβάνει έλεγχο για το αν βρίσκεται ένα αντικείμενο στο σύνολο. Τα αντικείμενα του set επίσης υποστηρίζουν μαθηματικές λειτουργίες όπως η ένωση, η τομή, η διαφορά και η συμμετρική διαφορά. Άρα, τα σύνολα επιτρέπουν στο χρήστη να ερευνήσει αν μια ομάδα αντικειμένων (υποσύνολο), αποτελεί κομμάτι ενός άλλου συνόλου, να ελέγξει την τομή μεταξύ δυο συνόλων και λοιπά.

Ένα χαρακτηριστικό παράδειγμα για την κατασκευή των συνόλων, είναι ένα καλάθι με φρούτα:

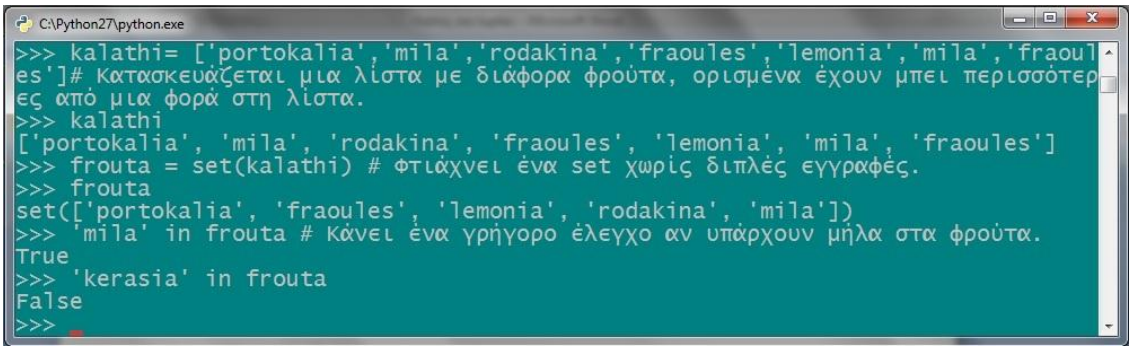
>>> kalathi = ['portokalia','mila','rodakina','fraoules','lemonia','mila','fraoules'] # **Κατασκευάζεται μια λίστα με διάφορα φρούτα, ορισμένα έχουν μπει περισσότερες από μια φορά στη λίστα.**
>>> kalathi
['portokalia', 'mila', 'rodakina', 'fraoules', 'lemonia', 'mila', 'fraoules']

Για να δημιουργηθεί ένα σύνολο από την παραπάνω λίστα, χρησιμοποιείται η εντολή set, βάζοντας σε παρένθεση το όνομα της λίστας. Σε μια καινούργια μεταβλητή, που ονομάζεται frouta, καταχωρείται το νέο set:

>>> frouta = set(kalathi) # **Φτιάχνει ένα set χωρίς διπλές εγγραφές.**
>>> frouta
set(['portokalia', 'fraoules', 'lemonia', 'rodakina', 'mila'])

Μόλις εκτελεστεί η νέα μεταβλητή, θα επιστρέψει στην οθόνη το σύνολο με τα φρούτα, που δημιουργήθηκε από τη λίστα kalathi. Με τον Boolean τελεστή in θα ελέγξει, αν το φρούτο που αναζητείται υπάρχει στο σύνολο.

>>> 'mila' in frouta # **Κάνει ένα γρήγορο έλεγχο αν υπάρχουν μήλα στα φρούτα.**
True
>>> 'kerasia' in frouta
False

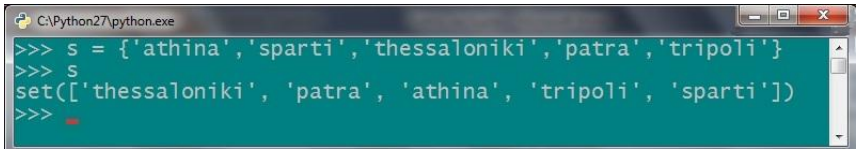


Εικόνα 69 – Δημιουργία ενός συνόλου και έλεγχος στοιχείων.

Ένα σύνολο, μπορεί να κατασκευαστεί κατευθείαν, βάζοντας τα στοιχεία μέσα σε άγκιστρο, χωρίς να έχει προηγηθεί κάποια λίστα. Ο διερμηνευτής αντιλαμβάνεται ότι έχει

ένα σύνολο και όχι ένα λεξικό, επειδή δεν υπάρχει άνω κάτω τελεία μεταξύ των τιμών, όπως γίνεται με τα λεξικά.

```
>>> s = {'athina','sparti','thessaloniki','patra','tripoli'}
>>> s
set(['thessaloniki', 'patra', 'athina', 'tripoli', 'sparti'])
```



Εικόνα 70 – Άμεση δημιουργία λεξικού.

3.6.1 Μαθηματικές πράξεις στα σύνολα.

Αναφέρθηκε προηγουμένως, ότι τα σύνολα υποστηρίζουν διάφορες μαθηματικές λειτουργίες. Στον παρακάτω πίνακα, απεικονίζονται οι πράξεις που μπορούν να εκτελεστούν στα σύνολα και η μαθηματική τους μορφή, μαζί με μία σύντομη περιγραφή κάθε εντολής.

Πίνακας 2: Οι μαθηματικές λειτουργίες στην python.		
Εντολή python	Μαθηματική μορφή	Περιγραφή εντολής
A	A	Σύνολο A
A – B	A \ B (Διαφορά)	Αφαιρεί από το σύνολο A το σύνολο B.
A | B	A U B (A ένωση B)	Ανήκει ή στο A ή στο B
A & B	A ∩ B (A τομή B)	Να ανήκει και στο A και στο B.
A ^ B	A x B (Συμμετρική διαφορά)	Να ανήκει ή στο A, ή στο B, όχι και στα δύο.
A < B	Ελέγχει αν όλα τα στοιχεία του A βρίσκονται στο B (A υποσύνολο του B).	
A > B	Ελέγχει αν όλα τα στοιχεία του B βρίσκονται στο A (A υπερσύνολο του B).	

Δημιουργούνται για παράδειγμα, δύο σύνολα A και B που περιέχουν τις λίστες athens και attiki αντίστοιχα. Μόλις εκτελεστεί, θα εμφανιστούν στην οθόνη και τα στοιχεία του.

```
>>> a = set('athens')
>>> b = set('attiki')
>>> a
set(['a', 'e', 'h', 'n', 's', 't'])
>>> b
set(['a', 'i', 'k', 't'])
```

Για να γίνουν πράξεις με τα σύνολα, πρέπει να εκτελεστούν οι εντολές, που αναφέρθηκαν νωρίτερα. Η εντολή A – B, δίνει τη διαφορά του συνόλου A από το B και θα επιστρέψει τα στοιχεία του A, τα οποία όμως δεν ανήκουν στο B.

```
>>> a - b
set(['h', 's', 'e', 'n'])
```

Με το A | B επιστρέφεται η ένωση των στοιχείων των δύο συνόλων, δηλαδή όλα τα στοιχεία, που υπάρχουν στα σύνολα.

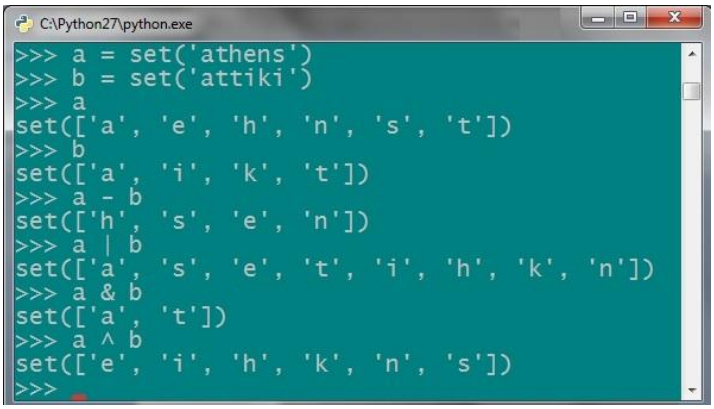
```
>>> a | b
set(['a', 's', 'e', 't', 'i', 'h', 'k', 'n'])
```

Το A & B επιστρέφει την τομή, ήτοι τα κοινά στοιχεία των δύο συνόλων.

```
>>> a & b
set(['a', 't'])
```

Τέλος, η συμμετρική διαφορά του Α από το Β, όπου εμφανίζονται όσα στοιχεία των δύο συνόλων ανήκουν μόνο σε ένα από τα σύνολα, όχι και στα δύο.

```
>>> a ^ b
set(['e', 'i', 'h', 'k', 'n', 's'])
```



Εικόνα 71 – Πράξεις στα σύνολα.

3.6.2 Μέθοδοι των συνόλων.

Εκτός από τις παραπάνω μαθηματικές εντολές, υπάρχουν και οι γνωστές πλέον μέθοδοι, που έχουν εφαρμογή και στα σύνολα.

◦ Με το .copy ως γνωστόν δημιουργείται ένα νέο σύνολο (frouta2) που θα είναι αντίγραφο του αρχικού.

```
>>> frouta='portokalia', 'mila', 'rodakina', 'fraoules', 'lemonia', 'mila', 'fraoules'
>>> frouta=set(frouta)
>>> frouta2=frouta.copy()
>>> frouta2
set(['portokalia', 'lemonia', 'fraoules', 'mila', 'rodakina'])
```

◦ Με το.add, προστίθεται ένα νέο στοιχείο στο σύνολο.

```
>>> frouta2.add('kerasia')
>>> frouta2
set(['portokalia', 'kerasia', 'rodakina', 'fraoules', 'mila', 'lemonia'])
```

◦ Το .remove αφαιρεί από ένα σύνολο, το στοιχείο που θέλει ο προγραμματιστής.

```
>>> frouta.remove('mila')
>>> frouta
set(['portokalia', 'fraoules', 'lemonia', 'rodakina'])
>>> frouta2
set(['portokalia', 'kerasia', 'rodakina', 'fraoules', 'mila', 'lemonia'])
```

Στη συνέχεια, συγκρίνονται τα δύο συνόλα, αν δηλαδή το frouta είναι υποσύνολο του frouta2.

```
>>> frouta<frouta2
True
```

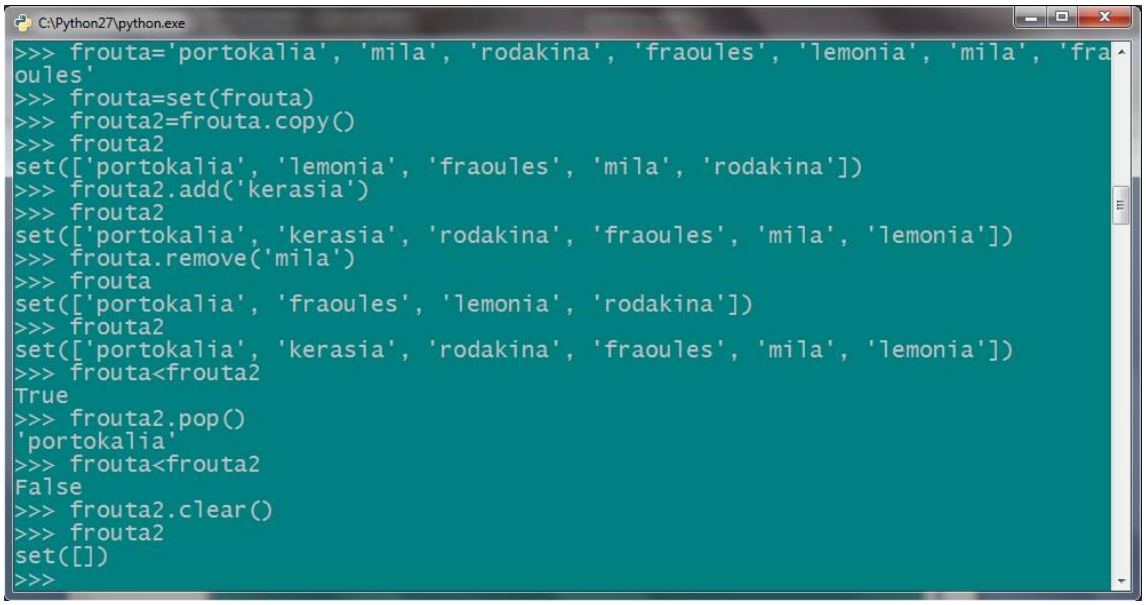
◦ Το .pop, θα αφαιρέσει ένα τυχαίο στοιχείο της λίστας.

```
>>> frouta2.pop()
'portokalia'
>>> frouta<frouta2
False
```

Αναφέρθηκε προηγουμένως ότι το < και το >, δείχνουν αν ένα σύνολο είναι υποσύνολο ή υπερσύνολο αντίστοιχα ενός άλλου συνόλου. Επειδή στο παράδειγμά το σύνολο frouta είναι υποσύνολο του frouta2, επιστρέφει την ένδειξη true στον έλεγχο που γίνεται. Στη συνέχεια όμως, αφού χρησιμοποιηθεί η μέθοδος .pop για το σύνολο frouta2, διαπιστώνεται ότι στον ίδιο έλεγχο θα πάρει την απάντηση false. Αυτό συμβαίνει, επειδή με το .pop αφαιρέθηκε από το σύνολο frouta2 το στοιχείο portokalia και έτσι, έπαψε να αποτελεί το σύνολο frouta υποσύνολο του frouta2.

◦ Η μέθοδος .clear διώχνει όλα τα στοιχεία από ένα σύνολο.

```
>>> frouta2.clear()
>>> frouta2
set([])
```



Εικόνα 72 - Μέθοδοι στα σύνολα.

3.7 Ασκήσεις στις δομές δεδομένων.

A. Στην παρακάτω λίστα, (Αεκ, Άρης, Ολυμπιακός, Παναθηναϊκός, Πάοκ, 21, 3, 7,13,4), να γίνουν τα ακόλουθα:

- Να εμφανιστούν τα στοιχεία που βρίσκονται στις θέσεις 2, 4 και 6.
- Υπάρχει στην λίστα το στοιχειό 7 και το στοιχειό 14;
- Να προστεθούν στην λίστα, τα στοιχειά Παναιτωλικός και 6 στις θέσεις του Πάοκ και του στοιχείου 4 αντίστοιχα.
- Να διαγραφούν τα στοιχεία της λίστας, που βρίσκονται μετά την έκτη θέση.

B. Σε μια λίστα που θα δημιουργηθεί, να βρεθεί το μήκος, το μεγαλύτερο και το μικρότερο στοιχειό της λίστας.

C. Να δημιουργηθεί μια λίστα και να γίνουν τα εξής:

- Να επαναληφτούν τα στοιχειά της λίστας 4 φόρες.
- Υπάρχουν στην λίστα οι αριθμοί 18 και 32;
- Να εμφανιστεί ένα κομμάτι της λίστας από το 2ο στοιχείο μέχρι το 5ο στοιχειό.
- Να προστεθεί στην λίστα ο αριθμός 65.
- Να αφαιρεθεί ένα στοιχείο της λίστας.

- Να ταξινομηθούν τα στοιχειά της λίστας και έπειτα να τα αντιστραφούν.
- Να προστεθεί ένα νέο στοιχειό στην λίστα, στην 4η θέση.

D. Σε 2 σύνολα που θα δημιουργηθούν και θα περιέχουν τα στοιχεία από τα ονόματα της Πάτρας και της Αχαΐας αντίστοιχα, να γίνουν τα παρακάτω:
   - Να αφαιρεθεί από το σύνολο Α, τα στοιχεία που υπάρχουν στο Β.
   - Να εμφανιστούν όλα τα στοιχειά που υπάρχουν στα σύνολα.
   - Να βρεθούν τα κοινά στοιχειά των 2 συνόλων και
   - Τα στοιχεία, που εμφανίζονται μόνο σε ένα από τα δύο σύνολα.

E. Στο παρακάτω σύνολο: ΜΜΜ= {΄Μετρο΄,΄Ηλεκτρικος΄,΄Λεωφορειο΄,΄Τραμ΄} να γίνουν τα εξής:
   - Να δημιουργηθεί ένα αντίγραφο του συνόλου.
   - Να προστεθεί στο σύνολο, το στοιχειό ταξί και να διαγραφεί το τραμ.
   - Να ελεγχτεί, αν το πρώτο σύνολο είναι υποσύνολο του αντίγραφου.
   - Να αφαιρεθεί ένα τυχαίο στοιχείο από την πρωτότυπη λίστα και να ξαναγίνει ο παραπάνω έλεγχος.
   - Τέλος, να διαγραφούν όλα τα στοιχειά του δεύτερου συνόλου.

F. Σε μια λίστα ακεραίων από το 0 μέχρι το 10 να βρεθεί το τετράγωνο τους και με τη μέθοδο της ουράς, να αφαιρεθούν τα δυο πρώτα στοιχεία της λίστας και να προστεθούν άλλα 3, (n+1,n+5 και n-4), όπου n είναι το τελευταίο στοιχείο της αρχικής ουράς. Στη συνέχεια να δημιουργηθεί ένα ταξινομημένο λεξικό από τις τιμές της λίστας (ουράς). Τα κλειδιά του λεξικού θα είναι 1ο stoixeio, 2ο stoixeio κλπ. και οι αντίστοιχες τιμές της ουράς. Τέλος να δημιουργηθεί αντίγραφο του λεξικού, να πολλαπλασιαστούν τις τιμές του με το 3 και να εμφανιστούν τα δυο λεξικά.

G. Να δημιουργηθούν 5 σύνολα αποτελούμενα από θετικούς ακέραιους αριθμούς μέχρι το 10 και στη συνέχεια να βρεθούν, ποιοι είναι οι αριθμοί που εμφανίζονται σε όλα τα σύνολα. Κάθε σύνολο θα αποτελείται από 7 αριθμούς.

Οι λύσεις των ασκήσεων βρίσκονται στο παράρτημα.

ΚΕΦΑΛΑΙΟ 4ο
Αντικειμενοστραφής Προγραμματισμός.

Η python δεν λειτουργεί αποκλειστικά για την εκτέλεση μιας σειράς εντολών, αλλά το αντίθετο μάλιστα. Είναι μια δυναμική και σύγχρονη γλώσσα, που υποστηρίζει εκτός των άλλων και το μοντέλο του αντικειμενοστραφούς προγραμματισμού. Χρησιμοποιείται κατά κύριο λόγο σε περιπτώσεις που υπάρχουν μεγάλα προγράμματα ή όταν ζητείται να επιλυθεί ένα ιδιαίτερο πρόβλημα σε ένα πρόγραμμα. Ο αντικειμενοστραφής προγραμματισμός απαιτεί τη χρήση κλάσεων (classes) και αντικείμενων (objects) για να κατασκευαστεί ένα πρόγραμμα. Εδώ, δεν εκτελείται απλά μια σειρά εντολών, όπως γινόταν μέχρι τώρα, αλλά στηρίζεται στο σύνολο των αντικειμένων και στις αλληλεπιδράσεις που έχουν μεταξύ τους. Έτσι, κατασκευάζονται προγράμματα με βάση αυτές τις αλληλεπιδράσεις.

4.1 Βασικές έννοιες του αντικειμενοστραφούς προγραμματισμού.

Ένα αντικειμενοστραφές πρόγραμμα βασίζεται σε κλάσεις και σε κάθε κλάση υπάρχει μια συλλογή από αλληλεπιδρώντα αντικείμενα. Κάθε αντικείμενο μπορεί να επεξεργάζεται δεδομένα και να λαμβάνει ή να στέλνει μηνύματα στα άλλα αντικείμενα. Η ανταλλαγή των δεδομένων μεταξύ των αντικειμένων, γίνεται τόσο αποτελεσματικά, ώστε να πετυχαίνουν πάντα τον επιθυμητό στόχο. Για να δημιουργήσει κάποιος, ένα αντικειμενοστραφές πρόγραμμα, πρέπει να γνωρίζει τις έννοιες που συναντώνται σε αυτή τη μέθοδο, όπως αυτές που περιγράφονται παρακάτω:

Κλάση (class): Μια κλάση περιγράφει αντικείμενα και καθορίζει τα βασικά χαρακτηριστικά (attributes) των αντικειμένων και των μεθόδων (methods) που θα χρησιμοποιηθούν. Εκτός από μεθόδους, μια κλάση περιέχει και πεδία (fields) που παίρνουν διάφορες τιμές και κάθε τιμή ορίζει και ένα στιγμιότυπο της κλάσης. Γενικά, μια κλάση είναι σαν ένα αντικείμενο και τα αντικείμενα που την απαρτίζουν είναι ένα στιγμιότυπο της.

Για να γίνει καλύτερα αντιληπτή η λειτουργία των κλάσεων, θα παρομοιαστούν με την παρασκευή ενός κέικ. Από τη μια πλευρά, είναι η συνταγή για το πώς θα δημιουργηθεί το κέικ και από την άλλη, το κέικ. Για να είναι έτοιμο, πρέπει πρώτα να ψηθεί, με βάση κάποιες οδηγίες. Όπως το ψήσιμο του γλυκού, έτσι και ένα αντικειμενοστραφές πρόγραμμα, περιέχει αντικείμενα σύμφωνα με τους όρους της κλάσης. Όπως με το κέικ, που χρειάζονται οι πρώτες ύλες και οι οδηγίες εκτέλεσης, έτσι και μια κλάση πρέπει να έχει μεταβλητές και μεθόδους. Υπάρχουν οι μεταβλητές κλάσης (class variables), που έχουν την ίδια τιμή σε όλες τις μεθόδους, αλλά και οι μεταβλητές αντικειμένων (instance variables), οι οποίες συνήθως έχουν διαφορετικές τιμές για διαφορετικά αντικείμενα. Σε μια κλάση, πρέπει να ορίζονται όλες οι απαραίτητες μέθοδοι, που θα χρησιμοποιηθούν.[11]

Αντικείμενα (objects): Παραπάνω αναφέρθηκε, πως οι κλάσεις περιγράφουν αντικείμενα, δηλαδή περιγραφές οι οποίες παίρνουν ένα όνομα για να τις αναγνωρίζει ο προγραμματιστής. Τα αντικείμενα μπορούν να περιέχουν απεριόριστα είδη δεδομένων. Για παράδειγμα, σε κλάση με το όνομα ζώα, ένα αντικείμενο της θα είναι ο σκύλος.

Στιγμιότυπο (Instance): Το στιγμιότυπο είναι η κατάσταση του αντικειμένου σε ένα συγκεκριμένο χρονικό σημείο κατά την εκτέλεση του προγράμματος.

[11] Python course (2011), Object oriented programming, Analogy: The Cake Class, Ανάκτηση στις 10-7-2011 από: http://www.python-course.eu/object_oriented_programming.php

Μέθοδος (Method): Η μέθοδος ορίζεται μέσα στην κλάση και δείχνει τη συμπεριφορά του αντικειμένου και καθορίζει τις δυνατότητες του.

Κληρονομικότητα (Inheritance): Κληρονομικότητα υπάρχει στην περίπτωση που μια κλάση είναι συνέχεια μιας άλλης κύριας κλάσης και έχουν κοινά χαρακτηριστικά μεταξύ τους. Έτσι, η κύρια κλάση κληροδοτεί τα στοιχεία της στην καινούρια. Αν για παράδειγμα σε μια κλάση που ονομάζεται ζώα, δημιουργηθεί μια υποκλάση γάτα, τότε η νέα κλάση θα κληρονομεί από τη βασική κλάση ζώα.[12]

4.2 Κλάσεις.

Οι κλάσεις και τα αντικείμενα είναι τα δύο βασικά συστατικά του αντικειμενοστραφούς προγραμματισμού. Ο μηχανισμός που χρησιμοποιεί η python για τις κλάσεις, είναι συνδυασμός του μηχανισμού της γλώσσας C++ και του Modula-3. Στην python οι κλάσεις έχουν όλα τα χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού. Όπως γίνεται στα modules, έτσι και οι κλάσεις συμβάλουν στη δυναμική δομή της python: δημιουργούνται τη στιγμή της εκτέλεσης και μπορούν να τροποποιηθούν αργότερα.

4.2.1 Δημιουργώντας μια κλάση.

Για να δημιουργηθεί μια κλάση, γράφεται το όνομα της, μετά από τη λέξη class. Στο block που ακολουθεί εισάγονται οι δηλώσεις, που θα έχει η κλάση[13]. Είναι αρκετά απλή διαδικασία, όπως φαίνεται στο παράδειγμα που ακολουθεί:

```
>>> class teipat():
…       pass
…
>>> t = teipat()
```

Στο παραπάνω παράδειγμα, με το class teipat(): κατασκευάστηκε μια κλάση με όνομα teipat. Με τη δήλωση pass δηλώνεται ότι το block της κλάσης είναι άδειο και η python δεν εκτελεί κάτι στην ουσία, αφού είναι μηδενική λειτουργία.[14] Στη συνέχεια με τη δήλωση t = teipat() δημιουργείται ένα αντικείμενο. Σε επόμενο παράδειγμα θα αναλυθεί, πως εισάγεται μια μέθοδος στην κλάση, που θα τυπώνει ένα μήνυμα όταν καλείται.

Εικόνα 73 – Δημιουργία μιας κενής κλάσης.

Όπως στις συναρτήσεις, που χρησιμοποιούταν η λέξη def για να οριστεί μια συνάρτηση, έτσι κι εδώ, τον ορισμό μιας μεθόδου. Οι μέθοδοι μοιάζουν με τις συναρτήσεις και ορίζονται μέσα στο block της κλάσης. Όταν ορίζεται μια μέθοδος ενός αντικειμένου όμως, μπαίνει ως πρώτο επιχείρημα στην παρένθεση, η παράμετρος self. Με το self εννοείται

[12] Λεβεντέας Δ. (2010), «Εκμάθηση python βήμα βήμα», (σελ. 95), Ελληνική κοινότητα Προγραμματιστών python – Ανάκτηση στις 12-7-2011 από http://python.org.gr/index.php?option=com_phocadownload&view=category&id=3:tutorials&Itemid=58

[13] Στο block της κλάσης πρέπει να υπάρχουν υποχρεωτικά μερικά κενά πριν γραφούν οι δηλώσεις, αλλιώς θα εμφανίσει σφάλμα.

[14] Κέντρο ΠΛΗ.ΝΕ.Τ.Ν.Φλώρινας, Η γλώσσα προγραμματισμού python – Εισαγωγή, Ανάκτηση στις 15-7-2011 από http://dide.flo.sch.gr/Plinet/Tutorials/Tutorials-Python-Introduction.html.

το αντικείμενο (στιγμιότυπο), για το οποίο καλείται κάθε φορά η μέθοδος. Η εκτέλεση του προγράμματος δεν επηρεάζεται, από το όνομα της παραμέτρου που χρησιμοποιήθηκε και δεν έχει σημασία αν είναι ο όρος self ή κάποιος άλλος αντί αυτού. Στην πραγματικότητα όμως, επειδή ο όρος αυτός είναι ευρέως αναγνωρίσιμος στην python, καλό είναι να μην χρησιμοποιείται άλλο όνομα. Ίσως άλλοι προγραμματιστές, που θα διαβάσουν τον κώδικα, να μπερδευτούν.

Τα αντικείμενα των κλάσεων υποστηρίζουν την αρχικοποίηση και την αναφορά ιδιοτήτων – χαρακτηριστικών (attributes). Αυτά θα εξηγηθούν στο παράδειγμα που ακολουθεί:

```
>>> class klasi:
…    def f(self):
...     print(‘Γεια σας’)
...
```

Εδώ, δημιουργήθηκε μια κλάση με το όνομα klasi και ορίστηκε η συνάρτηση f, που τυπώνει το μήνυμα ‘Γεια σας’. Αν εκτελεστεί το όνομα της κλάσης, θα εμφανίσει τη θέση που είναι αποθηκευμένη η κλάση. Είναι στο module __main__ και στη συγκεκριμένη θέση μνήμης του υπολογιστή 0x0000000001FE37C8.

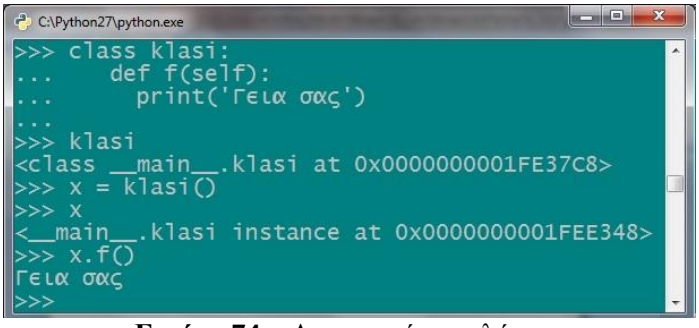
```
>>> klasi
<class __main__.klasi at 0x0000000001FE37C8>
```

Για την αρχικοποίηση της κλάσης, χρησιμοποιούνται σύμβολα, από τις συναρτήσεις και δημιουργείται ένα άδειο αντικείμενο. Γίνεται η υπόθεση, ότι το αντικείμενο της κλάσης είναι μια συνάρτηση χωρίς παραμέτρους, η οποία επιστρέφει ένα νέο στιγμιότυπο της κλάσης. Για παράδειγμα η παρακάτω εντολή:

```
>>> x = klasi()
>>> x
<__main__.klasi instance at 0x0000000001FEE348>
```

Δημιουργήθηκε ένα στιγμιότυπο (instance) της κλάσης και ορίστηκε το αντικείμενο στην τοπική μεταβλητή x. Στην επαλήθευση που γίνεται, εμφανίζει μήνυμα που λέει ότι το x είναι στιγμιότυπο της κλάσης klasi και τη θέση που έχει αποθηκευτεί. Οι αναφορές ιδιοτήτων, χρησιμοποιούν δεδομένη σύνταξη με το όνομα του αντικειμένου ακολουθούμενο από τελεία και το όνομα του attribute. Το x.f είναι μια έγκυρη αναφορά ιδιότητας και επιστρέφει το αποτέλεσμα της συνάρτησης.

```
>>> x.f()
Γεια σας
```



Εικόνα 74 – Αρχικοποίηση κλάσης.

4.3 Ειδικές μέθοδοι στα αντικείμενα.

Στην python υπάρχουν οι ειδικές μέθοδοι που έχουν εφαρμογή στα αντικείμενα των κλάσεων. Οι περισσότερες από τις ειδικές μεθόδους είναι οι ίδιες με τις λειτουργίες που υπάρχουν στις ακολουθίες, όπως για παράδειγμα η συνάρτηση len(), που εδώ γίνεται __len__(). Οι ειδικές μέθοδοι, γράφονται με διπλή κάτω παύλα πριν και μετά το όνομα και

μέσα σε παρένθεση οι παράμετροι. Όλες οι ειδικές μέθοδοι βρίσκονται στη βιβλιοθήκη της python και καλό είναι να μη δημιουργούνται νέες από το χρήστη, αλλά να χρησιμοποιούνται μόνο αυτές που παρέχονται από τη γλώσσα.

Ο παρακάτω πίνακας αναφέρει τις βασικότερες ειδικές μεθόδους που υπάρχουν στις κλάσεις.[15]

Πίνακας 2: Βασικές ειδικές μέθοδοι των κλάσεων.	
Όνομα μεθόδου	Περιγραφή μεθόδου
__init__(self,…)	Δηλώνει τα χαρακτηριστικά των αντικειμένων της κλάσης.
__str__(self)	Χρησιμοποιείται για τη συνάρτηση print() ή str().
__del__(self)	Χρησιμοποιείται για να καταστραφεί ένα αντικείμενο.
__lt__(self, other) __le__(self, other) __eq__(self, other) __ne__(self, other) __gt__(self, other) __ge__(self, other) κλπ.	Για να γίνει σύγκριση με ένα άλλο αντικείμενο, χρησιμοποιείται και ο αντίστοιχος τελεστής. (<,<=,==,!=,> και >=). Υπάρχουν ειδικές μέθοδοι για όλους τους τελεστές (+,-,*,/ κλπ.).
__getitem__(self,kleidi)	Καλείται για να βρεθεί μια λέξη κλειδί.
__len__(self)	Χρησιμεύει σε ακολουθίες, για τον εντοπισμό του μήκους της ακολουθίας (συνάρτηση len).

Υπάρχουν, μερικές ειδικές μέθοδοι για την εκτέλεση αριθμητικών πράξεων:

Πίνακας 3: Ειδικές μέθοδοι για αριθμητικές πράξεις.	
__add__(a,b)	Πρόσθεση **a+b**
__sub__(a,b)	Αφαίρεση **a-b**
__mul__(a,b)	Πολλαπλασιασμός **a*b**
__truediv__(a,b)	Διαίρεση **a/b**
__floordiv__(a,b)	Ακέραια Διαίρεση **a//b**
__mod__(a,b)	Υπόλοιπο διαίρεσης **a%b**
__pow__(a,b)	Ύψωση σε δύναμη **a**b**
__neg__ (a,b)	Αρνητικός αριθμός **–a**

Οι ειδικές μέθοδοι έχουν εφαρμογή στις κλάσεις και όχι στα στιγμιότυπα των αντικειμένων. Η κλήση μιας ειδικής μεθόδου, γίνεται ως εξής: όνομα-κλάσης. __ειδική-μέθοδος __ (αντικείμενα).

4.3.1 Ειδική μέθοδος __init__().

Πολλές κλάσεις συνηθίζεται να δημιουργούν αντικείμενα με περιπτώσεις προσαρμοσμένες σε μια αρχική δήλωση. Για το λόγο αυτό, υπάρχει μια πιο εξειδικευμένη μέθοδος που ονομάζεται __init__(). Καλείται όταν ένα αντικείμενο αρχικοποιείται και δηλώνει τα χαρακτηριστικά που πρέπει να έχει το αντικείμενο. Αυτή η μέθοδος έχει εφαρμογή σε όλα τα αντικείμενα της κλάσης και σε όλες τις υποκλάσεις αυτής. Στην πρώτη

[15] Λεβεντέας Δ. (2010), «Εκμάθηση python βήμα βήμα», (σελ. 107-109), Ελληνική Κοινότητα Προγραμματιστών python – Ανάκτηση στις 10-7-2011 από http://python.org.gr/index.php?option=com_phocadownload&view=category&id=3:tutorials&Itemid=58

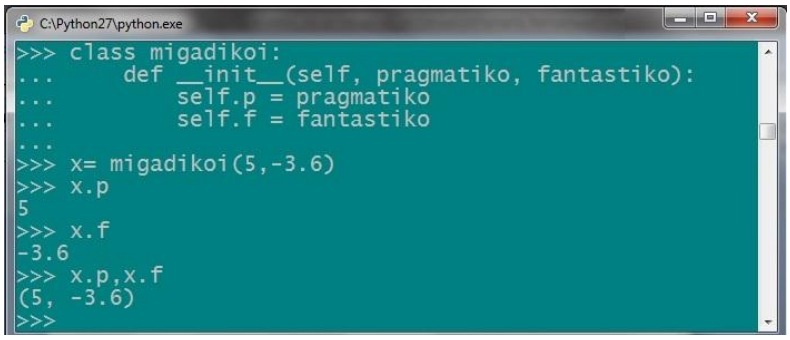
θέση των παραμέτρων μπαίνει το self και με αυτή τη λέξη προσδιορίζεται, η συμπεριφορά του εκάστοτε αντικειμένου. Στις υπόλοιπες θέσεις, προστίθενται τα χαρακτηριστικά (attributes) του αντικειμένου: (self,…)[16].

Σε κλάση με μιγαδικούς αριθμούς, πρέπει να δηλώνεται το πραγματικό και το φανταστικό μέρος του μιγαδικού. Δημιουργείται η κλάση migadikoi και με τη μέθοδο __init__(), δηλώνονται τα δύο χαρακτηριστικά των μιγαδικών, που είναι το πραγματικό και το φανταστικό μέρος. Όταν ένα αντικείμενο δημιουργείται, θα είναι ένας μιγαδικός αριθμός και κάθε μιγαδικός, θα αποτελείται από το πραγματικό (self.p) και το φανταστικό μέρος (self.f).

```
>>> class migadikoi:
...     def __init__(self, pragmatiko, fantastiko):
...         self.p = pragmatiko
...         self.f = fantastiko
...
```

Οι αναφορές ιδιοτήτων, όπως αναφέρθηκε παραπάνω, θα είναι οι ιδιότητες pragmatiko και fantastiko, που δηλώνονται στην παρένθεση μετά το self. Στη συνέχεια, στη μεταβλητή αντικειμένου self.p, δηλώνεται η ιδιότητα pragmatiko και στο self.f το fantastiko μέρος του μιγαδικού. Αφού οριστεί η κλάση, δημιουργείται η μεταβλητή x, που θα είναι αντικείμενο της κλάσης migadikoi και σε παρένθεση εισάγονται οι τιμές παραμέτρων, δηλαδή το πραγματικό και το φανταστικό μέρος του μιγαδικού αριθμού. Σε αυτό το παράδειγμα, το πραγματικό μέρος του μιγαδικού αριθμού είναι το 5 και το φανταστικό του μέρος το -3,6.

```
>>> x= migadikoi(5,-3.6)
>>> x.p
5
>>> x.f
-3.6
>>> x.p, x.f
(5, -3.6)
```



Εικόνα 75 – Δημιουργία κλάσης για μιγαδικούς αριθμούς.

Άλλο ένα ενδεικτικό παράδειγμα στις κλάσεις αποτελεί το παρακάτω, που θα δημιουργηθεί μια κλάση και θα παίρνει ως αντικείμενα τα τμήματα του ΤΕΙ Πάτρας:

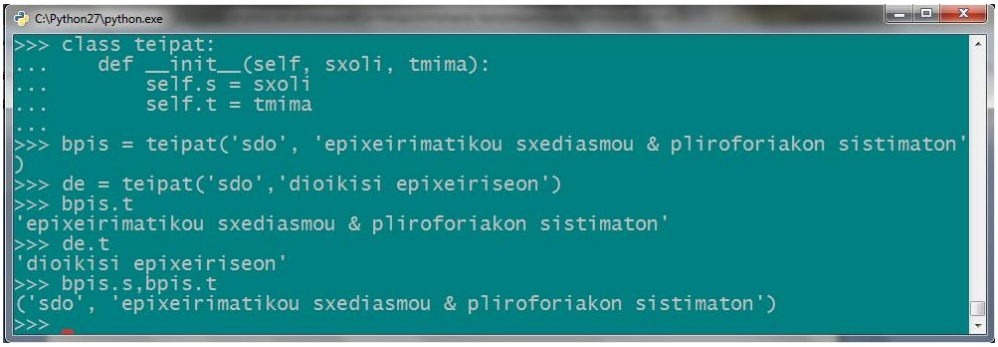
```
>>> class teipat:
...     def __init__(self, sxoli, tmima):
...         self.s = sxoli
```

[16] Το self επιτρέπει, να εφαρμοστούν τα χαρακτηριστικά, που ορίζονται στην αρχικοποίηση της κλάσης με το __init__(), σε όλα τα αντικείμενα που υπάγονται σε αυτή τη κλάση, χωρίς να πρέπει ο χρήστης να ορίσει από πριν τα αντικείμενα του.

```
...         self.t = tmima
...
```

Στο παραπάνω κομμάτι ορίστηκε η κλάση και οι ιδιότητες (attributes) των αντικειμένων της κλάσης, που είναι η σχολή και το τμήμα. Το αντικείμενο bpis που δημιουργείται, έχει ως ιδιότητες τη σχολή ΣΔΟ και το τμήμα επιχειρηματικού σχεδιασμού και πληροφοριακών συστημάτων. Το αντικείμενο de αντίστοιχα ανήκει στη σχολή ΣΔΟ και το τμήμα διοίκησης επιχειρήσεων. Για να προσπελαστούν οι ιδιότητες των αντικείμενων, δηλαδή τα attributes, χρησιμοποιούνται ως μεταβλητές των αντικειμένων.

```
>>> bpis = teipat('sdo', 'epixeirimatikou sxediasmou & pliroforiakon sistimaton')
>>> de = teipat('sdo','dioikisi epixeiriseon')
>>> bpis.t
'epixeirimatikou sxediasmou & pliroforiakon sistimaton'
>>> de.t
'dioikisi epixeiriseon'
>>> bpis.s,bpis.t
('sdo', 'epixeirimatikou sxediasmou & pliroforiakon sistimaton')
```



Εικόνα 76 – Παράδειγμα με τα τμήματα.

Για περεταίρω διερεύνηση των κλάσεων, θα αναλυθεί η παρακάτω περίπτωση. Θα δημιουργηθεί μια κλάση με πληροφορίες για τους σπουδαστές του τμήματος επιχειρηματικού σχεδιασμού και πληροφοριακών συστημάτων (bpis). Για κάθε σπουδαστή θα υπάρχουν συγκεκριμένες πληροφορίες όπως: το όνομα, το επώνυμο, τον αριθμό μητρώου και το εξάμηνο φοίτησης. Έτσι, θα δημιουργηθεί η παρακάτω κλάση που θα είναι το καλούπι για την δημιουργία αντικειμένων – φοιτητών.

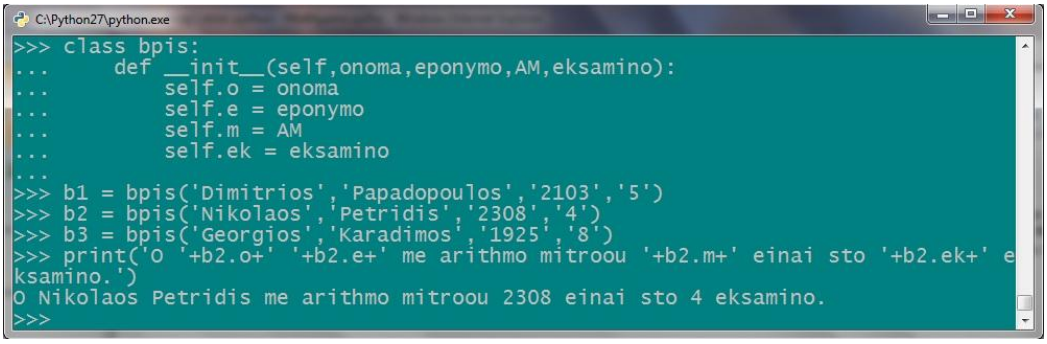
```
>>> class bpis:
...     def __init__(self,onoma,eponymo,AM,eksamino):
...         self.o = onoma
...         self.e = eponymo
...         self.m = AM
...         self.ek = eksamino
...
```

Ο παραπάνω κώδικας θα εκτελείται κάθε φορά, που δημιουργείται ένα νέο αντικείμενο φοιτητή. Όταν καλείται, θα λαμβάνει ως ιδιότητες του αντικειμένου, τα στοιχεία του εκάστοτε φοιτητή. Για τον πρώτο φοιτητή, εισάγονται τα στοιχεία του στο αντικείμενο b1. Ομοίως και στα αντικείμενα b2 και b3.

```
>>> b1 = bpis('Dimitrios','Papadopoulos','2103','5')
>>> b2 = bpis('Nikolaos','Petridis','2308','4')
>>> b3 = bpis('Georgios','Karadimos','1925','8')
```

Για την εμφάνιση μηνύματος με τα στοιχεία ενός φοιτητή, θα γραφτεί σχετικό μήνυμα μετά την εντολή print, βάζοντας το + ανάμεσα στις μεταβλητές και το μήνυμα.

>>> print('O '+b2.o+' '+b2.e+' me arithmo mitroou '+b2.m+' einai sto '+b2.ek+' eksamino.')
O Nikolaos Petridis me arithmo mitroou 2308 einai sto 4 eksamino.



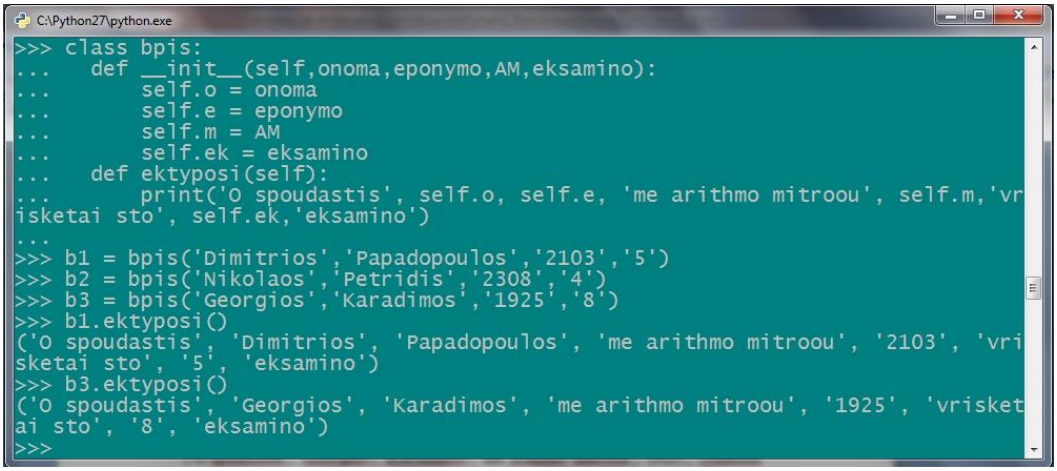
Εικόνα 77 – Παράδειγμα με φοιτητές και τα στοιχεία τους.

Υπάρχει επίσης η δυνατότητα, να δημιουργηθεί μια συνάρτηση μέσα στην κλάση, που να εκτυπώνει τα απαιτούμενα δεδομένα. Έτσι, δεν θα χρειάζεται κάθε φορά να ξαναγράφεται το αντίστοιχο μήνυμα.

```
>>> class bpis:
...   def __init__(self,onoma,eponymo,AM,eksamino):
...       self.o = onoma
...       self.e = eponymo
...       self.m = AM
...       self.ek = eksamino
...   def ektyposi(self):
...       print('O spoudastis', self.o, self.e, 'me arithmo mitroou', self.m,'vrisketai
sto', self.ek,'eksamino')
...
>>> b1 = bpis('Dimitrios','Papadopoulos','2103','5')
>>> b2 = bpis('Nikolaos','Petridis','2308','4')
>>> b3 = bpis('Georgios','Karadimos','1925','8')
```

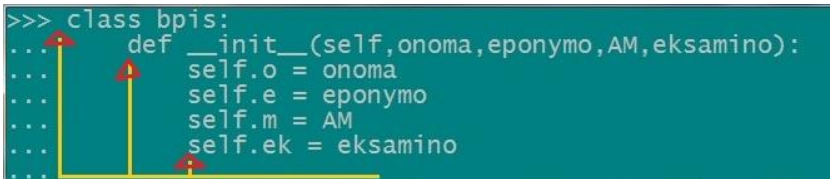
Δημιουργήθηκε η συνάρτηση ektyposi(self), η οποία όποτε καλείται, θα εμφανίζει το παραπάνω μήνυμα. Για να γίνει η εκτύπωση των attributes (ιδιοτήτων) ενός αντικειμένου, θα εκτελεστεί το όνομα του αντικειμένου, ακολουθούμενο από το όνομα της συνάρτησης, όπως φαίνεται παρακάτω.

```
>>> b1.ektyposi()
('O spoudastis', 'Dimitrios', 'Papadopoulos', 'me arithmo mitroou', '2103', 'vrisketai
sto', '5', 'eksamino')
>>> b3.ektyposi()
('O spoudastis', 'Georgios', 'Karadimos', 'me arithmo mitroou', '1925', 'vrisketai sto',
'8', 'eksamino')
```



Εικόνα 78 – Δημιουργία συνάρτησης εκτύπωσης.

Όπως φάνηκε στα παραπάνω παραδείγματα, για να οριστεί μια μέθοδος της κλάσης, δεν εισάγεται το def ακριβώς κάτω από το class, αλλά μεσολαβούν μερικοί κενοί χαρακτήρες πρώτα. Το ίδιο ισχύει και για τις ιδιότητες των αντικειμένων.[17]



Εικόνα 79 – Ορθή συγγραφή μιας κλάσης.

4.3.2 Άλλες ειδικές μέθοδοι.

Η python έχει αποθηκευμένη μια πλούσια συλλογή από ειδικές μεθόδους όπως αναφέρθηκε, με βασικότερη την __init__(). Όλες όμως έχουν την χρησιμότητα τους, ανάλογα με την περίπτωση.

Ένα παράδειγμα στο οποίο θα αναλυθούν μερικοί άλλοι ειδικοί τελεστές στην πράξη, είναι το παρακάτω. Δημιουργείται μια κλάση, στην οποία τα αντικείμενα θα δέχονται πληροφορίες για αυτοκίνητα. Αν κάποιο από τα αυτοκίνητα αυτά, είναι χαμηλών κυβικών (μέχρι 1000 κ.ε.) θα διαγράφεται. Θα γίνουν και άλλες πράξεις με τα αντικείμενα αυτά, όπως σύγκριση, πρόσθεση κλπ.

```
>>> class auto:
...     def __init__(self,marka,montelo,kybika):
...         self.m = marka
...         self.mo = montelo
...         self.k = kybika
...         if self.k <= 1000:
...             print('To',self.m,self.mo,'aporripetai logo xamilon kybikon')
```

Αν προστεθεί ένα αυτοκίνητο μέχρι 1000 κ.ε., τότε θα εμφανίζεται μήνυμα να για τη διαγραφή του. Ορίζονται στη συνέχεια και οι ειδικές μέθοδοι, όπως το del, η ισότητα, η πρόσθεση και η συνάρτηση str().

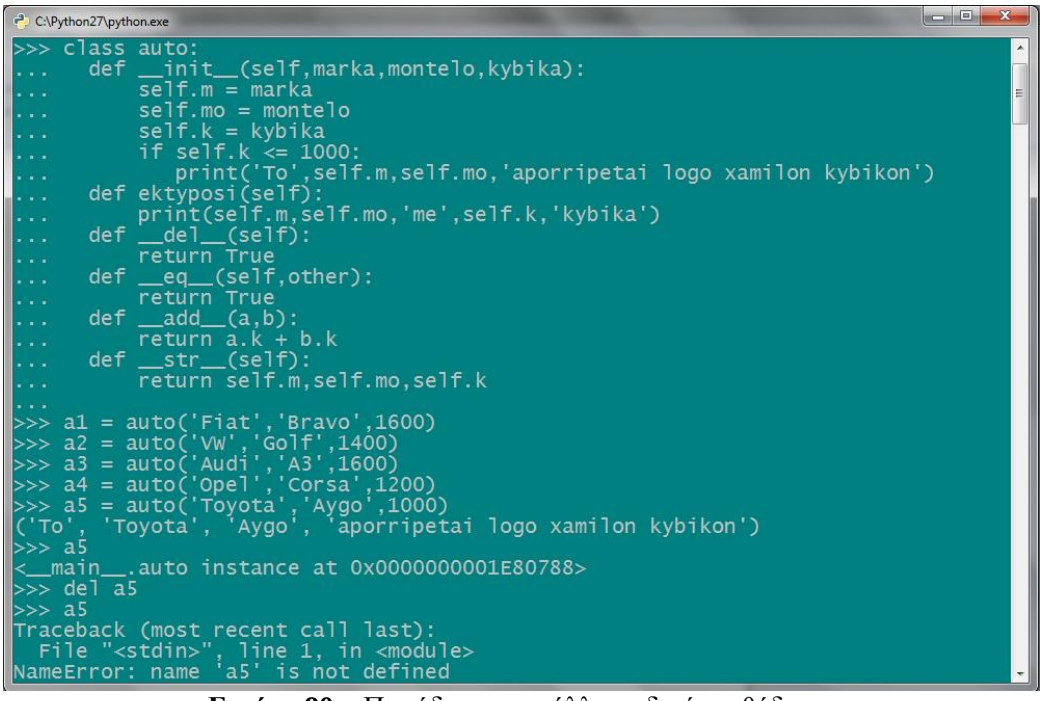
```
...     def ektyposi(self):
...         print(self.m,self.mo,'me',self.k,'kybika')
```

[17] Όλες οι μέθοδοι της κλάσης θα γράφονται πάντα στην ίδια ευθεία και όχι πιο μέσα, δηλαδή εκεί που είναι το μεσαίο βέλος.

```
...   def __del__(self):
...       return True
...   def __eq__(self,other):
...       return True
...   def __add__(a,b):
...       return a.k + b.k
...   def __str__(self):
...       return self.m, self.mo, self.k
...
>>> a1 = auto('Fiat','Bravo',1600)
>>> a2 = auto('VW','Golf',1400)
>>> a3 = auto('Audi','A3',1600)
>>> a4 = auto('Opel','Corsa',1200)
>>> a5 = auto('Toyota','Aygo',1000)
('To', 'Toyota', 'Aygo', 'aporripetai logo xamilon kybikon')
```

Για το αντικείμενο a5, εμφανίζει μήνυμα απόρριψης και πρέπει να καταστραφεί. Εκτελείται κατευθείαν η εντολή del, μαζί με το όνομα του αντικειμένου.

```
>>> a5
<__main__.auto instance at 0x0000000001E80788>
>>> del a5
>>> a5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a5' is not defined
```



Εικόνα 80 – Παράδειγμα με άλλες ειδικές μεθόδους.

Ενώ στην αρχή το αντικείμενο a5 υπήρχε, όπως δείχνει η επαλήθευση, μετά την εκτέλεση του del, διεγράφη από τη μνήμη του υπολογιστή. Για να γίνει σύγκριση, χρησιμοποιείται κατευθείαν η ισότητα (ή ότι άλλος τελεστής σύγκρισης απαιτείται) και εμφανίζεται το μήνυμα True ή False, ανάλογα με το αν ισχύει ή όχι η ισότητα.

```
>>> 1200 == a1.k
```

False
>>> a2.k == 1400
True

Με το __add__ προσθέτει τις τιμές των προκαθορισμένων χαρακτηριστικών δύο αντικειμένων που θα δηλώνονται στην παρένθεση. Έχει οριστεί να προσθέτει τα κυβικά δύο αυτοκινήτων. Για να κληθεί η add, θα μπει πρώτα το όνομα της κλάσης και στη συνέχεια η μέθοδος. Εδώ, θα προσθέσει: 1600 + 1200.

>>> auto.__add__(a1,a4)
2800

Το __str__ θα εκτυπώσει τα χαρακτηριστικά του αντικειμένου, που δηλώθηκαν στο όρισμα της. Παρατηρείται ότι το αποτέλεσμα του είναι παρόμοιο, με εκείνο της μεθόδου ektyposi().

>>> auto.__str__(a2)
('VW', 'Golf', 1400)
>>> a3.ektyposi()
('Audi', 'A3', 'me', 1600, 'kybika')

Εικόνα 81 - Παράδειγμα με άλλες ειδικές μεθόδους (..συνέχεια).

4.4 Μεταβλητές κλάσεων.

Μέχρι τώρα έχουν αναλυθεί οι μέθοδοι κλάσεων και αντικειμένων και οι μεταβλητές των αντικειμένων, αλλά δεν πρέπει να παραλειφτεί η αναφορά και τις μεταβλητές των κλάσεων. Αυτές οι μεταβλητές έχουν εφαρμογή μέσα στη κλάση και τα αντικείμενα της. Μια τέτοια περίπτωση, που χρησιμοποιούνται μεταβλητές μέσα στις κλάσεις, ως συνέχεια του προηγούμενου παραδείγματος με τους φοιτητές, είναι όταν γίνεται μέτρηση των καταχωρημένων φοιτητών στο τμήμα.[18]

>>> class bpis:
... c = 0

Μέσα στο block της κλάσης και πριν δηλωθούν οι μεταβλητές της, εισάγεται ένας μετρητής c, που παίρνει την τιμή μηδέν. Εδώ, η c είναι μεταβλητή της κλάσης bpis. Επειδή η c είναι μεταβλητή κλάσης, όταν την χρησιμοποιείται, θα καλείται ως bpis.c. Με το += 1, θα αυξάνεται η τιμή της κατά 1.

... def __init__(self,onoma,eponymo,AM,eksamino):
... self.o = onoma
... self.e = eponymo
... self.m = AM
... self.ek = eksamino
... bpis.c += 1
... def ektyposi(self):

[18] Λεβεντέας Δ. (2010), «Εκμάθηση python βήμα βήμα», (σελ. 103), Ελληνική κοινότητα Προγραμματιστών python – Ανάκτηση στις 16-7-2011 από http://python.org.gr/index.php?option=com_phocadownload&view=category&id=3:tutorials&Itemid=58

... print('O spoudastis', self.o, self.e, 'me arithmo mitroou', self.m,'vrisketai sto', self.ek,'eksamino')

Η μέθοδο plithos στη συνέχεια, θα εμφανίζει μήνυμα με το πλήθος των φοιτητών. Επειδή όμως η συγκεκριμένη μέθοδος χρησιμοποιεί τη μεταβλητή της κλάσης, θα πρέπει να μετατραπεί σε στατική μέθοδο, καλώντας την ενσωματωμένη συνάρτηση της python staticmethod. Περισσότερες λεπτομέρειες για τις στατικές μεθόδους, βρίσκονται στην ενότητα που ακολουθεί.

```
... @staticmethod
... def plithos():
...     print('To tmima bpis exei',bpis.c,'spoudastes')
...
>>> b1 = bpis('Dimitrios','Papadopoulos','2103','5')
>>> b2 = bpis('Nikolaos','Petridis','2308','4')
>>> b3 = bpis('Georgios','Karadimos','1925','8')
>>> b1.ektyposi()
('O spoudastis', 'Dimitrios', 'Papadopoulos', 'me arithmo mitroou', '2103', 'vrisketai sto', '5', 'eksamino')
>>> bpis.plithos()
('To tmima bpis exei', 3, 'spoudastes')
```

Εικόνα 82 – Παράδειγμα με μεταβλητές κλάσεων.

4.5 Στατικές μέθοδοι – Προσανατολισμένο μοντέλο προγραμματισμού.

Η python υποστηρίζει το προσανατολισμένο μοντέλο προγραμματισμού (AOP – Aspect Oriented Programming) και σε αυτό το μοντέλο μπορεί ο προγραμματιστής να μετατρέψει μια κλάση ή τα στιγμιότυπα της δυναμικά, κατά τη διάρκεια εκτέλεσης του προγράμματος. Οι στατικές μέθοδοι δεν ασχολούνται με τα στιγμιότυπα, αλλά λειτουργούν στο επίπεδο της κλάσης. Ένα τέτοιο παράδειγμα είναι η στατική μέθοδος. Μια στατική μέθοδος ορίζεται με τη συνάρτηση staticmethod ή με τον διακοσμητή (decorator), που είναι της μορφής @staticmethod. Αξίζει να σημειωθεί, ότι υπάρχει και η συνάρτηση classmethod, που κάνει την ίδια δουλειά και χρησιμοποιείται, όταν γίνεται αναφορά σε συγκεκριμένη κλάση. Στο παράδειγμα που εφαρμόστηκε το staticmethod, θα μπορούσε να είχε μπει και το classmethod.

```
... def plithos():
...     print('To tmima bpis exei',bpis.c,'spoudastes')
```

... plithos = staticmethod(plithos)[19]

```
...     def plithos():
...         print('To tmima bpis exei',bpis.c,'spoudastes')
...     plithos = staticmethod(plithos)
...
```

Εικόνα 83 – Ορθή σύνταξη μιας στατικής μεθόδου.

Υπάρχουν και οι διακοσμητές, οι οποίοι είναι ένα είδος συντόμευσης της διαδικασίας, που γίνεται με το staticmethod. Με χρήση διακοσμητή η διαδικασία θα γίνει ως εξής:

```
... @staticmethod
... def plithos():
...     print('To tmima bpis exei',bpis.c,'spoudastes')
```

```
...     @staticmethod
...     def plithos():
...         print('To tmima bpis exei',bpis.c,'spoudastes')
...
```

Εικόνα 84 – Δημιουργία στατικής μεθόδου με χρήση διακοσμητή.

4.6 Κληρονομικότητα.

Η κληρονομικότητα είναι βασικό χαρακτηριστικό, του αντικειμενοστραφούς προγραμματισμού και είναι αναμφίβολα χρήσιμο. Περιορίζει τη μεγάλη έκταση των προγραμμάτων, λόγω της επαναχρησιμοποίησης του κώδικα και μειώνει το χρόνο για την συγγραφή ενός προγράμματος.

Η κληρονομικότητα, επιτρέπει τη δημιουργία κλάσεων, που παίρνουν τα χαρακτηριστικά τους από άλλες κλάσεις. Η κλάση που κληροδοτεί στοιχεία σε άλλες ονομάζεται υπερκλάση ή βασική κλάση. Αντίστοιχα, η κλάση που παίρνει στοιχεία από μια άλλη, θα ονομάζεται υποκλάση. Ένα αντικείμενο μπορεί να χρησιμοποιείται ως μέλος από πολλές κλάσεις, ενώ έχει οριστεί μόνο σε μια. Η μέθοδος της υποκλάσης μπορεί επίσης, να επεκτείνει μια μέθοδο της αρχικής κλάσης ή να την αντικαταστήσει. Στο παράδειγμα που ακολουθεί, αν και είναι μακροσκελές εξηγεί στην πράξη, πως γίνονται όλα αυτά.

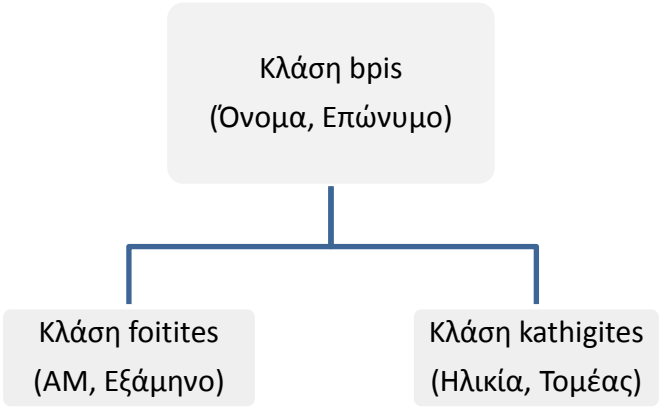
Νωρίτερα, αναλύθηκε το παράδειγμα της κατασκευής μιας κλάσης, για τους φοιτητές του τμήματος επιχειρηματικού σχεδιασμού και πληροφοριακών συστημάτων. Με βάση την κληρονομικότητα, θα αναπτυχθεί το παράδειγμα αυτό και για τους καθηγητές, που διδάσκουν στο συγκεκριμένο τμήμα. Στην αρχική φάση, υπήρχε η κλάση bpis, που είχε ως αντικείμενα τους σπουδαστές του τμήματος. Εδώ όμως, η κλάση αυτή θα περιέχει αντικείμενα φοιτητών και καθηγητών του τμήματος. Τα πεδία των αντικειμένων θα είναι διαφορετικά για κάθε περίπτωση. Για τους φοιτητές θα υπάρχουν τα πεδία: όνομα, επώνυμο, αριθμό μητρώου και εξάμηνο φοίτησης. Για τους καθηγητές από την άλλη, χρειάζονται πεδία όπως όνομα, επώνυμο, ηλικία και τον τομέα που ανήκει κάθε διδάσκοντας: τομέας πληροφοριακών συστημάτων ή λήψης επιχειρηματικών αποφάσεων. Πέρα από τα κοινά τους χαρακτηριστικά όμως, το όνομα και το επώνυμο, όλα τα υπόλοιπα είναι ιδιαίτερα για κάθε κατηγορία και χωρίς την κληρονομικότητα δεν γίνεται να συνυπάρξουν.

Έτσι, θα υπάρχει η κλάση bpis, που είναι η βασική και οι υπόλοιπες θα είναι δευτερεύουσες. Οι κλάσεις kathigites και foitites που αντιστοιχούν στις δύο κατηγορίες, θα

[19] Το plithos = staticmethod(plithos) γράφεται στο ίδιο ύψος με το def της μεθόδου και όχι ακριβώς κάτω από το print.

κληρονομούν από την υπερκλάση. Επίσης, οι κοινές μεταβλητές αντικειμένου (attributes), όνομα και επώνυμο, θα ανήκουν στην bpis και θα κληροδοτούνται στις υπόλοιπες.

Σχεδιάγραμμα: Η βασική κλάση με τις δυο υποκλάσεις της.




```
>>> class bpis:
...   def __init__(self,onoma,eponymo):
...       self.o = onoma
...       self.e = eponymo
...   def ektyposi(self):
...       print('O', self.o, self.e,'vrisketai sto tmima ep.sxediasmou & pl.syst.')
...
```

Κατασκευάζεται αρχικά η κλάση bpis, που είναι και η βασική, εισάγοντας μόνο το ονοματεπώνυμο στις ιδιότητες της κλάσης. Στη συνέχεια ορίζεται και η συνάρτηση της εκτύπωσης, για αυτά τα attributes του αντικειμένου.

Όσο για την υποκλάση των καθηγητών, θα μπει το bpis στην παρένθεση, αντί του self, επειδή αυτή είναι η βασική της κλάση. Παρακάτω δηλώνεται, ότι τις μεταβλητές όνομα και επώνυμο θα τις πάρει από τη βασική κλάση.

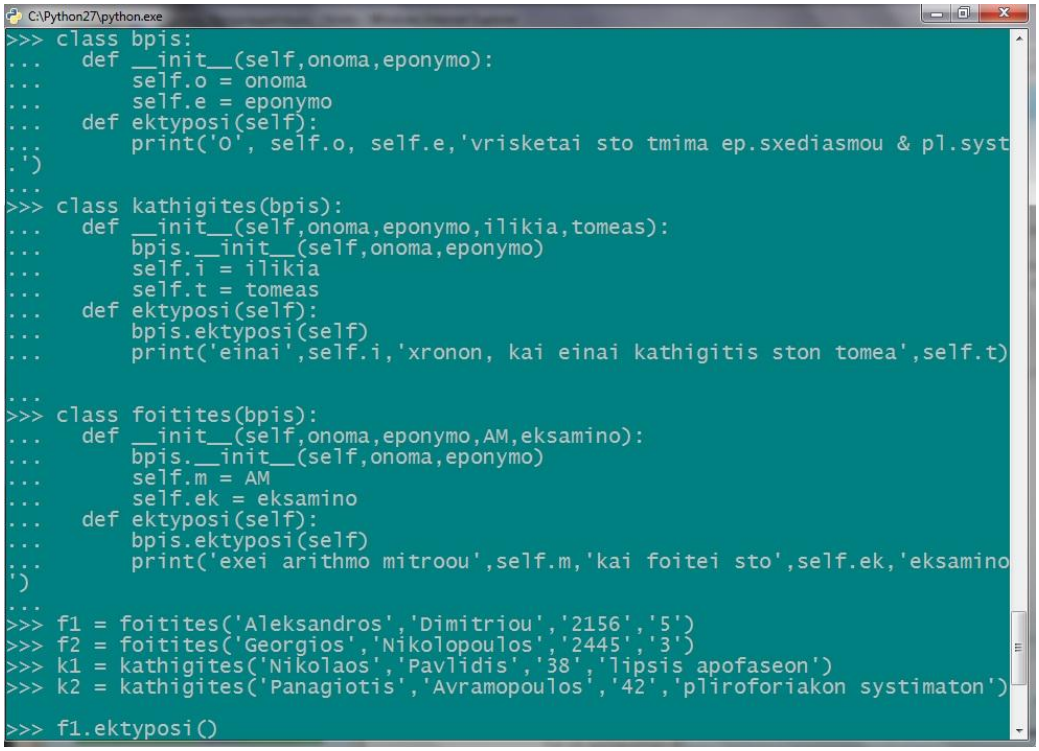
```
>>> class kathigites(bpis):
...   def __init__(self,onoma,eponymo,ilikia,tomeas):
...       bpis.__init__(self,onoma,eponymo)
...       self.i = ilikia
...       self.t = tomeas
...   def ektyposi(self):
...       bpis.ektyposi(self)
...       print('einai',self.i,'xronon, kai einai kathigitis ston tomea',self.t)
...
```

Στη συνάρτηση της εκτύπωσης, καλείται η αντίστοιχη συνάρτηση της υπερκλάσης και γράφεται το αντίστοιχο μήνυμα, για να εμφανίσει τις ιδιότητες των αντικειμένων, που ανήκουν στην υποκλάση. Όταν γίνεται εκτύπωση στην υποκλάση, θα καλείται η συνάρτηση της βασικής κλάσης και θα επιστρέφονται δύο μηνύματα στο τέλος, που θα αφορούν όλα τα χαρακτηριστικά του αντικειμένου. Η μέθοδος ektyposi στις υποκλάσεις είναι προέκταση της αντίστοιχης, στη βασική κλάση. Ομοίως, θα κατασκευαστεί και η δεύτερη υποκλάση των φοιτητών.

```
>>> class foitites(bpis):
```

```
...         def __init__(self,onoma,eponymo,AM,eksamino):
...           bpis.__init__(self,onoma,eponymo)
...           self.m = AM
...           self.ek = eksamino
...         def ektyposi(self):
...           bpis.ektyposi(self)
...           print('exei arithmo mitroou',self.m,'kai foitei sto',self.ek,'eksamino')
...
>>> f1 = foitites('Aleksandros','Dimitriou','2156','5')
>>> f2 = foitites('Georgios','Nikolopoulos','2445','3')
>>> k1 = kathigites('Nikolaos','Pavlidis','38','lipsis apofaseon')
>>> k2 = kathigites('Panagiotis','Avramopoulos','42','pliroforiakon systimaton')
```

Αφού τελειώσει η διαδικασία, καταχωρούνται μερικά αντικείμενα φοιτητών και καθηγητών για να τα εκτυπωθούν παρακάτω.

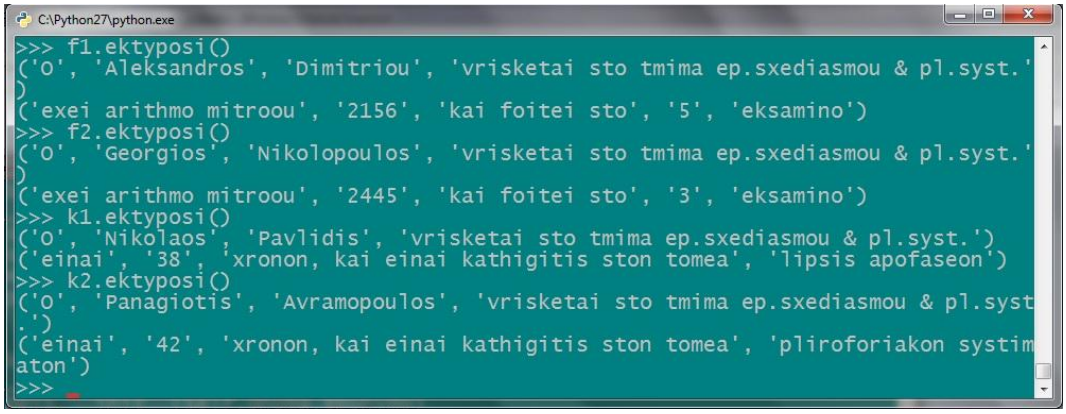


Εικόνα 85 – Παράδειγμα στην κληρονομικότητα.

Για να γίνει η εκτύπωση, θα εκτελεστεί η ίδια εντολή με το προηγούμενο παράδειγμα. Χάρη στη σύνδεση της υποκλάσης με τη βασική κλάση, θα εμφανίσει το σωστό μήνυμα εκτύπωσης τόσο για τους φοιτητές όσο και για τους διδάσκοντες.

```
>>> f1.ektyposi()
('O', 'Aleksandros', 'Dimitriou', 'vrisketai sto tmima ep.sxediasmou & pl.syst.')
('exei arithmo mitroou', '2156', 'kai foitei sto', '5', 'eksamino')
>>> f2.ektyposi()
('O', 'Georgios', 'Nikolopoulos', 'vrisketai sto tmima ep.sxediasmou & pl.syst.')
('exei arithmo mitroou', '2445', 'kai foitei sto', '3', 'eksamino')
>>> k1.ektyposi()
('O', 'Nikolaos', 'Pavlidis', 'vrisketai sto tmima ep.sxediasmou & pl.syst.')
('einai', '38', 'xronon, kai einai kathigitis ston tomea', 'lipsis apofaseon')
>>> k2.ektyposi()
```

('O', 'Panagiotis', 'Avramopoulos', 'vrisketai sto tmima ep.sxediasmou & pl.syst.')
('einai', '42', 'xronon, kai einai kathigitis ston tomea', 'pliroforiakon systimaton')



Εικόνα 86 – Συνέχεια παραδείγματος.

Διαπιστώνεται ότι, η ιδιότητα της κληρονομικότητας στη python, όντως βοηθά στη συγγραφή του προγράμματος. Με λιγότερο κόπο, συγκεντρώθηκαν στην ίδια κλάση, τα στοιχεία για τους καθηγητές και τους σπουδαστές του τμήματος, ενώ δεν χρειάστηκε να γραφεί δύο φορές ο ίδιος κώδικας. Γίνεται δηλαδή εφικτή, η ομαδοποίηση των αντικείμενων, που ίσως να έχουν ελάχιστα κοινά σημεία μεταξύ τους.

4.7 Ασκήσεις στον αντικειμενοστραφή προγραμματισμό.

A. Μια εταιρία έχει 3 καταστήματα και προσφέρει προϊόντα και υπηρεσίες, που αφορούν τη πληροφορική. Τα 3 αυτά καταστήματα βρίσκονται στην Πάτρα, στην Κόρινθο και στην Καλαμάτα αντίστοιχα. Να δημιουργηθεί ένα αντικειμενοστραφές πρόγραμμα, που θα δέχεται τον τζίρο κάθε καταστήματος και θα εμφανίζει το συνολικό και ανά τομέα τζίρο (πώληση προϊόντων και παροχή υπηρεσιών), για την προηγούμενη περίοδο. Το ποσό της παροχής υπηρεσιών είναι στο 20% του τζίρου σε κάθε πόλη. Να εκτυπωθεί ο τζίρος ανά τομέα, που έκανε κάθε κατάστημα. Θα υπολογίζει επίσης και μια εκτίμηση του τζίρου για την επόμενη περίοδο, που προβλέπεται να είναι μειωμένος κατά 40% στις πωλήσεις και 22% στην παροχή υπηρεσιών. Το μέγεθος των πωλήσεων κάθε καταστήματος, είναι μεταξύ των 400.000 και 900.000 ευρώ (το ποσό θα το δίνει ο χρήστης, μετά από σχετικό μήνυμα).

B. Να κατασκευαστεί με τη βοήθεια των κλάσεων, πρόγραμμα που αφού καταχωρηθεί ένα ΙΧ αυτοκίνητο ή μοτοσυκλέτα, θα υπολογίζει τα τέλη κυκλοφορίας του για το 2012. Θα χρησιμοποιηθούν υποκλάσεις για να υπολογιστούν τα τέλη, με βάση τα κυβικά ή τους ρύπους, ανάλογα με την άδεια πρώτης κυκλοφορίας του αυτοκινήτου. Αν η άδεια κυκλοφορίας, είναι μετά τις 1/11/2010, θα υπολογίζονται με βάση του ρύπους και τα υπόλοιπα με βάση τα κυβικά. Τα στοιχεία τα δίνει ο χρήστης σε μια λίστα, ύστερα από κάποιο μήνυμα.

- Σε αυτήν την άσκηση θα χρειαστεί και το continue, που συναντάται στο 5^{ο} κεφάλαιο (ενότητα 5.4).

Οι απαντήσεις στις ασκήσεις βρίσκονται στο παράρτημα.

ΚΕΦΑΛΑΙΟ 5ο
Έλεγχος Ροής.

Στην ενότητα αυτή, θα αναλυθεί η λειτουργία του έλεγχου ροής στην python. Ο έλεγχος ροής ανήκει στο διαδικαστικό μοντέλο προγραμματισμού. Εδώ υπάρχουν όροι όπως if, elif, while, for κλπ.

Όταν τρέχει ένα πρόγραμμα, εκτελούνται απλά οι γραμμές του κώδικα που είναι γραμμένες. Σε όλα τα προγράμματα όμως, υπάρχουν διάφορες συνθήκες που πρέπει να λαμβάνονται υπόψη και να εκτελείται διαφορετικός κώδικας ανάλογα με την περίπτωση. Άλλες φορές ο κώδικας είναι εξαιρετικά μακροσκελής και πολλές φορές επαναλαμβανόμενος.

Για να διευκολύνεται η διαδικασία της συγγραφής ενός προγράμματος χρησιμοποιούνται οι δομές ελέγχου. Υπάρχουν οι δομές επιλογής όπως το if και οι δομές επανάληψης όπως το for και το while. Επίσης, υπάρχει και η δομή try..except που χρησιμοποιείται στις εξαιρέσεις και θα αναλυθεί στο κεφάλαιο των εξαιρέσεων. Παρακάτω, θα αναλυθεί ο τρόπος που λειτουργούν όλα αυτά στην python.

5.1 Εντολή if.

Η δομή επιλογής if όπως λέει και το όνομα της, επιλέγει περιπτώσεις και για κάθε περίπτωση που εξετάζεται, εκτελούνται και διαφορετικές εντολές. Η δομή της if στην python είναι ως εξής:

```
If συνθήκη:
    Εντολές…
Elif συνθήκη2:
    Εντολές2…
Else :
    Εντολές3…
```

Το elif[20] σημαίνει 'αλλιώς αν' και τέλος το else ('αλλιώς'), που δεν εξετάζει κάποια συνθήκη, επειδή είναι η τελευταία περίπτωση και εκτελείται όταν δεν ισχύουν οι παραπάνω συνθήκες.

Στο επόμενο παράδειγμα, θα διαβαστεί ένας αριθμό από το μηδέν μέχρι το 100 και θα εμφανίσει μήνυμα για το αν είναι περιττός, άρτιος ή αν είναι μηδέν.

```
>>> x = input('Δώσε μου έναν αριθμό από το 0 μέχρι το 100  :')
Δώσε μου έναν αριθμό από το 0 μέχρι το 100  :35
```

Με τη συνάρτηση input εκχωρείται στη μεταβλητή x, η τιμή που δίνει ο χρήστης, κατά την εκτέλεση του προγράμματος. Στη συνέχεια θα εξεταστεί, αν ο αριθμός αυτός είναι το μηδέν, αν είναι άρτιος ή περιττός αριθμός.

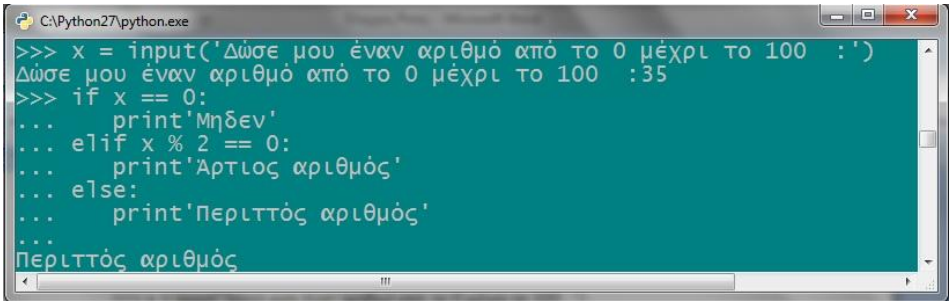
```
>>> if x == 0:
...     print 'Μηδέν'
```

Χρησιμοποιείται η εντολή if για να ελεγχτεί στην αρχή αν το x ισούται με 0. Μετά τον έλεγχο της συνθήκης (x==0), ακολουθεί άνω κάτω τελεία, που σημαίνει ότι ακολουθεί block εντολών. Στη συνέχεια με το elif, εξετάζεται η περίπτωση το υπόλοιπο της διαίρεσης με το 2, να ισούται με μηδέν. Αν δηλαδή το x είναι άρτιος. Τέλος, με το else εξετάζεται η περίπτωση να μην είναι τίποτα από τα παραπάνω, να είναι δηλαδή περιττός αριθμός. Σε

[20] Σε άλλες γλώσσες προγραμματισμού υπάρχει το else if, στην python όμως μετατράπηκε για συντομία σε elif.

όλους τους ελέγχους (if, elif και else), μπαίνει άνω κάτω τελεία, επειδή ανοίγει νέο block και εκεί προστίθενται οι αντίστοιχες εντολές. Αφού ελέγξει τον αριθμό 35, επιστρέφει μήνυμα ότι είναι περιττός αριθμός.

```
... elif x % 2 == 0:
...    print ‘Άρτιος αριθμός’
... else:
...    print ‘Περιττός αριθμός’
...
Περιττός αριθμός
```



Εικόνα 87 – Παράδειγμα με τη δομή if.

5.2 Δομές επανάληψης.

Οι δομές επανάληψης επιτρέπουν να εκτελεστεί μια διαδικασία πολλές φορές μέχρι να παύσει να ισχύει μια συνθήκη. Αυτό επιτρέπει στον προγραμματιστή να μειώσει την έκταση του κώδικα. Ανάλογα με την επανάληψη που επιθυμεί, υπάρχουν οι εντολές for και while.

5.2.1 Εντολή for..in

Η εντολή for χρησιμοποιείται για την εκτέλεση επαναλαμβανόμενων εντολών. Η σύνταξη της for διαφέρει λίγο στην python από τις άλλες γλώσσες προγραμματισμού όπως η C και η Pascal. Στην python επαναλαμβάνεται η διαδικασία, για τα στοιχεία μιας ακολουθίας με τη σειρά που εμφανίζονται. Δεν ορίζεται το βήμα και το τέλος της επανάληψης. Για επιτευχθεί αυτό, πρέπει να οριστεί ένα slice της λίστας, ή να διατρεχθεί ένα range() με βήμα, αλλά αυτά αφορούν καθαρά το slice και τη συνάρτηση range αντίστοιχα και δεν έχουν σχέση με τη for.

Άρα η επανάληψη for θα εκτελέσει ένα σύνολο εντολών στα δεδομένα μιας λίστας που έχει οριστεί. Η σύνταξη της for..in παρατίθεται παρακάτω:

```
For i in λίστα:
   Εντολές…
```

Με το i εννοείται το αντίστοιχο στοιχείο της λίστας, που θα αυξάνεται κατά 1 όποτε εκτελούνται οι εντολές. Υπάρχει επίσης και η λίστα, για τα στοιχεία της οποίας γίνεται η επανάληψη. Έτσι, θα εκτελούνται κάθε φορά οι εντολές για το i στοιχείο της λίστας. Αυτό το είδος της επανάληψης, είναι γνωστό στις άλλες γλώσσες προγραμματισμού, ως for.

Στο παρακάτω παράδειγμα, κατασκευάζεται μια λίστα a στην οποία εισάγονται μερικές πόλεις και εκτυπώνεται μήνυμα για κάθε πόλη.

```
>>> a = ['Athina','Patra','Kalamata','Tripoli','Veroia','Arta']
```

Θα δημιουργηθεί η for χρησιμοποιώντας αυτή τη φορά το δείκτη poli αντί του i και στη συνέχεια το όνομα της λίστας a. Στην εμφάνιση του μηνύματος, με το a.index(poli), θα εννοείται ο δείκτης (index) του στοιχείου της λίστας. Επειδή όμως η λίστα, έχει τιμή δείκτη που ξεκινάει από το μηδέν, θα προστεθεί κατά ένα για να εμφανίζεται το μήνυμα σωστά.

```
>>> for poli in a:
...     print('I', a.index(poli)+1,'poli tis listas einai i:', poli)
...
('I', 1, 'poli tis listas einai i:', 'Athina')
('I', 2, 'poli tis listas einai i:', 'Patra')
('I', 3, 'poli tis listas einai i:', 'Kalamata')
('I', 4, 'poli tis listas einai i:', 'Tripoli')
('I', 5, 'poli tis listas einai i:', 'Veroia')
('I', 6, 'poli tis listas einai i:', 'Arta')
```

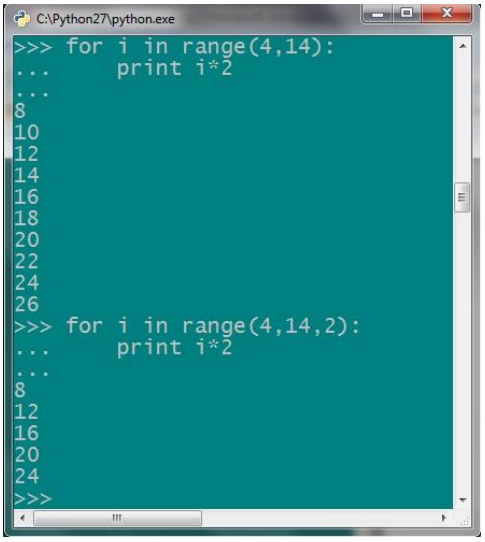
Εικόνα 88 – Παράδειγμα με τη δομή for..in.

Σε άλλο παράδειγμα τώρα, θα εφαρμοστεί η for σε μια λίστα αριθμών που θα προκύψει χρησιμοποιώντας τη συνάρτηση range(). Σε μια λίστα αριθμών από το 4 μέχρι το 14, θα διπλασιαστούν οι τιμές της.

```
>>> for i in range(4,14):
...     print i*2
...
8
10
12
14
16
18
20
22
24
26
```

Σε αυτή την επανάληψη, διπλασιάστηκαν τα στοιχεία της λίστας. Παρακάτω, φαίνεται πως μπορεί να εφαρμοστεί το βήμα, αλλά πρέπει να επισημανθεί, ότι είναι χαρακτηριστικό της range() συνάρτησης και όχι του βρόγχου for. Θα κάνει επανάληψη με βήμα 2. Ουσιαστικά, δεν ορίζεται επανάληψη το βήμα δύο, αλλά παίρνει μια λίστα με τα μισά στοιχεία σε σχέση με πριν.

```
>>> for i in range(4,14,2):
...     print i*2
...
8
12
16
20
24
```



Εικόνα 89 – Επανάληψη με βήμα.

Η for εφαρμόζεται πιο εύκολα στην python, σε σχέση με άλλες γλώσσες προγραμματισμού, χάρη στην καλή συνεργασία με συναρτήσεις όπως η range.

5.2.2 *Εντολή While.*

Στην εντολή while εκτελείται ο κώδικας με τις εντολές που έχουν δηλωθεί, μέχρι να σταματήσει να ισχύει μια συνθήκη. Η while έχει αυτή τη μορφή:

While συνθήκη:

Εντολές…

Αποτελέσματα

Όσο δηλαδή ισχύει η συνθήκη θα εκτελούνται οι εντολές που βρίσκονται στο block. Όταν σταματήσει να είναι αληθής η συνθήκη, που ορίστηκε στην αρχή, θα επιστρέψει τα αποτελέσματα.

Για παράδειγμα, σε μια λίστα με θετικούς αριθμούς, θα εμφανίζει μήνυμα αν ο αριθμός που εξετάζεται είναι άρτιος ή περιττός.

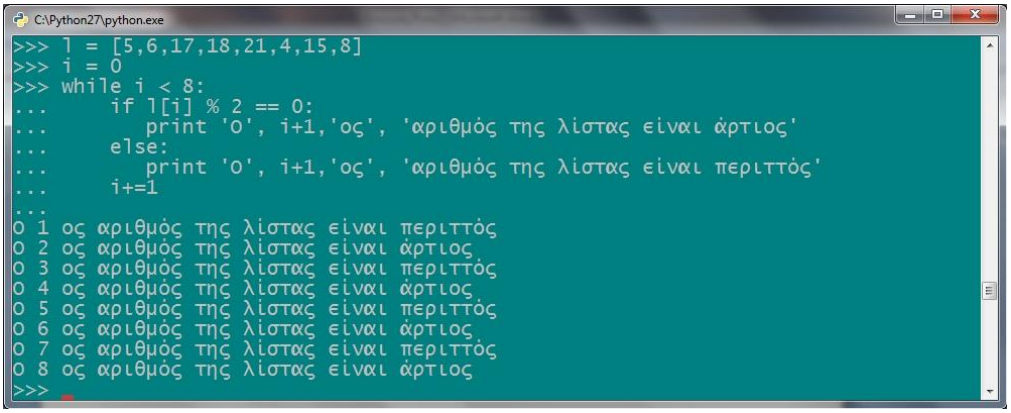
```
>>> l = [5,6,17,18,21,4,15,8]
>>> i = 0
```

Στο βρόγχο με το while, θα υπάρχει όρος για να εκτελεστεί η επανάληψη, για όσο διάστημα η τιμή της μεταβλητής i, είναι μικρότερη από το 8. Το i, έχει αρχικά τη τιμή 0, που αντιστοιχεί στο δείκτη του πρώτου στοιχείου της λίστας. Όταν πάρει την τιμή 7, θα εξεταστεί το τελευταίο στοιχείο. Έτσι, θα σταματήσει η επανάληψη μετά το 7. Με το i+=1 αυξάνεται η τιμή του i κατά 1. Με τις ίδιες εντολές του αντίστοιχου παραδείγματος παραπάνω, θα εξεταστεί αν ο αριθμός είναι άρτιος ή περιττός.

```
>>> while i < 8:
...     if l[i] % 2 == 0:
...       print 'O', i+1,'ος', 'αριθμός της λίστας είναι άρτιος'
...     else:
...       print 'O', i+1,'ος', 'αριθμός της λίστας είναι περιττός'
...     i+=1
...
O 1 ος αριθμός της λίστας είναι περιττός
O 2 ος αριθμός της λίστας είναι άρτιος
O 3 ος αριθμός της λίστας είναι περιττός
O 4 ος αριθμός της λίστας είναι άρτιος
O 5 ος αριθμός της λίστας είναι περιττός
```

Ο 6 ος αριθμός της λίστας είναι άρτιος
Ο 7 ος αριθμός της λίστας είναι περιττός
Ο 8 ος αριθμός της λίστας είναι άρτιος

Στην print δεν υπήρχε παρένθεση και εμφάνισε το μήνυμα ενιαίο χωρίς κόμματα και αποστρόφους. Αν γραφτεί όμως το μήνυμα σε παρένθεση πρέπει να χρησιμοποιηθούν οπωσδήποτε λατινικοί χαρακτήρες για να εμφανιστεί, αλλιώς δεν θα αναγνωριστεί η κωδικοποίηση.



Εικόνα 90 – Παράδειγμα με τη while.

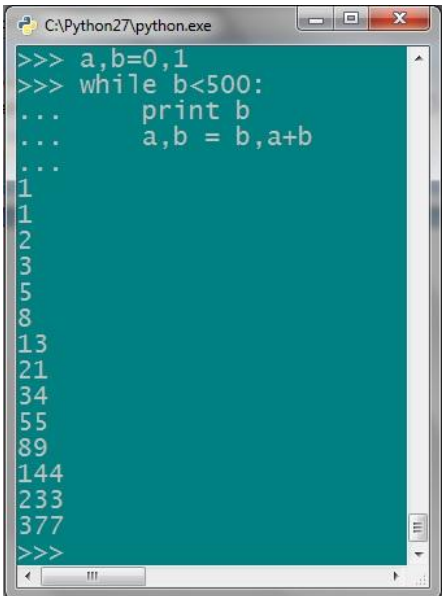
Υπάρχει περίπτωση στη while, να μην συνταχθεί σωστά η συνθήκη ή να μην μεταβληθεί η τιμή της μεταβλητής, που εξετάζεται με τη συνθήκη. Ένα τέτοιο λάθος θα οδηγήσει σε ατέρμονη επανάληψη και μια τέτοια περίπτωση θα αναγκάσει την python να εκτελεί συνέχεια εντολές. Για να σταματήσει η ατέρμονη διαδικασία, θα πρέπει να γίνει η διακοπή της από το χρήστη (Keyboard Interrupt), πατώντας τα πλήκτρα Ctrl + C. Όταν εκτελεστεί αυτό, θα επιστρέψει την παρακάτω εξαίρεση.

```
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
```

Στην αναφορά για τις συναρτήσεις, χρησιμοποιήθηκε η επανάληψη while, στο παράδειγμα με τη συνάρτηση Fibonacci. Ακολουθεί ο κώδικας, που χρησιμοποιήθηκε για τη δημιουργία μιας σειράς Fibonacci μέχρι το 500, χωρίς όμως τώρα να οριστεί κάποια συνάρτηση.

```
>>> a,b=0,1
>>> while b < 500:
        ...     print b
        ...     a,b = b,a+b
        ...
1
1
2
3
5
8
13
21
34
55
89
```

144
233
377



Εικόνα 91 – Παράδειγμα Fibonacci.

5.3 Εντολή break.

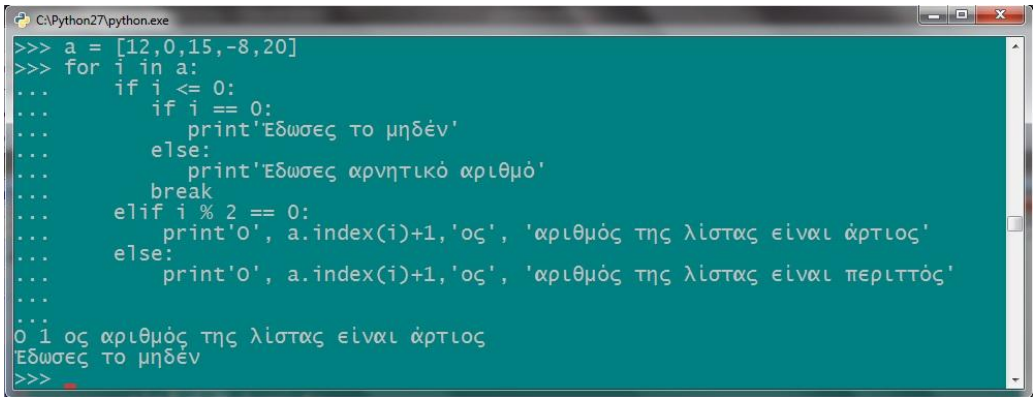
Η εντολή break μπορεί να εκτελεστεί σε ένα βρόγχο επανάληψης είτε είναι for, είτε while. Αυτή η εντολή διακόπτει την εκτέλεση του βρόγχου και συνεχίζει με τις εντολές μετά το βρόγχο εφόσον υπάρχουν. [21]

Ξαναχρησιμοποιώντας το παράδειγμα της λίστας με τους θετικούς αριθμούς και τον έλεγχο των άρτιων και των περιττών, θα εξετάζεται αν η τιμή είναι αρνητική ή μηδέν. Σε τέτοια περίπτωση, θα σταματάει η διαδικασία και θα εκτυπώνεται σχετικό μήνυμα για το λόγο που σταμάτησε.

```
>>> a = [12,0,15,-8,20]
>>> for i in a:
...     if i <= 0:
...       if i == 0:
...         print'Έδωσες το μηδέν'
...       else:
...         print'Έδωσες αρνητικό αριθμό'
...       break
...     elif i % 2 == 0:
...       print'O', a.index(i)+1,'ος', 'αριθμός της λίστας είναι άρτιος'
...     else:
...       print'O', a.index(i)+1,'ος', 'αριθμός της λίστας είναι περιττός'
...
...
O 1 ος αριθμός της λίστας είναι άρτιος
Έδωσες το μηδέν
```

[21] Swaroop CH, «*A Byte of Python*» (Ελληνική μετάφραση), Έλεγχος ροής, Ανάκτηση στις 15-9-2011 από http://www.swaroopch.org/notes/Python_el:%CE%A0%CE%B5%CF%81%CE%B9%CE%B5%CF%87%CF%8C%CE%BC%CE%B5%CE%BD%CE%B1

Παρατηρείται ότι η εντολή break, σταματάει την επανάληψη. Είναι κάτι σαν έξοδος από τον επαναληπτικό βρόγχο και δεν εκτελούνται οι επόμενες εντολές που βρίσκονται στο block.



Εικόνα 92 – Παράδειγμα με την εντολή break.

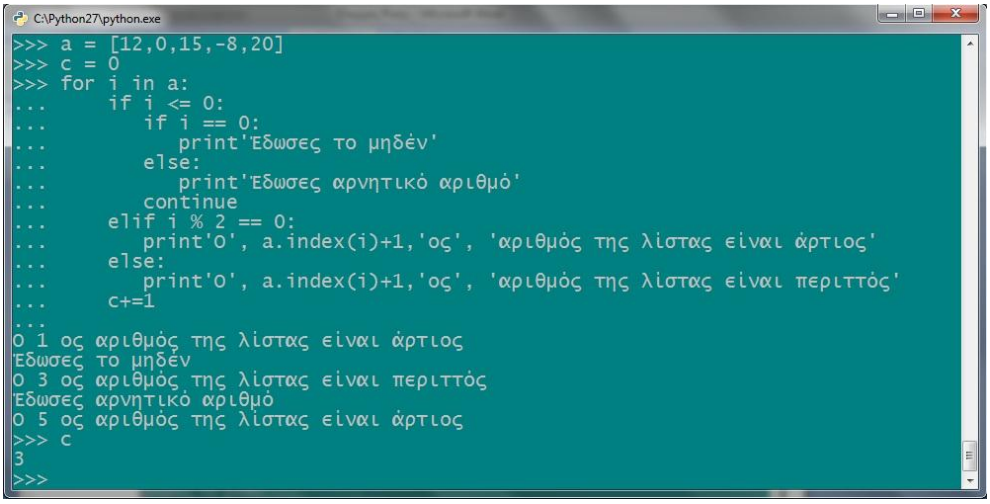
5.4 Εντολή continue.

Το continue χρησιμοποιείται, για να παραλειφθούν οι εντολές του block στην επανάληψη και να συνεχιστεί με την εκτέλεση των εντολών της επόμενης τιμής της ακολουθίας.

Θα εκτελεστεί το ίδιο παράδειγμα, αλλά με την εφαρμογή της continue στη θέση της break. Θα μπει και ένας μετρητής c, για να γίνει ξεκάθαρος ο τρόπος λειτουργίας της εντολής continue, που θα μετράει πόσοι θετικοί αριθμοί υπάρχουν στη λίστα.

```
>>> a = [12,0,15,-8,20]
>>> c = 0
>>> for i in a:
...     if i <= 0:
...       if i == 0:
...         printΈδωσες το μηδέν'
...       else:
...         printΈδωσες αρνητικό αριθμό'
...       continue
...     elif i % 2 == 0:
...       print'O', a.index(i)+1,'ος', 'αριθμός της λίστας είναι άρτιος'
...     else:
...       print'O', a.index(i)+1,'ος', 'αριθμός της λίστας είναι περιττός'
...     c+=1
...
O 1 ος αριθμός της λίστας είναι άρτιος
Έδωσες το μηδέν
O 3 ος αριθμός της λίστας είναι περιττός
Έδωσες αρνητικό αριθμό
O 5 ος αριθμός της λίστας είναι άρτιος
>>> c
3
```

Από τη τιμή του μετρητή, παρατηρείται, πως όντως δεν εκτελέστηκαν οι υπόλοιπες εντολές του block, μετά από την εκτέλεση του continue.



Εικόνα 93 – Παράδειγμα με την continue.

Η break σταματάει εντελώς την επανάληψη, ενώ το continue διακόπτει μόνο τις εντολές στο τρέχον block και συνεχίζει με την επόμενη επανάληψη.

5.5 Ασκήσεις στον έλεγχο ροής.

A. Να δοθεί ένας ακέραιος αριθμός και να επιστρέφει μήνυμα αν είναι θετικός ή αρνητικός. Θα εμφανίζεται μήνυμα στο χρήστη, που θα του ζητάει να δώσει έναν ακέραιο.

B. Για ένα μάθημα που εξετάστηκε ένας φοιτητής, θα δίνεται ο βαθμός του στο μάθημα αυτό και θα τον καλείται να δώσει τον βαθμό που πιστεύει ότι έχει πάρει. Ανάλογα με το τι θα γράψει θα του εμφανίζει ανάλογο μήνυμα.

C. Να κατασκευαστεί μια λίστα και να εκτυπώνεται μήνυμα για κάθε ένα στοιχειό της λίστας.

D. Σε μια λίστα αριθμών από το 8 μέχρι το 18 να διπλασιαστούν οι τιμές της.

E. Στην ίδια λίστα αριθμών να γίνει επανάληψη με βήμα 4.

F. Να δημιουργηθεί πρόγραμμα στο οποίο, θα επιλέγεται μέσα από συνάρτηση τυχαίας επιλογής, ένας αριθμός από το 1 μέχρι το 100 και στη συνέχεια θα ζητείται από το χρήστη του προγράμματος να μαντέψει αυτόν τον αριθμό. Θα εμφανίζει κατά την έξοδο του και πόσες επιτυχίες είχε τελικά.
    - Στοιχεία για την άσκηση: Θα εισαχθεί το module random από τη βιβλιοθήκη της python και θα χρησιμοποιηθεί η μέθοδος random.randint(1,100), για να επιλεχθεί ένας τυχαίος ακέραιος αριθμός, τον οποίο θα πρέπει να μαντέψει ο χρήστης.

Οι λύσεις των ασκήσεων βρίσκονται στο παράρτημα.

ΚΕΦΑΛΑΙΟ 6ο
Είσοδος, έξοδος και αρχεία.

Σε αυτό το κεφάλαιο θα αναλυθεί πως γίνεται η είσοδος και η έξοδος των δεδομένων στην python, αλλά και πως δημιουργούνται νέα αρχεία και τον τρόπο που αυτά επεξεργάζονται.

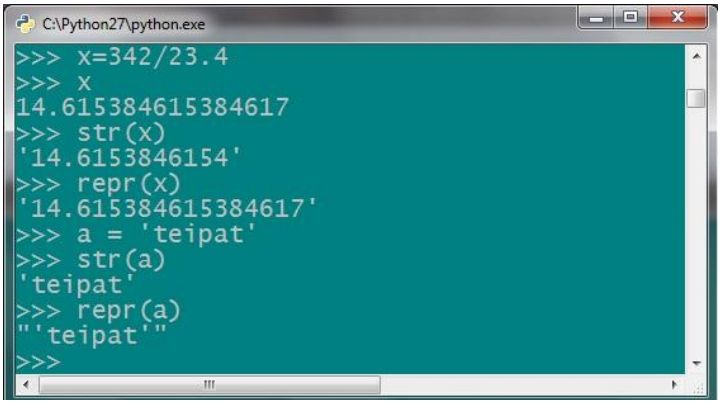
6.1 Είσοδος και έξοδος δεδομένων.

Με τον όρο είσοδο εννοείται ο τρόπος εισαγωγής των δεδομένων στην python και πως αυτά χρησιμοποιούνται αργότερα. Τα αποτελέσματα μπορούν, είτε να εμφανιστούν στην οθόνη για να τα διαβάσει ο χρήστης, είτε να αποθηκευτούν σε ένα αρχείο για μελλοντική χρήση.

Για την εισαγωγή δεδομένων σε ένα πρόγραμμα, υπάρχει όπως είναι γνωστό, η συνάρτηση input(). Εδώ, γράφεται το κατάλληλο μήνυμα, που θα εμφανιστεί στην οθόνη του υπολογιστή και θα δώσει ο χρήστης τα δεδομένα.

Η έξοδος μπορεί να επιτευχθεί με τη γνωστή εντολή print(), αλλά και με διάφορες εκφράσεις. Όταν υπάρχουν αρχεία, χρησιμοποιείται η εντολή write(). Άλλες μέθοδοι εξόδου είναι η συνάρτηση string str() και repr(), που εμφανίζουν την τιμή ως αλφαριθμητικό. Η str() επιστρέφει τιμές που είναι αναγνώσιμες από το χρήστη, ενώ το repr() θα εμφανίζει χαρακτήρες που μπορούν να διαβαστούν από τον διερμηνευτή. Αρκετές φορές και οι δυο μέθοδοι έχουν το ίδιο αποτέλεσμα.

```
>>> x=342/23.4
>>> x
14.615384615384617
>>> str(x)
'14.6153846154'
>>> repr(x)
'14.615384615384617'
>>> a = 'teipat'
>>> str(a)
'teipat'
>>> repr(a)
"'teipat'"
```



Εικόνα 94 - Μέθοδοι για εμφάνιση δεδομένων.

Υπάρχουν και άλλες μέθοδοι με το str όπως το str.rjust() και str.ljust(), που κάνει στοίχιση του κειμένου δεξιά και αριστερά αντίστοιχα. Για περισσότερες λεπτομέρειες, μπορεί να κληθεί η βοήθεια με την εντολή help(str).

6.2 Αρχεία.

Τα αρχεία, είναι δεδομένα αποθηκευμένα σε συγκεκριμένη μορφή στο σκληρό δίσκο του υπολογιστή. Ένα αρχείο, μπορεί να είναι ένα έγγραφο κειμένου (txt), που να περιέχει python εντολές. Πρέπει να βρίσκεται στην εγκατάσταση του συστήματος και συγκεκριμένα στο φάκελο της python (εικόνα 96).

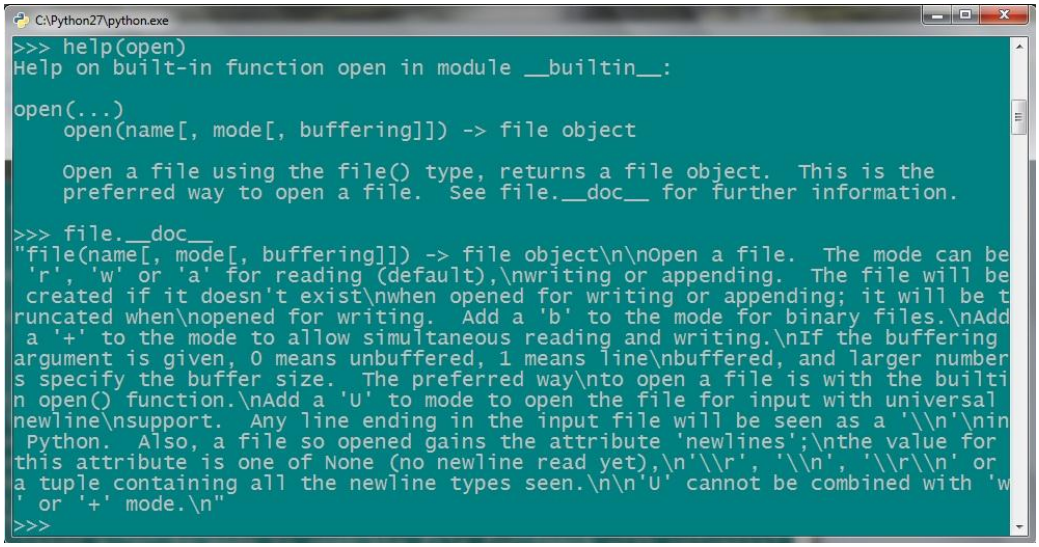
Οι λειτουργίες που ενδέχεται να γίνουν σε ένα αρχείο, είναι η επεξεργασία των δεδομένων που περιέχει, ή απλά η ανάγνωσή τους. Για να ανοίξει ένα αρχείο, εκτελείται η εντολή open(filename, mode) – όνομα αρχείου, λειτουργία. Μέσα στην παρένθεση βρίσκονται σε εισαγωγικά το όνομα του αρχείου και ο επιθυμητός τρόπος χρήσης. Η δήλωση του τρόπου χρήσης είναι υποχρεωτική. Υπάρχουν αρκετοί τρόποι χρήσης ενός αρχείου, οι βασικότεροι από αυτούς παρατίθενται στον πίνακα που ακολουθεί:

Πίνακας 4: Λειτουργίες στα αρχεία.	
Τρόπος Χρήσης	Περιγραφή
‘r’	Άνοιγμα αρχείου μόνο για ανάγνωση
‘w’	Εγγραφή αρχείου
‘a’	Προσθήκη στοιχείων στο αρχείο
‘b’	Αρχείο σε δυαδική μορφή

- Με το ‘w’ γράφονται νέα στοιχεία στο αρχείο, αλλά αυτό θα διαγράψει όλα τα προηγούμενα δεδομένα του αρχείου, για να γραφούν τα καινούργια.
- Το ‘a’ προέρχεται από το append και δίνει τη δυνατότητα να προστεθούν νέα δεδομένα στο τέλος του αρχείου, χωρίς όμως να χάνονται τα παλιά στοιχεία.
- Το ‘b’ επιστρέφει δυαδικής μορφής αρχεία και όχι αρχεία κειμένου.

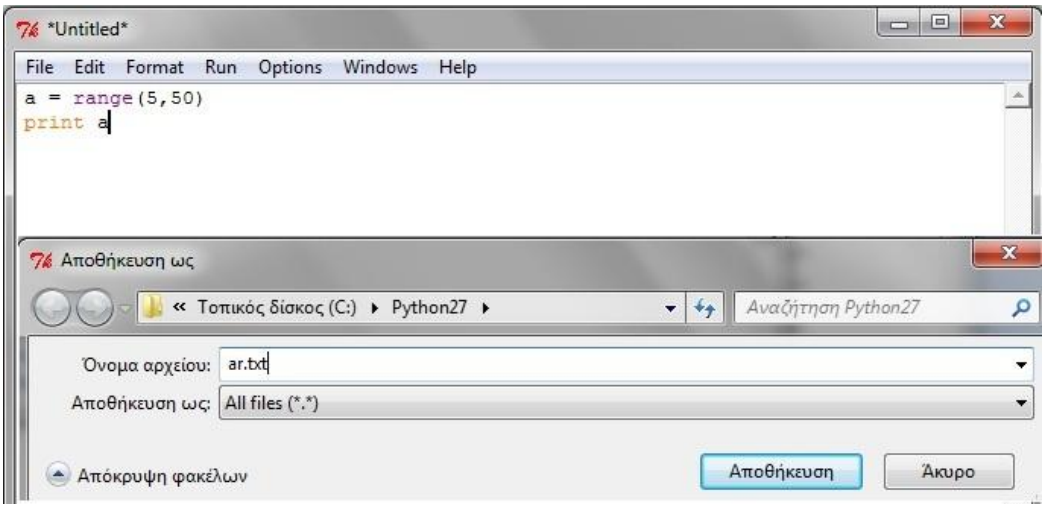
Αν δεν δηλωθεί ο τρόπος λειτουργίας στο open(), θα εμφανίσει το αρχείο κειμένου μόνο για ανάγνωση (χρήση r). Υπάρχει τέλος η δυνατότητα να συνδυαστούν περισσότερες χρήσεις σε ένα αρχείο, όπως τα ‘rb’(δυαδικής μορφής αρχείο μόνο για ανάγνωση), ‘wb’(δυαδικό αρχείο μόνο για εγγραφή), αλλά και με το +. Έτσι, δημιουργούνται εντολές όπως το ‘r+’, που ανοίγει το αρχείο για ανάγνωση και γράψιμο και τέλος το ‘r+b’, που αφορά ανάγνωση και εγγραφή σε δυαδικό αρχείο.

Όλα αυτά φαίνονται αναλυτικά στην python με τη βοήθεια της εντολής help(open), όπως φαίνεται παρακάτω:



Εικόνα 95 – Κλήση της βοήθειας στα αρχεία.

Για να γίνει απόλυτα κατανοητός ο τρόπος λειτουργίας των αρχείων στην python, θα δημιουργηθεί ένα απλό αρχείο κειμένου, που θα χρησιμοποιηθεί στη συνέχεια. Με τη βοήθεια του κειμενογράφου της python, δημιουργείται και αποθηκεύεται ένα έγγραφο κειμένου (.txt), με μερικές python εντολές. Το αρχείο θα αποθηκευτεί στο φάκελο της python έκδοσης.



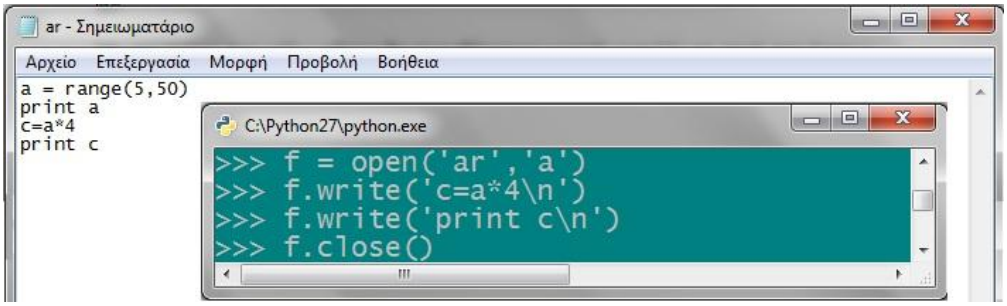
Εικόνα 96 – Δημιουργία αρχείου.

6.2.1 *Προσθήκη στοιχείων και εγγραφή (append και write).*

Αφού ανοιχτεί το παραπάνω αρχείο με το open() και την λειτουργία της προσθήκης (‘a’), θα χρησιμοποιηθεί η εντολή write(), για να προστεθούν μερικές εντολές σε αυτά που υπάρχουν. Δημιουργείται μια νέα μεταβλητή f, στην οποία ανατίθεται το αρχείο.

```
>>> f = open('ar','a')
>>> f.write('c=a*4\n')
>>> f.write('print c\n')
>>> f.close()
```

Για την εισαγωγή των νέων στοιχείων, θα εφαρμοστεί η μέθοδος f.write(‘νέα γραμμή στοιχείων\n) Το f είναι η μεταβλητή που θα εφαρμοστεί η write και το \n, δηλώνει το τέλος της γραμμής (αλλάζει γραμμή). Με το close() κλείνει η διαδικασία, που ξεκίνησε το open(). Έτσι, αν ανοίξει εκ νέου το αρχείο, θα είναι ως φαίνεται στην παρακάτω εικόνα (εικόνα 97).



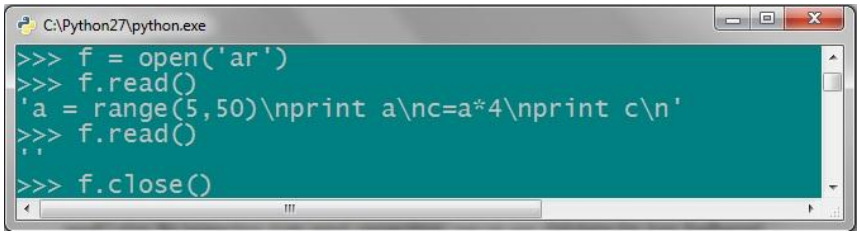
Εικόνα 97 – Προσθήκη στοιχείων στο αρχείο.

Παρατηρείται ότι οι γραμμές που προστέθηκαν με τις εντολές, αποθηκεύτηκαν στο αρχείο.

Αν είχε κληθεί η λειτουργία 'w' (write) αντί του 'a', θα γινόταν η ίδια διαδικασία, αλλά όμως θα χάνονταν οι εντολές a = range(5,50) και print a. Για να μην χαθούν, έπρεπε εισαχθούν πάλι στο αρχείο, δηλαδή κάπως έτσι: f.write('a = range(5,50)\n') κλπ.

6.2.2 *Ανάγνωση αρχείου read().*

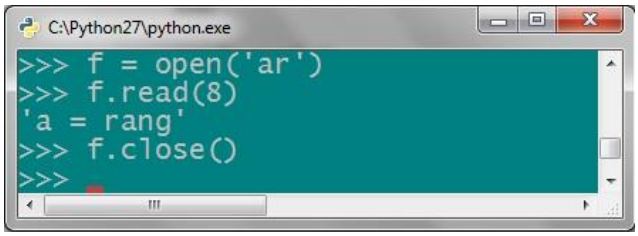
Υπάρχουν και άλλες μέθοδοι που χρησιμοποιούνται στα αρχεία, ανάλογα με τον τρόπο χρήσης. Αν ανοίξει το αρχείο ξανά, αλλά μόνο για ανάγνωση αυτή τη φορά, μπορούν τότε να διαβαστούν τα δεδομένα με τις μεθόδους read(), readline() και readlines(). Η μέθοδος read(), διαβάζει όλο το αρχείο και το εμφανίζει. Αν ξαναεκτελεστεί η read(), τότε θα εμφανίσει έναν κενό χαρακτήρα, για να δηλώσει ότι έχει διαβαστεί όλο το αρχείο.



Εικόνα 98 – Ανάγνωση αρχείου με το read.

Η read() μπορεί να δεχτεί και τιμές, που αναφέρονται σε bytes, όπως read(2) κλπ. Αν μπει τιμή στην παρένθεση, θα διαβαστούν όσα bytes δηλώνονται. Στο παράδειγμα που ακολουθεί, Θα διαβαστούν 8 bytes από την τρέχουσα θέση και όχι όλο το αρχείο.[22]

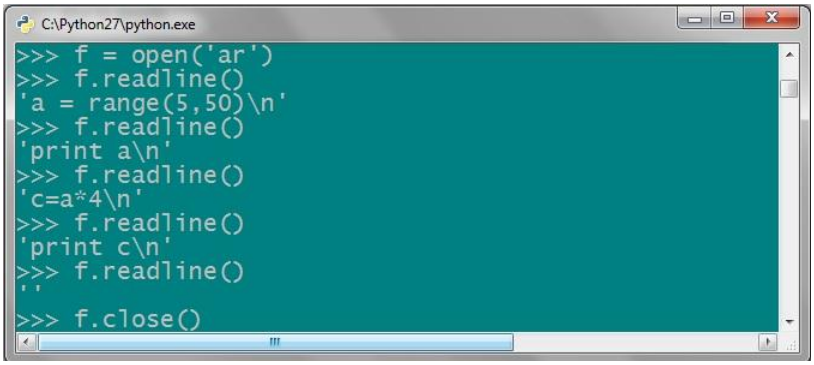
```
>>> f = open('ar')
>>> f.read(8)
'a = rang'
```



Εικόνα 99 – Ανάγνωση συγκεκριμένων bytes.

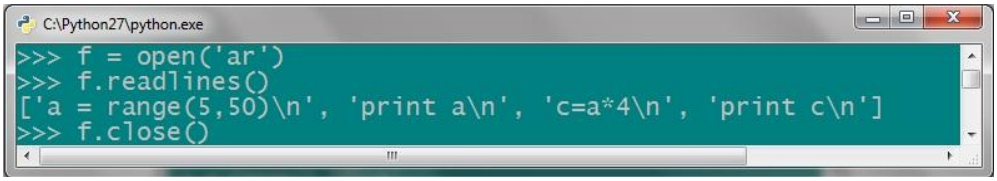
[22] Κάθε χαρακτήρας είτε είναι κενό, είτε όχι, θα αντιστοιχεί σε ένα byte.

Όσο για τη μέθοδο readline(), θα διαβάζει τον κώδικα ανά γραμμή. Όταν τελειώσει ο κώδικας, σε κάθε εκτέλεση της readline(), θα εμφανίζει έναν κενό χαρακτήρα.



Εικόνα 100 – Μέθοδος readline.

Τέλος, η readlines() θα εμφανίσει μια λίστα, που κάθε στοιχείο της θα αποτελείται από μια γραμμή του αρχείου.



Εικόνα 101 – Ανάγνωση αρχείου με readlines.

Εκτός από τις παραπάνω μεθόδους, υπάρχουν και άλλες πιο εξειδικευμένες όπως η tell() και η seek().

Η tell() θα επιστρέψει στην οθόνη, έναν ακέραιο αριθμό μαζί με την τρέχουσα θέση σε bytes, με αρίθμηση, που ξεκινά από τον πρώτο χαρακτήρα του αρχείου. Όταν βρίσκεται στην αρχή, εμφανίζει τιμή byte 0, αλλά αυτή η τιμή δεν αφορά το πρώτο στοιχείο του αρχείου.

Η seek() εφαρμόζεται για να μεταφερθεί ο προγραμματιστής σε ένα συγκεκριμένο byte του αρχείου, που θα δηλώνει στην παρένθεση. Η seek μπορεί να έχει και δύο παραμέτρους στην παρένθεση και να είναι της μορφής f.seek (offset, from_what), όπου το from_what παίρνει τιμές 0,1 ή 2. Το 0 είναι για την αρχή, το 1 αναφέρεται στο τρέχων byte του αρχείου και τέλος η τιμή 2, που είναι το τέλος του αρχείου. Το offset μπορεί να πάρει θετικές και αρνητικές τιμές. Εδώ εννοούνται τα bytes, που πρέπει να μετακινηθεί πάνω ή κάτω, σε σχέση με το byte της παραμέτρου from_what. Αν υπάρχει μόνο μια παράμετρος στην παρένθεση, όπως 2 ή 15, θα μεταφερθεί στο 2ο και στο 15ο byte αντίστοιχα.

Ακολουθεί μια εφαρμογή των παραπάνω μεθόδων, για να σχηματιστεί μια πιο ολοκληρωμένη εικόνα του τρόπου λειτουργίας τους. Με την εντολή που θα εκτελεστεί, θα μεταφερθεί στο 6ο byte του αρχείου, όπως διαπιστώνεται με την εκτέλεση της μεθόδου tell().

```
>>> f = open('ar')
>>> f.seek(6)
>>> f.tell()
6L
>>> f.read(1)
'n'
>>> f.tell()
7L
>>> f.read(5)
```

```
'ge(5,'
>>> f.tell()
12L
>>> f.seek(10,1)
```

Η εντολή seek με τιμές παραμέτρων 10 και 1 αντίστοιχα, πηγαίνει 10 bytes μπροστά από την τρέχουσα θέση. Θα μεταφερθεί δηλαδή στη θέση με αριθμό bytes 22 από τα 12 bytes. Με την παρακάτω εντολή, γίνεται μεταφορά 8 bytes πριν από το τέλος του αρχείου.

```
>>> f.tell()
22L
>>> f.read(1)
' '
>>> f.tell()
23L
>>> f.seek(-8,2)
>>> f.read()
'rint c\n'
>>> f.seek(-1,2)
>>> f.read()
'\n'
>>> f.close()
```

Εικόνα 102 – Τρόπος λειτουργίας των μεθόδων tell() και seek().

6.3 Δυαδικά αρχεία (b).

Η python δίνει τη δυνατότητα της επεξεργασίας δυαδικών αρχείων, εκτός από αρχεία κειμένου. Όπως αναφέρθηκε, οι δυνατότητες χρήσης είναι οι ίδιες και στα δύο είδη αρχείων, αλλά υπάρχουν μερικά σημεία που πρέπει να επισημανθούν, ειδικά αν χρησιμοποιούνται τα windows.

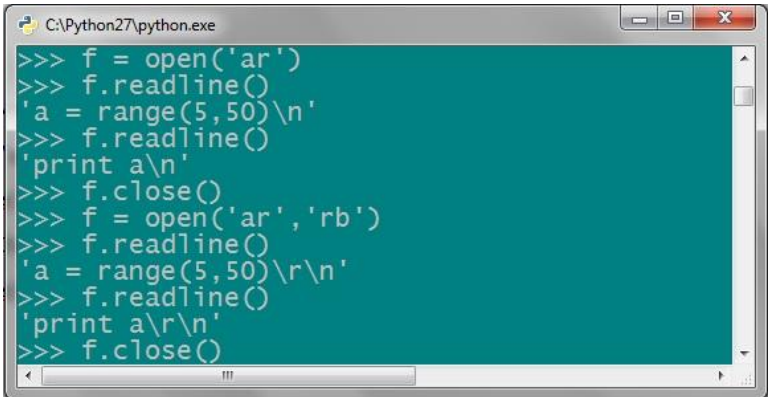
Η python στα windows διαχωρίζει, τα δυαδικά αρχεία από τα αρχεία κειμένου. Έτσι οι χαρακτήρες στο τέλος κάθε γραμμής, διαφοροποιούνται αυτόματα κάθε φορά, που γίνεται

ανάγνωση ή εγγραφή σε ένα αρχείο. Δεν θα προκύψει πρόβλημα, αν το κείμενο έχει κωδικοποίηση ASCII. Σε περιπτώσεις όμως, που υπάρχουν φωτογραφίες ή εκτελέσιμα προγράμματα στο αρχείο (αφορά αρχεία jpeg και exe), θα κάνει τη μετατροπή αυτή με αποτέλεσμα να χαλάσει τα δεδομένα. Σε τέτοια 'ευαίσθητα' δεδομένα δεν γίνεται επεξεργασία με την μορφή αρχείων κειμένων (text), αλλά με τη δυαδική μορφή 'b'.

Σε άλλα λειτουργικά συστήματα όπως τα Unix, δεν ελλοχεύουν τέτοιοι κίνδυνοι, μιας και δεν γίνεται η αυτόματη αντιστοιχία, όπως στα windows και δεν είναι αναγκαίο να εφαρμοστεί το δυαδικό 'b'.

Για να γίνει αντιληπτή η διαφορά, θα ανοιχτεί το αρχείο του παραδείγματος για ανάγνωση και με τις δύο μορφές. Αφού ανοίξει σε δυαδική μορφή, παρατηρείται ότι το τέλος της γραμμής μεταφράζεται στη δυαδική μορφή σε \r\n από \n.

```
>>> f = open('ar')
>>> f.readline()
'a = range(5,50)\n'
>>> f.readline()
'print a\n'
>>> f.close()
>>> f = open('ar','rb')
>>> f.readline()
'a = range(5,50)\r\n'
>>> f.readline()
'print a\r\n'
>>> f.close()
```



Εικόνα 103 – Άνοιγμα αρχείου σε δυαδική μορφή.

6.4 Pickle module.

Η Python παρέχει στον προγραμματιστή ένα χρήσιμο module, που ονομάζεται pickle και επιτρέπει την αποθήκευση ενός αντικείμενου σε ένα αρχείο, χωρίς να χάνονται τα δεδομένα κατά την ανάκτηση του. Το module pickle μπορεί να μετατρέψει οποιοδήποτε αντικείμενο σε μια απλή σειρά από strings (data serialization), για να αποθηκευθεί στη συνέχεια ή να αποσταλεί μέσω δικτύου. Η μετατροπή των δεδομένων, γίνεται με τέτοιο τρόπο, ώστε να μην αλλοιώνονται κατά την ανάκτηση του αντικειμένου. Χρησιμοποιείται σε περιπτώσεις πολύπλοκων δεδομένων, όπως είναι οι λίστες, τα λεξικά και τα αντικείμενα των κλάσεων.

Η διαδικασία της μετατροπής των δεδομένων ονομάζεται pickling και η ανάκτηση τους unpickling. Το pickle χρησιμοποιεί διάφορα πρωτόκολλα για να κάνει τη μετατροπή, τα οποία είναι τα εξής[23]:

- Το πρωτόκολλο έκδοσης 0, είναι το αρχικό ASCII πρωτόκολλο που δημιουργήθηκε και το αποτέλεσμα της μετατροπής είναι σε αναγνώσιμη μορφή από το χρήστη.
- Το πρωτόκολλο έκδοσης 1, είναι πρωτόκολλο που κάνει μετατροπή σε δυαδική μορφή και δεν χρησιμοποιείται πια.
- Το πρωτόκολλο έκδοσης 2, είναι το δυαδικό πρωτόκολλο που χρησιμοποιείται για την έκδοση 2 της python.
- Το πρωτόκολλο έκδοσης 3, είναι αυτό που χρησιμοποιείται στην 3 έκδοση της python.

Από προεπιλογή το pickle, χρησιμοποιεί το πρωτόκολλο έκδοσης 0. Για να εισαχθεί το module από την βιβλιοθήκη της python, χρησιμοποιείται η εντολή import pickle. Με τις μεθόδους pickle.dump() και pickle.load() αντίστοιχα, γίνεται pickling και unpickling του αντικειμένου, που ορίζεται στην παρένθεση.

Στην πρακτική εφαρμογή του pickle, θα δημιουργηθεί ένα λεξικό, που θα έχει ως λέξη κλειδί ένα νομό της Πελοποννήσου και τιμή την πρωτεύουσα κάθε νομού. Στη συνέχεια θα γίνει pickle το λεξικό και αργότερα θα εμφανιστεί. Εισάγεται αρχικά η pickle για να χρησιμοποιηθεί αργότερα. Ύστερα, δημιουργείται η μεταβλητή prarxeio, που της εκχωρούνται τα δεδομένα της pr και στη συνέχεια εισάγονται τα δεδομένα του λεξικού.

```
>>> import pickle
>>> prarxeio = 'pr.data'
>>>pr={'Axaias':'Patra','Korinthias':'Korinthos','Argolidas':'Nafplio','Arkadias':'Tripoli','Ileias'
:'Pyrgos','Messinias':'Kalamata','Lakonias':'Sparti'}
```

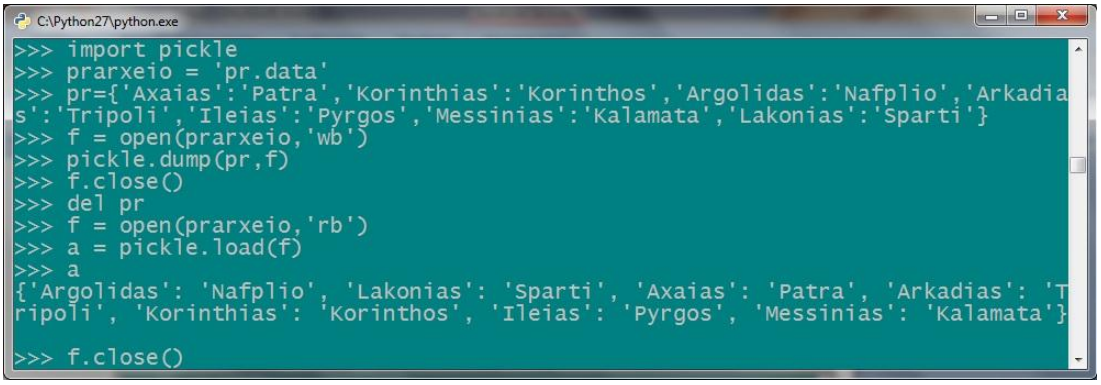
Εν συνεχεία, ανοίγεται το δυαδικό αρχείο prarxeio, με τη λειτουργία της εγγραφής και με την εντολή pickle.dump(pr,f), γίνεται pickling της μεταβλητής pr από το αρχείο f.

```
>>> f = open(prarxeio,'wb')
>>> pickle.dump(pr,f)
>>> f.close()
>>> del pr
```

Μόλις κλείσει το αρχείο, διαγράφεται και το λεξικό (μεταβλητή pr), που είχε δημιουργηθεί στην αρχή, για να διαπιστωθεί αν έχει εφαρμοστεί σωστά το pickling. Αφού ξανανοιχθεί το δυαδικό αρχείο, αυτή τη φορά όμως για ανάγνωση, γίνεται ανάκτηση των δεδομένων. Χρησιμοποιώντας τη μέθοδο της pickle load, φορτώνονται τα δεδομένα σε μια μεταβλητή a και στο τέλος εμφανίζονται στην οθόνη.

```
>>> f = open(prarxeio,'rb')
>>> a = pickle.load(f)
>>> a
{'Argolidas': 'Nafplio', 'Lakonias': 'Sparti', 'Axaias': 'Patra', 'Arkadias': 'Tripoli', 'Korinthias':
'Korinthos', 'Ileias': 'Pyrgos', 'Messinias': 'Kalamata'}
>>> f.close()
```

[23]Λεβεντέας Δ. (2010), «Εκμάθηση python βήμα βήμα», (σελ. 112), Ελληνική κοινότητα Προγραμματιστών python – Ανάκτηση στις 14-10-2011 από http://python.org.gr/index.php?option=com_phocadownload&view=category&id=3:tutorials&Itemid=58



Εικόνα 104 – Παράδειγμα με τη μέθοδο pickle.

Όπως διαπιστώθηκε, η ανάκτηση του λεξικού που δημιουργήθηκε νωρίτερα, έγινε χωρίς να υπάρχει αλλοίωση των δεδομένων και χωρίς να χαθεί η τιμή κάποιου από τα κλειδιά.

6.5 Άσκηση στα αρχεία.

Να δημιουργηθεί ένα αρχείο και να εισαχθούν δύο λεξικά με ζευγάρια νομούς και πρωτεύουσες, για την Πελοπόννησο και την Ήπειρο αντίστοιχα. Στη συνέχεια, να γίνει ενοποίηση σε ένα λεξικό, με όλα τα παραπάνω δεδομένα. Να δημιουργηθεί μετά μια λίστα, αποτελούμενη από τις τιμές του λεξικού. Να ανοιχθεί εν συνεχεία το αρχείο, σε αντίστοιχη λειτουργία και να προστεθεί μια εντολή εμφάνισης της λίστας με τα ονόματα των πόλεων και τέλος, να διαβαστεί όλο το αρχείο ανά γραμμή.
Για να μην αλλοιωθούν τα δεδομένα από το λεξικό, θα χρησιμοποιηθεί το module pickle.

Η λύση της άσκησης, βρίσκεται στο παράρτημα.

ΚΕΦΑΛΑΙΟ 7ο
Εξαιρέσεις.

Όπως έχει διαπιστωθεί στα μέχρι τώρα παραδείγματα, η python εμφανίζει διάφορα μηνύματα όταν υπάρχει κάποιο πρόβλημα στην εκτέλεση μιας εντολής. Ανάλογα λοιπόν με τη φύση του προβλήματος ξεχωρίζονται δύο κατηγορίες, οι εξαιρέσεις και τα σφάλματα. Σε κάθε εξαίρεση ή σφάλμα, η python σταματάει να εκτελεί τον κώδικα μέχρι να διορθωθεί το πρόβλημα.

7.1 Σφάλματα.

Σφάλματα είναι οι περιπτώσεις, που υπάρχουν συντακτικά λάθη, όπως αυτά που θα κάνει κάποιος αρχάριος με την python. Παρακάτω, έγινε ένα συντακτικό λάθος, επειδή εκτελέστηκε η εντολή printt αντί για print.

```
>>> a = 22
>>> printt a
  File "<stdin>", line 1
    printt a
          ^
SyntaxError: invalid syntax
```

Πάντα στην python εμφανίζεται μήνυμα για να ενημερώσει τον προγραμματιστή, για το πρόβλημα και το σημείο, που αυτό έχει εντοπιστεί. Αναφέρεται, ότι υπάρχει σφάλμα στη line 1 και το περιγράφει: SyntaxError: invalid syntax.

7.2 Εξαιρέσεις.

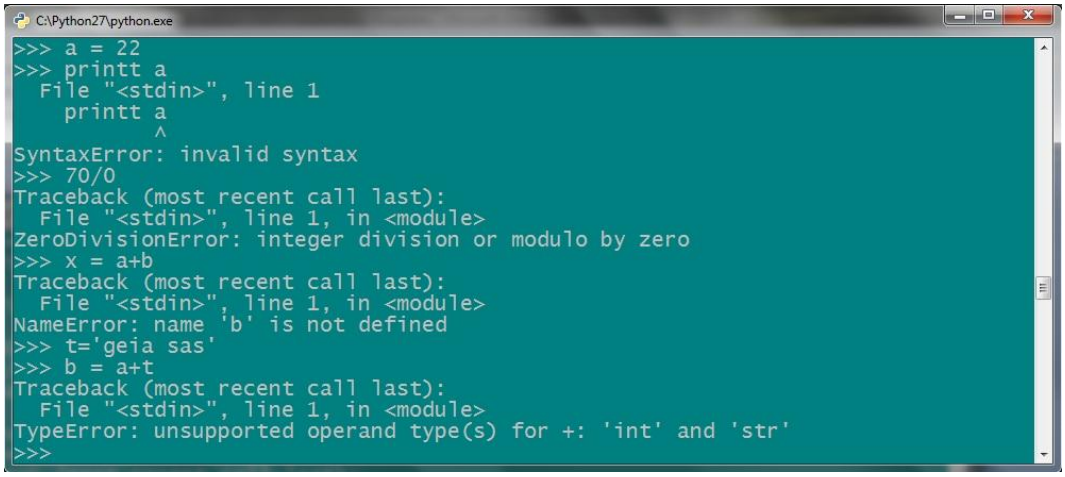
Στις εξαιρέσεις υπάρχουν διάφορα σφάλματα, που μπορεί να προκύψουν. Παρουσιάζεται εξαίρεση και σε περίπτωση, που είναι αδύνατη η εκτέλεση μιας εντολής. Χαρακτηριστικό είναι το παράδειγμα της διαίρεσης ενός αριθμού με το 0. Δεν υπάρχει συντακτικό λάθος, αλλά εμφανίζει το ZeroDivisionError σφάλμα, επειδή έχει δοθεί στον παρανομαστή το 0.

```
>>> 70/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Παρακάτω παρουσιάζονται μερικές περιπτώσεις εξαιρέσεων, που συναντώνται στην python. Στο πρώτο παράδειγμα, επιστρέφει την εξαίρεση NameError, επειδή δεν έχει οριστεί η μεταβλητή b.

```
 >>> x = a+b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
>>> t='geia sas'
>>> b = a+t
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Σε αυτήν την περίπτωση, εμφανίζεται το TypeError, επειδή επιχειρήθηκε να γίνει πρόσθεση αλφαριθμητικού με ακέραιο αριθμό.

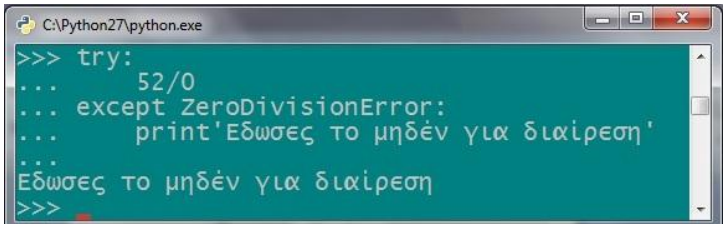


Εικόνα 105 – Παραδείγματα σφαλμάτων και εξαιρέσεων.

7.3 Χειρισμός εξαιρέσεων.

Για τη διαχείριση των εξαιρέσεων, χρησιμοποιείται η δομή try..except. Στο παράδειγμα που ακολουθεί, γίνεται διαχείριση της εξαίρεσης, για τη διαίρεση ενός ακεραίου με το 0. Εδώ, ορίζεται από τον προγραμματιστή, το μήνυμα που θα εμφανιστεί σε τέτοια περίπτωση (εξαίρεση ZeroDivisionError).

```
>>> try:
...     52/0
... except ZeroDivisionError:
...     print'Έδωσες το μηδέν για διαίρεση'
...
Έδωσες το μηδέν για διαίρεση
```



Εικόνα 106 – Διαχείριση μιας εξαίρεσης.

Η try, εκτελεί τις εντολές που υπάρχουν στο block της και αν χρειαστεί να καλέσει μια εξαίρεση, εμφανίζει το μήνυμα που έχει οριστεί, αντί αυτής. Η δομή try επίσης, μπορεί να υποστηρίξει και περαιτέρω εντολές με το else. Το else περιέχει εντολές που θα εκτελεστούν σε περίπτωση που δεν υπάρχει εξαίρεση. Παρέχεται η δυνατότητα να υπάρχουν πολλά except, που θα χειρίζονται περισσότερες εξαιρέσεις με διαφορετικό τρόπο (διαφορετικά μηνύματα).

Στο except μπορεί να εξεταστεί και μια λίστα εξαιρέσεων, που έχει προκαθοριστεί. Σε περίπτωση, που έχει δεν προσδιοριστεί κάποια, τότε αναφέρεται σε όλες τις εξαιρέσεις. Παρακάτω, φαίνεται πως είναι η δομή της try, ανεπτυγμένη.

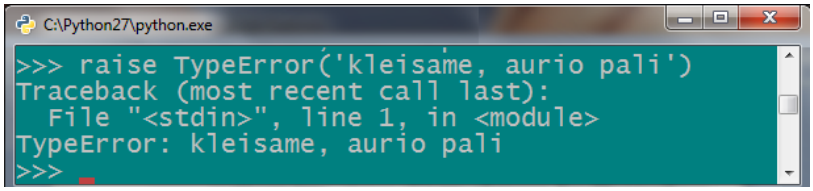
```
try:
  εντολές…
except εξαίρεση_a:
```

```
  print ...
except (εξαίρεση _b, εξαίρεση _c, εξαίρεση_d):
  print …
else:
  εντολές ..
```

7.4 Ανάδειξη εξαιρέσεων (raise).

Η εντολή raise, αναδεικνύει κάποια εξαίρεση ανά πάσα στιγμή. Δεν έχει σημασία τι είδος εξαίρεσης θα δηλωθεί, αφού λειτουργεί με όλες τις εξαιρέσεις NameError, TypeError κλπ. Με αυτή την εντολή, αναγκάζεται η python να εμφανίσει ένα σφάλμα.

```
>>> raise NameError('kleisame, aurio pali')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: kleisame, aurio pali
```



Εικόνα 107 – Ανάδειξη εξαίρεσης με το raise.

7.5 Δημιουργία εξαιρέσεων.

Για να δημιουργηθεί μια νέα εξαίρεση από την αρχή, θα χρησιμοποιηθούν οι κλάσεις. Η νέα κλάση που θα δημιουργηθεί, θα έχει και το όνομα της καινούργιας εξαίρεσης και για να κληρονομεί τα χαρακτηριστικά των εξαιρέσεων, πρέπει να οριστεί ως υποκλάση της Exception. Όταν τελειώσει αυτή η διαδικασία, θα έχει κατασκευαστεί μια νέα εξαίρεση και θα καλείται με την εντολή raise. Αναλύεται η όλη διαδικασία της δημιουργίας μιας εξαίρεσης, στο παρακάτω παράδειγμα.

Θα δημιουργηθεί μια εξαίρεση, που ανάλογα με την ημέρα που θα δηλώνεται, θα ενημερώνει το χρήστη, αν η γραμμή εντολών απεργεί ή όχι τη συγκεκριμένη ημέρα. Αν απεργεί, θα λαμβάνεται η εξαίρεση. Αρχικά, δημιουργείται η κλάση Apergia, που θα κληρονομεί από την κλάση Exception.

```
>>> class Apergia(Exception):
...     def __init__(self,value):
...         Exception.__init__(self)
...         self.v = value
...     def __str__(self):
...         return repr('Exoume 24ori apergia!! De tha ektelestei kamia entoli.')
...
```

Χρησιμοποιείται η ειδική μέθοδος __str__(), για να εισαχθεί το μήνυμα που θα εμφανίζεται στο χρήστη, σε περίπτωση που πραγματοποιείται αυτή η εξαίρεση. Αν ο προγραμματιστής, δεν επιθυμεί να εμφανίζεται ειδικό μήνυμα στην εξαίρεση, θα αντικατασταθεί το return repr('Exoume 24ori apergia!! De tha ektelestei kamia entoli.'), με το return repr(self.v). Αφού δημιουργηθεί η κλάση, γράφεται εν συνεχεία ο υπόλοιπος κώδικας και θα στο τέλος, θα γίνει raise του exception στο σημείο που χρειάζεται.

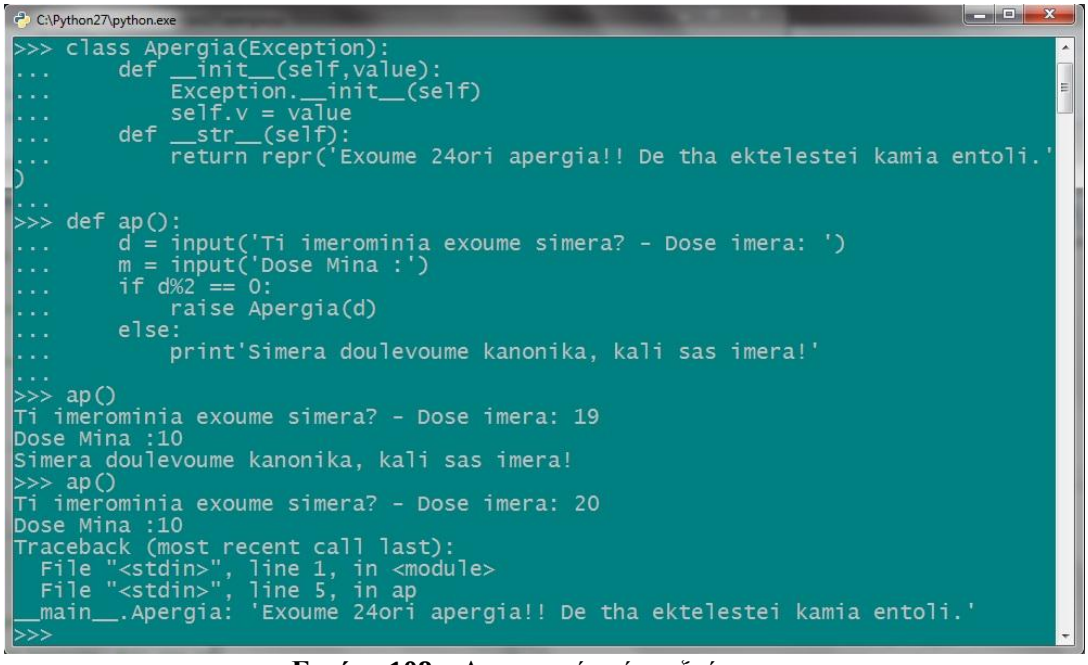
Θα προστεθεί η συνάρτηση ap() και θα γραφούν οι εντολές μέσα στο block, για να μπορεί να εκτελεστεί ο κώδικας πολλές φορές (για να γίνουν πολλές δοκιμές).

```
>>> def ap():
...     d = input('Ti imerominia exoume simera? - Dose imera: ')
...     m = input('Dose Mina :')
...     if d%2 == 0:
...         raise Apergia(d)
...     else:
...         print'Simera doulevoume kanonika, kali sas imera!'
...
```

Προσδιορίζεται με την if σε ποιες περιπτώσεις απεργεί η python (θα απεργεί αρκετά συχνά). Με την εντολή raise Apergia(d) αναδεικνύεται η εξαίρεση και το αποτέλεσμα της, παρουσιάζεται στην οθόνη του χρήστη. Αν δεν γίνεται εκείνη την ημέρα απεργία, θα επιστρέφεται αντίστοιχο μήνυμα. Παρακάτω, εκτελείται η συνάρτηση που μόλις ορίστηκε.

```
>>> ap()
Ti imerominia exoume simera? - Dose imera: 19
Dose Mina :10
Simera doulevoume kanonika, kali sas imera!
>>> ap()
Ti imerominia exoume simera? - Dose imera: 20
Dose Mina :10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in ap
__main__.Apergia: 'Exoume 24ori apergia!! De tha ektelestei kamia entoli.'
```

Παρατηρείται ότι στη δεύτερη ημερομηνία που έδωσε ο χρήστης, η python απεργεί και του επιστράφηκε σφάλμα και μήνυμα που εξηγεί τι έγινε.

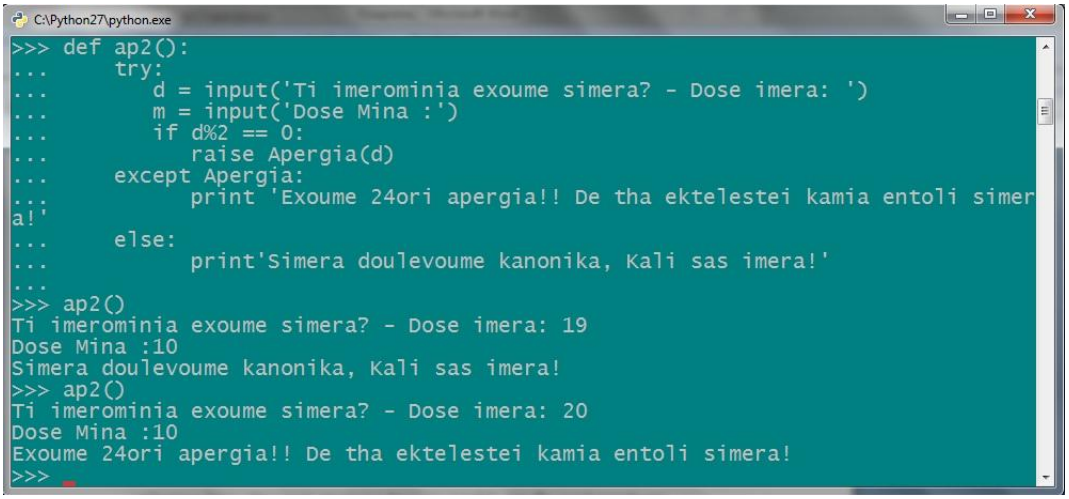


Εικόνα 108 – Δημιουργία νέας εξαίρεσης.

Θα δημιουργηθεί και μια δεύτερη συνάρτηση, που θα χειρίζεται την εξαίρεση με τη δομή try..except, που παρουσιάστηκε στην παραπάνω ενότητα.

```
>>> def ap2():
...     try:
...         d = input('Ti imerominia exoume simera? - Dose imera: ')
```

```
...       m = input('Dose Mina :')
...       if d%2 == 0:
...         raise Apergia(d)
...     except Apergia:
...         print 'Exoume 24ori apergia!! De tha ektelestei kamia entoli simera!'
...     else:
...         print'Simera doulevoume kanonika, Kali sas imera!'
...
>>> ap2()
Ti imerominia exoume simera? - Dose imera: 19
Dose Mina :10
Simera doulevoume kanonika, Kali sas imera!
>>> ap2()
Ti imerominia exoume simera? - Dose imera: 20
Dose Mina :10
Exoume 24ori apergia!! De tha ektelestei kamia entoli simera!
```



Εικόνα 109 – Διαχείριση της νέας εξαίρεσης.

Όταν καλείται η εξαίρεση με το raise, δεν εκτελείται η κλάση Apergia, αλλά εμφανίζει το μήνυμα, που είναι γραμμένο στο except Apergia, αντί να εκτελέσει το σφάλμα. Στο else της δομής try είναι οι εντολές, που θα εκτελεστούν σε περίπτωση, όταν δεν υπάρχει εξαίρεση.

7.6 Finally block.

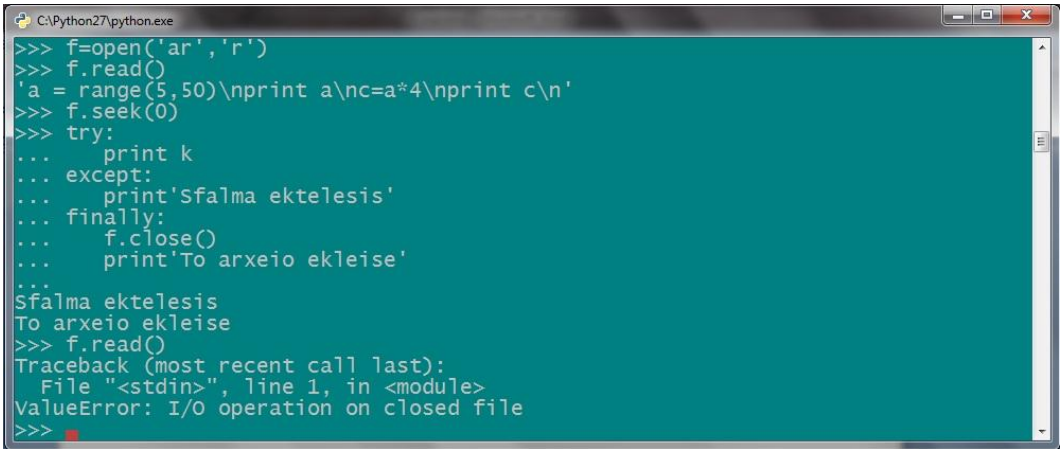
Σε προηγούμενο κεφάλαιο είχε αναφερθεί πως, όταν ανοίγει ένα αρχείο με το open(), πρέπει στο τέλος κλείνει με την εντολή close(). Όταν χρησιμοποιείται όμως η δομή try σε ένα αρχείο, πρέπει να ληφθούν κάποια μέτρα, ώστε να κλείσει στο τέλος το αρχείο που άνοιξε. Για να επιτευχθεί αυτό, εφαρμόζεται ένα block που ονομάζεται finally. Οι εντολές που περιέχει θα εκτελούνται σε κάθε περίπτωση, όπως στο παράδειγμα που ακολουθεί:

```
>>> f=open('ar','r')
>>> f.read()
'a = range(5,50)\nprint a\nc=a*4\nprint c\n'
>>> f.seek(0)
>>> try:
...   print k
```

```
... except:
...    print'Sfalma ektelesis'
... finally:
...    f.close()
...    print'To arxeio ekleise'
...
Sfalma ektelesis
To arxeio ekleise
```

Παρά το σφάλμα που παρουσιάστηκε, η εντολή που βρίσκεται μέσα στο block του finally εκτελέστηκε κανονικά. Με αυτό το πείραμα επιβεβαιώνεται ότι αυτές οι εντολές, εκτελούνται κανονικά. Αυτό αποδεικνύεται και με την εκτέλεση της read:

```
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file
```



Εικόνα 110 – Παράδειγμα με το finally block.

Η εντολή finally θα εκτελείται πάντα και ας υπάρχει το else μετά την εξαίρεση. Σε ενδεχόμενο σφάλματος, δε θα τρέξουν οι εντολές που βρίσκονται στο else, αλλά σε κάθε περίπτωση το finally θα εκτελείται.

7.7 Ασκήσεις στις εξαιρέσεις.

A. Να δημιουργηθεί πρόγραμμα, το οποίο θα ελέγχει αν υπάρχουν σε μια λίστα αρνητικοί αριθμοί και θα εμφανίζει μήνυμα, ότι υπάρχει αρνητικός αριθμός.

B. Σε μια κλήρωση, ο νικητής θα εξάγεται από μια λίστα ονομάτων. Να δημιουργηθεί μια νέα εξαίρεση, που σε περίπτωση που ο νικητής είναι συγκεκριμένο όνομα, θα εμφανίζει μήνυμα ότι είναι ο διοργανωτής και δεν μπορεί να κληρωθεί. Έτσι, θα καλείται ο χρήστης να κληρώσει ένα άλλο όνομα. Τον νικητή θα τον επιλέγει ο χρήστης αφού θα δίνει ένα τυχαίο αριθμό, που θα αντιστοιχεί σε κάποιο στοιχείο της λίστας.

C. Να καταχωρηθούν σε μια λίστα οι μέσες τιμές της αμόλυβδης βενζίνης για 10 περιοχές της Ελλάδας, του προηγούμενου μήνα. Να δημιουργηθεί εξαίρεση για την περίπτωση, που καταχωρείται τιμή μεγαλύτερη από 10 ευρώ και να εμφανίζεται μήνυμα.

Οι λύσεις βρίσκονται στο παράρτημα.

ΚΕΦΑΛΑΙΟ 8ο
Λίγα ακόμα για την python.

Τελειώνοντας την μελέτη της γλώσσα προγραμματισμού python και των μοντέλων που υποστηρίζει, θα γίνει μια αναφορά στην εντολή λάμδα (Lambda). Η εντολή αυτή, είναι χαρακτηριστικό του συναρτησιακού μοντέλου προγραμματισμού, ενός προγραμματιστικού παραδείγματος (μοντέλου), που υπολογίζει τη τιμή μιας μεταβλητής βάσει μαθηματικών συναρτήσεων και όχι βάσει των αλλαγών μιας κατάστασης. Το συναρτησιακό μοντέλο παρουσιάστηκε στις δομές δεδομένων με τις εντολές filter, map και reduce. Αυτή η εντολή, δημιουργήθηκε από τον λ-λογισμό και για αυτό η python ονόμασε αυτού του είδους τις εκφράσεις λάμδα.

8.1 Εκφράσεις Λάμδα.

Για να εφαρμοστούν οι εντολές λάμδα, χρησιμοποιούνται οι εκφράσεις (expressions) της python. Οι εκφράσεις, υπολογίζουν μια τιμή με βάση τις τιμές των άλλων μεταβλητών. Μια έκφραση αποτελείται από το όνομα της έκφρασης (lambda) και τις παραμέτρους. Στη συνέχεια, μετά από άνω κάτω τελεία (:), θα γράφεται η αριθμητική παράσταση, που πρέπει να υπολογιστεί.

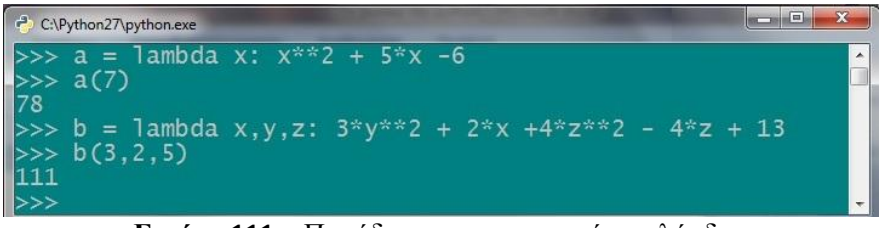
Στο παράδειγμα που ακολουθεί, δημιουργείται η μεταβλητή a στην οποία ορίζεται η έκφραση lambda, με παράμετρο το x. Στη συνέχεια, εισάγεται η συνάρτηση που πρέπει να υπολογιστεί.[24]

```
>>> a = lambda x: x**2 + 5*x - 6
>>> a(7)
78
```

Εκτελείται η συνάρτηση a, δίνοντας στο x την τιμή 7 και επιστρέφεται αμέσως το αποτέλεσμα, που είναι το 78.

Ακολουθεί και ένα παράδειγμα με τρείς παραμέτρους, που είναι οι x, y και z.

```
>>> b = lambda x,y,z: 3*y**2 + 2*x +4*z**2 - 4*z + 13
>>> b(3,2,5)
111
```



Εικόνα 111 – Παράδειγμα με τις εκφράσεις λάμδα.

Αξίζει να σημειωθεί, πως η έκφραση lambda έχει προτεραιότητα έναντι των άλλων πράξεων.

[24] Pilgrim M. (2004), Dive into python, Using lambda functions. Ανάκτηση στις 2-11-2011 από http://www.diveintopython.net/power_of_introspection/lambda_functions.html

8.2 Άσκηση στις εκφράσεις λάμδα.

Να δημιουργηθούν εκφράσεις λ, για τον υπολογισμό του εμβαδού του τριγώνου, του τετραγώνου, του ορθογωνίου παραλληλογράμμου και τραπεζίου. Στη συνέχεια να υπολογιστούν τα παρακάτω εμβαδά:

a. Τρίγωνο με βάση 12 εκ. και ύψος 8 εκ.
b. Τετράγωνο με πλευρά 8 εκ.
c. Ορθογώνιο με πλευρές 4 εκ. και 7 εκ.
d. Τραπέζιο παραλληλόγραμμο με βάσεις 15 εκ. και 11 εκ. αντίστοιχα και ύψος 9 εκ.

Η λύση της άσκησης βρίσκεται στο παράρτημα.

ΕΠΙΛΟΓΟΣ

Η παρούσα εργασία, είχε σκοπό την μελέτη της γλώσσας προγραμματισμού python και των προγραμματιστικών μοντέλων που υποστηρίζει. Ύστερα από μια ιστορική αναδρομή και αναφορά στους λόγους που οδήγησαν στην δημιουργία της, γίνεται μια ανάλυση των χαρακτηριστικών της γλώσσας. Εν συνεχεία, παρουσιάζεται ο τρόπος συγγραφής των εντολών και μέσω παραδειγμάτων, δίνεται μια σαφής εικόνα για τον τρόπο λειτουργίας της.

Εδώ, μελετήθηκαν τα προγραμματιστικά μοντέλα που μπορούν να εφαρμοστούν στην python και πως πετυχαίνεται η δημιουργία εφαρμογών σε κάθε μοντέλο. Επειδή ο καλύτερος τρόπος για να εμπεδωθεί ο τρόπος λειτουργίας μιας γλώσσας προγραμματισμού, είναι ο πειραματισμός και η μελέτη. Για αυτό το λόγο, έχουν δημιουργηθεί και ασκήσεις σε κάθε κεφάλαιο, οι οποίες επιτρέπουν στον αναγνώστη να πειραματιστεί με την γλώσσα και τα χαρακτηριστικά της. Οι λύσεις των ασκήσεων βρίσκονται στο παράρτημα για να μπορεί να γίνει έλεγχος για την σωστή επίλυση τους και την κάλυψη των ενδεχόμενων κενών, σχετικά με τον τρόπο λειτουργίας των διαφόρων μοντέλων που υποστηρίζει.

Κλείνοντας, η μελέτη αυτή αποτελεί μια βάση των όσων μπορούν να γίνουν με την python. Οι δυνατότητές της δεν περιορίζονται σε αυτά που αναφέρθηκαν σε αυτή τη μελέτη, αλλά με βάση αυτά μπορεί να δημιουργήσει ο προγραμματιστής, προγράμματα καινοτόμα και πρωτοποριακά. Η μελέτη μιας ανοικτού κώδικα και διαρκώς αναπτυσσόμενης γλώσσας προγραμματισμού δεν μπορεί να έχει τέλος, επειδή καθημερινά, δημιουργούνται νέα δεδομένα, που θα απαιτήσουν την εξακολούθηση της μελέτης. Ο καλύτερος τρόπος παρακολούθησης των νέων εξελίξεων σε μια ανοικτή γλώσσα είναι η παραγωγή κώδικα και η δημιουργία προγραμμάτων, έτσι ώστε να αποτελέσει ο ίδιος προγραμματιστής, ο δημιουργός των εξελίξεων και να είναι συνάμα ο μελετητής των νέων αλλαγών.

ΒΙΒΛΙΟΓΡΑΦΙΑ ΚΑΙ ΠΗΓΕΣ

http://dide.flo.sch.gr/Plinet/Tutorials/Tutorials-Python-Introduction.html
http://docs.python.org/
http://greektuts.net/category/programming/python/
http://groups.google.com/group/alt.sources/browse_thread/thread/3b6652abb0e23b03/53ee3620bc53c281
http://python.org.gr
http://python.org.gr/uploads
http://python.org/
http://www.python-course.eu/object_oriented_programming.php
http://python-history.blogspot.com/
Pilgrim M, *Dive into python*. Online Διαθέσιμο: http://diveintopython.net/
Swaroop CH, *A Byte of Python* (Ελληνική μετάφραση). Online. Διαθέσιμο: http://www.swaroopch.com/notes/Python_el:%CE%A0%CE%B5%CF%81%CE%B9%CE%B5%CF%87%CF%8C%CE%BC%CE%B5%CE%BD%CE%B1 Ανάκτηση 15-7-2011
Λεβεντέας Δ, *Εκμάθηση python βήμα βήμα,* Ελληνική κοινότητα προγραμματιστών python, Online. Διαθέσιμο- http://python.org.gr/index.php?option=com_phocadownload&view=category&id=3:tutorials&Itemid=58#

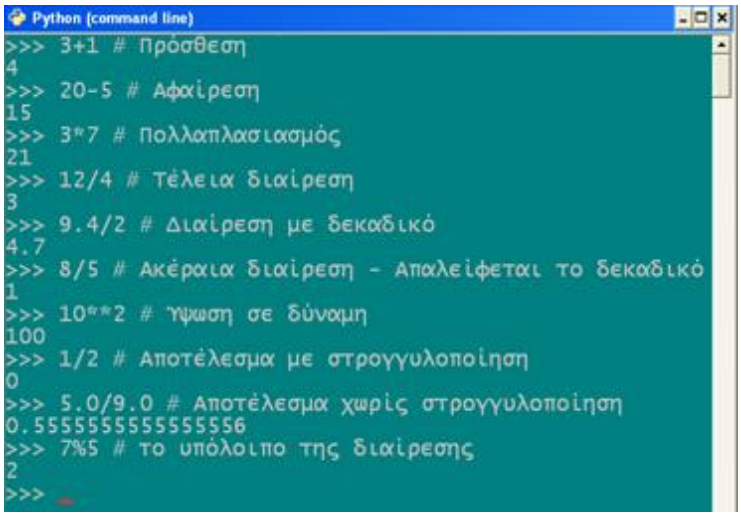
ΠΑΡΑΡΤΗΜΑ

Λύσεις των ασκήσεων.

I. Ασκήσεις στα βασικά της python.

A. Λύση

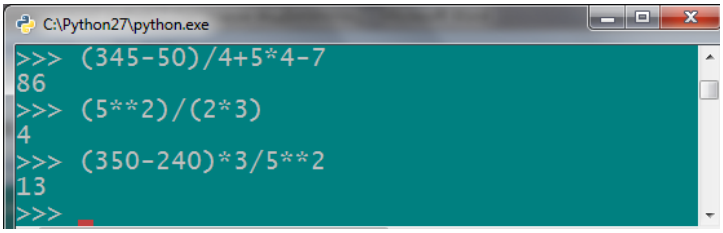
```
>>> 3+1 # Πρόσθεση
4
>>> 20-5  # Αφαίρεση
15
>>> 3*7  # Πολλαπλασιασμός
21
>>> 12/4 # Τέλεια διαίρεση
3
9.4/2  # Διαίρεση με δεκαδικό
4.7
8/5  # Ακέραια διαίρεση – Απαλείφεται το δεκαδικό
1
10**2  # Ύψωση σε δύναμη
1 /2  # Αποτέλεσμα με στρογγυλοποίηση
0
5.0/9.0  # Αποτέλεσμα χωρίς στρογγυλοποίση
0.5555555555555556
7%5  # το υπόλοιπο της διαίρεσης
2
```



Εικόνα 112 – Βασικά: λύση της Α. Άσκησης.

B. Λύση

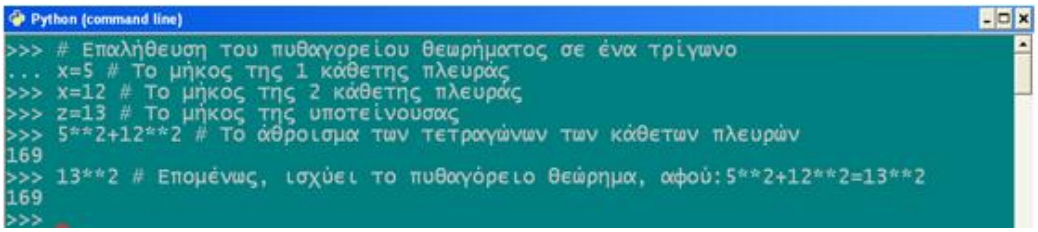
```
>>> (345-50)/4+5*4-7
    86
    >>> (5**2)/(2*3)
    4
    >>> (350-240)*3/5**2
    13
```



Εικόνα 113 - Βασικά: λύση της Β άσκησης.

C. Λύση

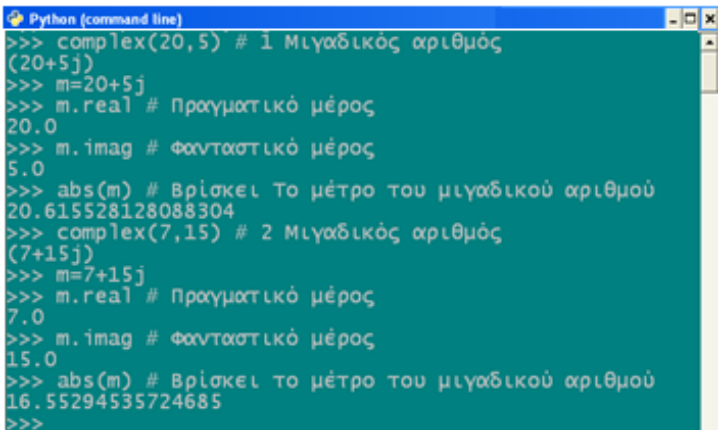
```
>>> # Επαλήθευση του πυθαγορείου θεωρήματος σε ένα τρίγωνο
… X=5 # Το μήκος της 1 κάθετης πλευράς
>>> X=12 # Το μήκος της 2 κάθετης πλευράς
>>> Z=13 # Το μήκος της υποτείνουσας
>>> 5**2+12**2 # Το άθροισμα των τετραγώνων των κάθετων πλευρών
169
>>> 13**2 # Επομένως, ισχύει το πυθαγόρειο θεώρημα, αφού:5**2+12**2=13**2
169
```



Εικόνα 114 - Βασικά: λύση της C άσκησης.

D. Λύση

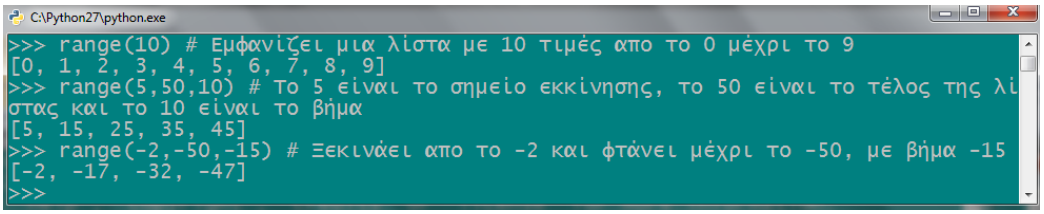
```
>>> Complex(20,5) # 1 Μιγαδικός αριθμός
(20+5j)
>>> m=20+5j
>>> m.real # Πραγματικό μέρος
20.0
>>> m.imag # Φανταστικό μέρος
5.0
>>> abs(m) # Βρίσκει το μέτρο του μιγαδικού αριθμού
20.615528128088304
>>> complex(7,15) # 2 Μιγαδικός αριθμός
(7+15j)
>>> m=7+15j
m.real # Πραγματικό μέρος
7.0
m.imag # Φανταστικό μέρος
15.0
>>> abs(m) # Βρίσκει το μέτρο του μιγαδικού αριθμού
16.55294535724685
```



Εικόνα 115 - Βασικά: λύση της D άσκησης.

E. Λύση

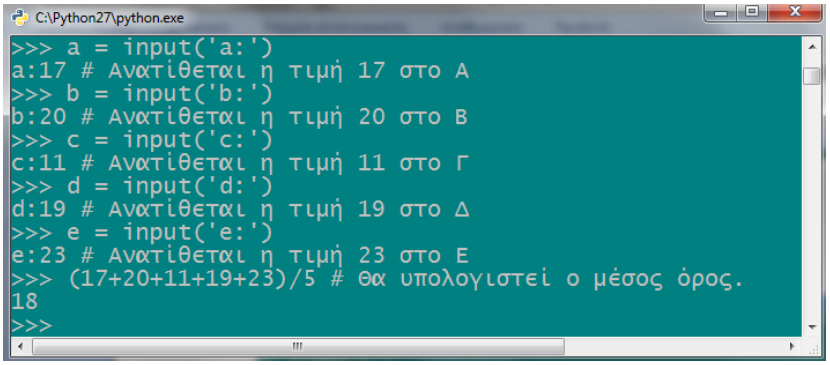
>>> range(10) # Εμφανίζει μια λίστα με 10 τιμές από το 0 μέχρι το 9
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5,50,10) # Το 5 είναι το σημείο εκκίνησης, το 50 είναι το τέλος της λίστας και το 10 είναι το βήμα.
[5, 15, 25, 35, 45]
>>> range(-2,-50,-15) # Ξεκινάει από το -2 και φτάνει μέχρι το -50, με βήμα -15.
[-2, -17, -32, -47]



Εικόνα 116 - Βασικά: λύση της E άσκησης.

F. Λύση

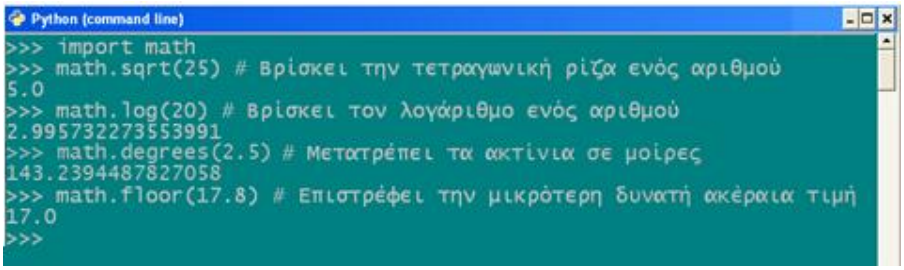
>>> a = input('a:')
a:17 # Ανατίθεται η τιμή 17 στο Α
>>> b = input('b:')
b:20 # Ανατίθεται η τιμή 20 στο Β
>>> c = input('c:')
c:11 # Ανατίθεται η τιμή 11 στο Γ
>>> d = input('d:')
d:19 # Ανατίθεται η τιμή 19 στο Δ
>>> e = input('e:')
e:23 # Ανατίθεται η τιμή 23 στο Ε
>>> (17+20+11+19+23)/5 # Θα υπολογιστεί ο μέσος όρος.
18



Εικόνα 117 - Βασικά: λύση της F άσκησης.

G. Λύση

```
>>> import math
>>> math.sqrt(25)  #  Βρίσκει την τετραγωνική ρίζα ενός αριθμού
5.0
>>> math.log(20) # Βρίσκει τον λογάριθμο ενός αριθμού
2.995732273553991
>>> math.degrees(2.5) # Μετατρέπει τα ακτίνα σε μοίρες
143.2394487827058
>>> math.floor(17.8) # Επιστρέφει την μικρότερη δυνατή ακέραια τιμή
17.0
```



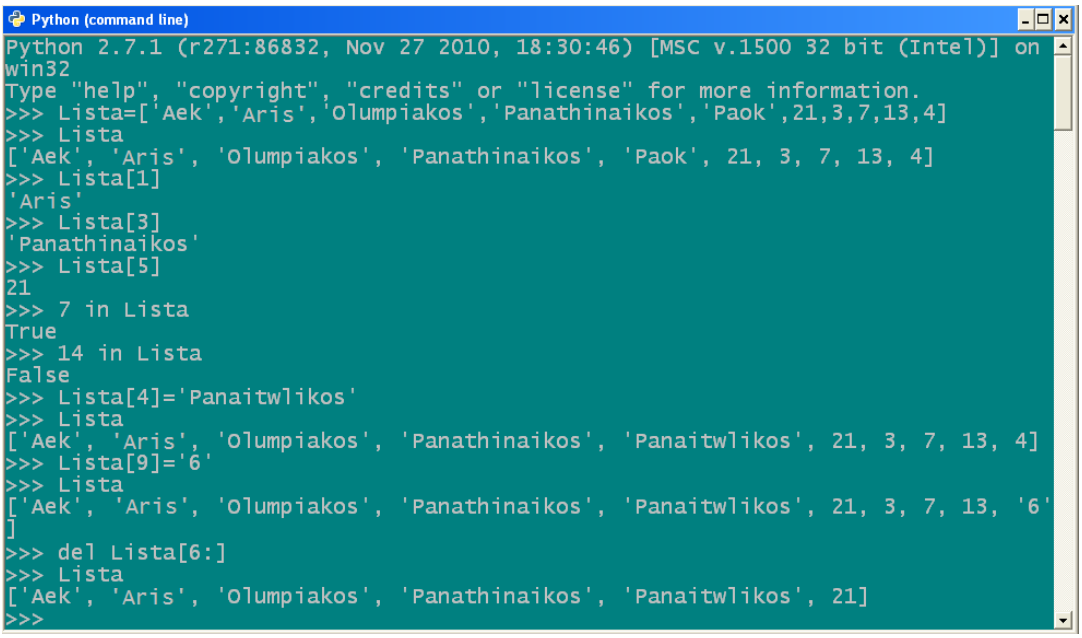
Εικόνα 118 - Βασικά: λύση της G άσκησης.

II. Ασκήσεις στις δομές δεδομένων.

A. Λύση

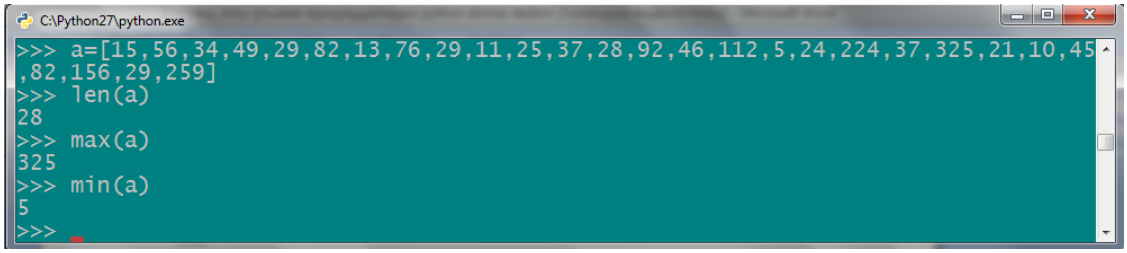
```
>>> Lista=[΄Αεκ΄,΄Aris΄,΄Olumpiakos΄,΄panathinaikos΄,΄paok΄,21,3,7,13,4]
>>> Lista
 [΄Αεκ΄, ΄Aris΄, ΄Olumpiakos΄, ΄panathinaikos΄, ΄paok΄, 21, 3, 7, 13, 4]
>>> Lista[1]
΄Aris΄
>>> Lista[3]
΄panathinaikos΄
>>> Lista[5]
21
>>> 7 in Lista
True
>>> 14 in Lista
False
>>> Lista[4]=΄panaitwlikos΄
```

>>> Lista
[΄Αεκ΄, ΄Aris΄, ΄Olumpiakos΄, ΄panathinaikos΄, ΄ panaitwlikos ΄, 21, 3, 7, 13, 4]
>>>Lista[9]=΄6΄
>>>Lista
[΄Αεκ΄, ΄Aris΄, ΄Olumpiakos΄, ΄panathinaikos΄, ΄ panaitwlikos ΄, 21, 3, 7, 13, 6]
>>> del lista[6:]
>>>Lista
[΄Αεκ΄, ΄Aris΄, ΄Olumpiakos΄, ΄panathinaikos΄, ΄ panaitwlikos΄, 21]



Εικόνα 119 – Δομές δεδομένων: λύση της Α άσκησης.

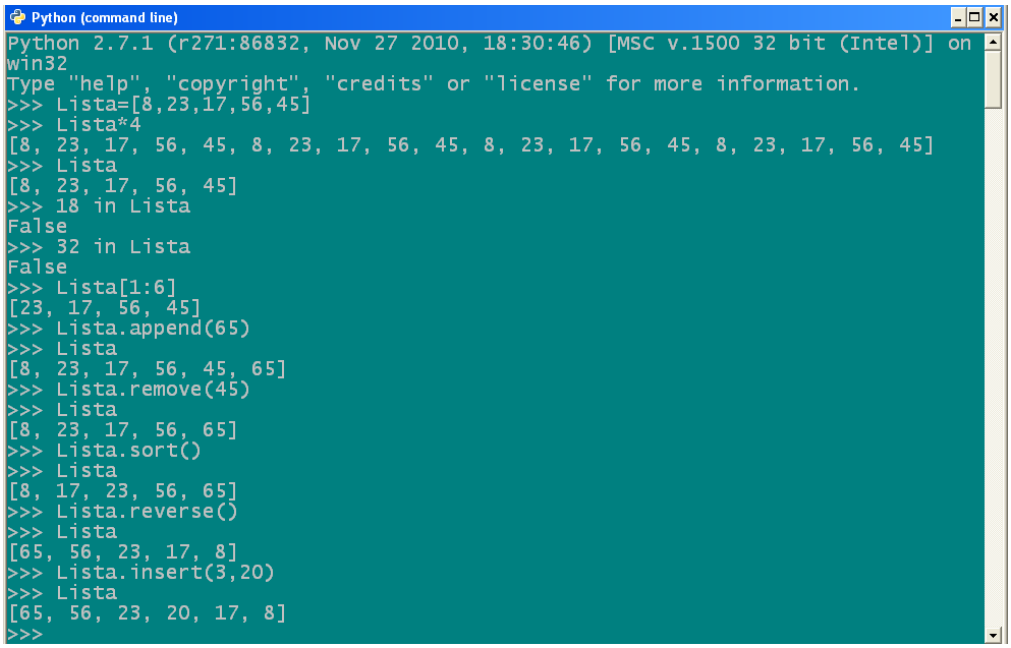
B. Λύση
>>>a=[15,56,34,49,29,82,13,76,29,11,25,37,28,92,46,112,5,24,224,37,325,21,10,45,82,156,29,259]
>>> len(a)
28
>>> max(a)
325
>>> min(a)
5



Εικόνα 120 - Δομές δεδομένων: λύση της Β άσκησης.

C. Λύση
>>> Lista=[8,23,17,56,45]
>>> Lista*4
[8, 23, 17, 56, 45, 8, 23, 17, 56, 45, 8, 23, 17, 56, 45, 8, 23, 17, 56, 45]

```
>>> Lista
[8,23,17,56,45]
>>> 18 in Lista
False
>>> 32 in Lista
False
>>> Lista[1:6]
[23, 17, 56, 45]
>>> Lista.append(65)
>>> Lista
[8, 23, 17, 56, 45, 65]
>>> Lista.remove(45)
>>> Lista
[8, 23, 17, 56, 65]
>>> Lista.sort()
>>> Lista
[8, 17, 23, 56, 65]
>>> Lista.reverse()
>>> Lista
[65, 56, 23, 17, 8]
>>> Lista.insert(3,20)
>>> Lista
[65, 56, 23, 20, 17, 8]
```

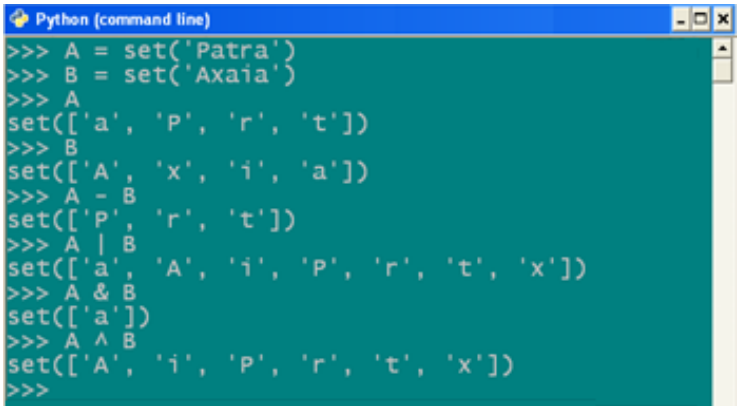


Εικόνα 121 - Δομές δεδομένων: λύση της C άσκησης.

D. Λύση

```
>>> A = set(´Patra´)
>>> B = set(´Axaia´)
>>> A
Set([´a´,´p´,´r´,´t´])
>>> B
```

Set([´A´,´x´,´i´,´a´])
>>> A-B
set([´p´,´r´,´t´])
>>> A | B
Set([´a´,´A´,´i´,´P´,´r´,´t´,´x´])
>>> A & B
set([´a´])
>>> A^B
set([´A´,´i´,´p´,´r´,´t´,´x´])



Εικόνα 122 - Δομές δεδομένων: λύση της D άσκησης.

E. Λύση

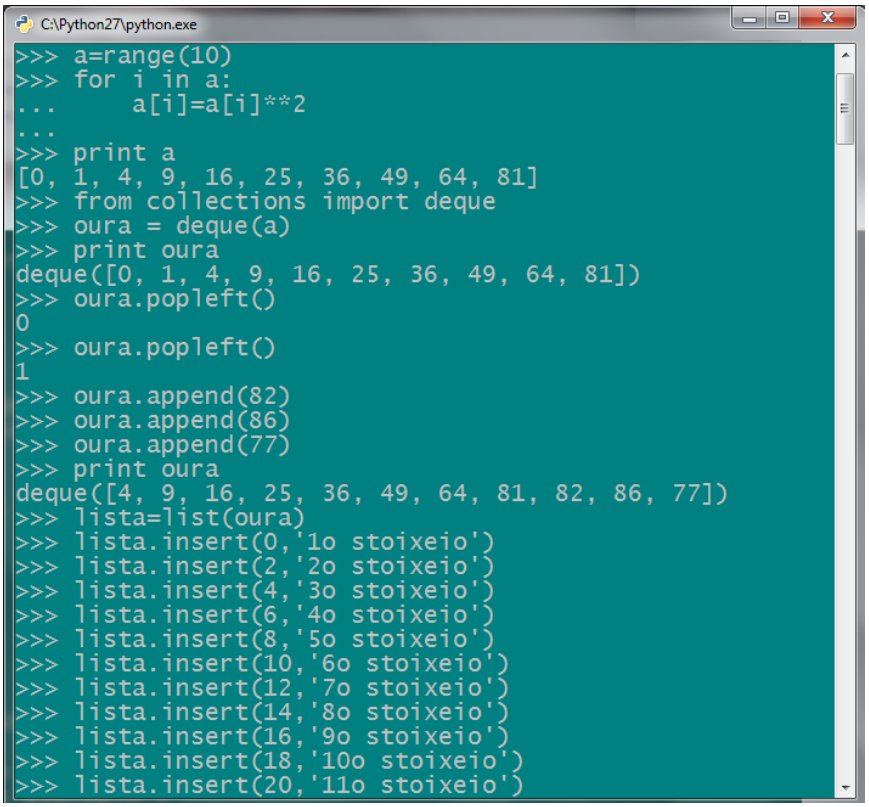
>>> mmm = {'metro','ilektrikos','leoforeio','tram'}
>>> m=set(mmm)
>>> m
set(['tram', 'ilektrikos', 'metro', 'leoforeio'])
>>> m2=m.copy()
>>> m2
set(['tram', 'ilektrikos', 'metro', 'leoforeio'])
>>> m.add('taxi')
>>> m.remove('tram')
>>> m
set(['taxi', 'ilektrikos', 'metro', 'leoforeio'])
>>> m2
set(['tram', 'ilektrikos', 'metro', 'leoforeio'])
>>> m<m2
False
>>> m.pop()
'taxi'
>>> m<m2
True
>>> m2.clear()
>>> m2
set([])

Εικόνα 123 - Δομές δεδομένων: λύση της Ε άσκησης.

F. Λύση

```
>>> a=range(10)
>>> for i in a:
...     a[i]=a[i]**2
...
>>> print a
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> from collections import deque
>>> oura = deque(a)
>>> print oura
deque([0, 1, 4, 9, 16, 25, 36, 49, 64, 81])
>>> oura.popleft()
0
>>> oura.popleft()
1
>>> oura.append(82)
>>> oura.append(86)
>>> oura.append(77)
>>> print oura
deque([4, 9, 16, 25, 36, 49, 64, 81, 82, 86, 77])
>>> lista=list(oura)
>>> lista.insert(0,'1o stoixeio')
>>> lista.insert(2,'2o stoixeio')
>>> lista.insert(4,'3o stoixeio')
>>> lista.insert(6,'4o stoixeio')
>>> lista.insert(8,'5o stoixeio')
>>> lista.insert(10,'6o stoixeio')
>>> lista.insert(12,'7o stoixeio')
>>> lista.insert(14,'8o stoixeio')
>>> lista.insert(16,'9o stoixeio')
>>> lista.insert(18,'10o stoixeio')
```

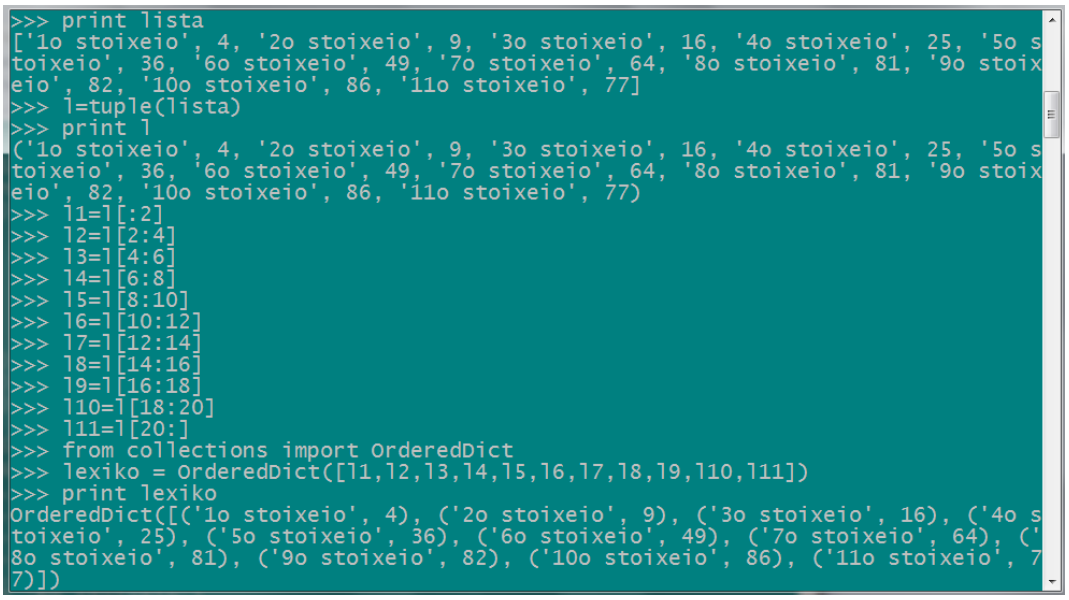
>>> lista.insert(20,'11o stoixeio')



Εικόνα 124 - Δομές δεδομένων: λύση της F άσκησης 1.

>>> print lista
['1o stoixeio', 4, '2o stoixeio', 9, '3o stoixeio', 16, '4o stoixeio', 25, '5o stoixeio', 36, '6o stoixeio', 49, '7o stoixeio', 64, '8o stoixeio', 81, '9o stoixeio', 82, '10o stoixeio', 86, '11o stoixeio', 77]
>>> l=tuple(lista)
>>> print l
('1o stoixeio', 4, '2o stoixeio', 9, '3o stoixeio', 16, '4o stoixeio', 25, '5o stoixeio', 36, '6o stoixeio', 49, '7o stoixeio', 64, '8o stoixeio', 81, '9o stoixeio', 82, '10o stoixeio', 86, '11o stoixeio', 77)
>>> l1=l[:2]
>>> l2=l[2:4]
>>> l3=l[4:6]
>>> l4=l[6:8]
>>> l5=l[8:10]
>>> l6=l[10:12]
>>> l7=l[12:14]
>>> l8=l[14:16]
>>> l9=l[16:18]
>>> l10=l[18:20]
>>> l11=l[20:]
>>> from collections import OrderedDict
>>> lexiko = OrderedDict([l1,l2,l3,l4,l5,l6,l7,l8,l9,l10,l11])
>>> print lexiko

OrderedDict([('1o stoixeio', 4), ('2o stoixeio', 9), ('3o stoixeio', 16), ('4o stoixeio', 25), ('5o stoixeio', 36), ('6o stoixeio', 49), ('7o stoixeio', 64), ('8o stoixeio', 81), ('9o stoixeio', 82), ('10o stoixeio', 86), ('11o stoixeio', 77)])



Εικόνα 125 - Δομές δεδομένων: λύση της F άσκησης 2.

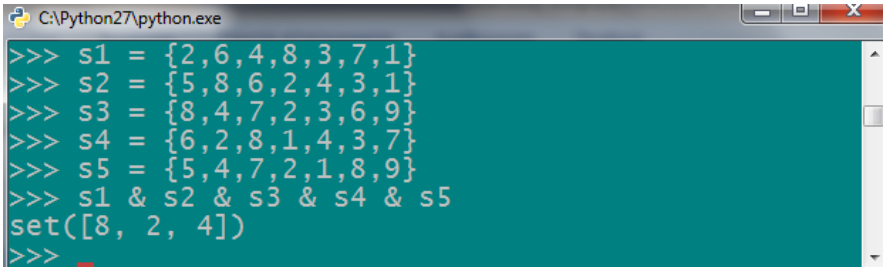
>>> lexiko2 = lexiko.copy()
>>> print lexiko2
OrderedDict([('1o stoixeio', 4), ('2o stoixeio', 9), ('3o stoixeio', 16), ('4o stoixeio', 25), ('5o stoixeio', 36), ('6o stoixeio', 49), ('7o stoixeio', 64), ('8o stoixeio', 81), ('9o stoixeio', 82), ('10o stoixeio', 86), ('11o stoixeio', 77)])
>>> for i in lexiko2:
... lexiko2[i]=lexiko2[i]*3
...
>>> print 'To proto lexiko einai:',lexiko
To proto lexiko einai: OrderedDict([('1o stoixeio', 4), ('2o stoixeio', 9), ('3o stoixeio', 16), ('4o stoixeio', 25), ('5o stoixeio', 36), ('6o stoixeio', 49), ('7o stoixeio', 64), ('8o stoixeio', 81), ('9o stoixeio', 82), ('10o stoixeio',86), ('11o stoixeio', 77)])
>>> print 'Kai to deftero einai:',lexiko2
Kai to deftero einai: OrderedDict([('1o stoixeio', 12), ('2o stoixeio', 27), ('3o stoixeio', 48), ('4o stoixeio', 75), ('5o stoixeio', 108), ('6o stoixeio', 147), ('7º stoixeio', 192), ('8o stoixeio', 243), ('9o stoixeio', 246), ('10o stoixeio', 258), ('11o stoixeio', 231)])

```
>>> lexiko2 = lexiko.copy()
>>> print lexiko2
OrderedDict([('1o stoixeio', 4), ('2o stoixeio', 9), ('3o stoixeio', 16), ('4o s
toixeio', 25), ('5o stoixeio', 36), ('6o stoixeio', 49), ('7o stoixeio', 64), ('
8o stoixeio', 81), ('9o stoixeio', 82), ('10o stoixeio', 86), ('11o stoixeio', 7
7)])
>>> for i in lexiko2:
...     lexiko2[i]=lexiko2[i]*3
...
>>> print 'To proto lexiko einai:',lexiko
To proto lexiko einai: OrderedDict([('1o stoixeio', 4), ('2o stoixeio', 9), ('3o
 stoixeio', 16), ('4o stoixeio', 25), ('5o stoixeio', 36), ('6o stoixeio', 49),
('7o stoixeio', 64), ('8o stoixeio', 81), ('9o stoixeio', 82), ('10o stoixeio',
86), ('11o stoixeio', 77)])
>>> print 'Kai to deftero einai:',lexiko2
Kai to deftero einai: OrderedDict([('1o stoixeio', 12), ('2o stoixeio', 27), ('3
o stoixeio', 48), ('4o stoixeio', 75), ('5o stoixeio', 108), ('6o stoixeio', 147
), ('7o stoixeio', 192), ('8o stoixeio', 243), ('9o stoixeio', 246), ('10o stoix
eio', 258), ('11o stoixeio', 231)])
>>>
```

Εικόνα 126 - Δομές δεδομένων: λύση της F άσκησης 3.

G. Λύση

```
>>> s1 = {2,6,4,8,3,7,1}
>>> s2 = {5,8,6,2,4,3,1}
>>> s3 = {8,4,7,2,3,6,9}
>>> s4 = {6,2,8,1,4,3,7}
>>> s5 = {5,4,7,2,1,8,9}
>>> s1 & s2 & s3 & s4 & s5
set([8, 2, 4])
```



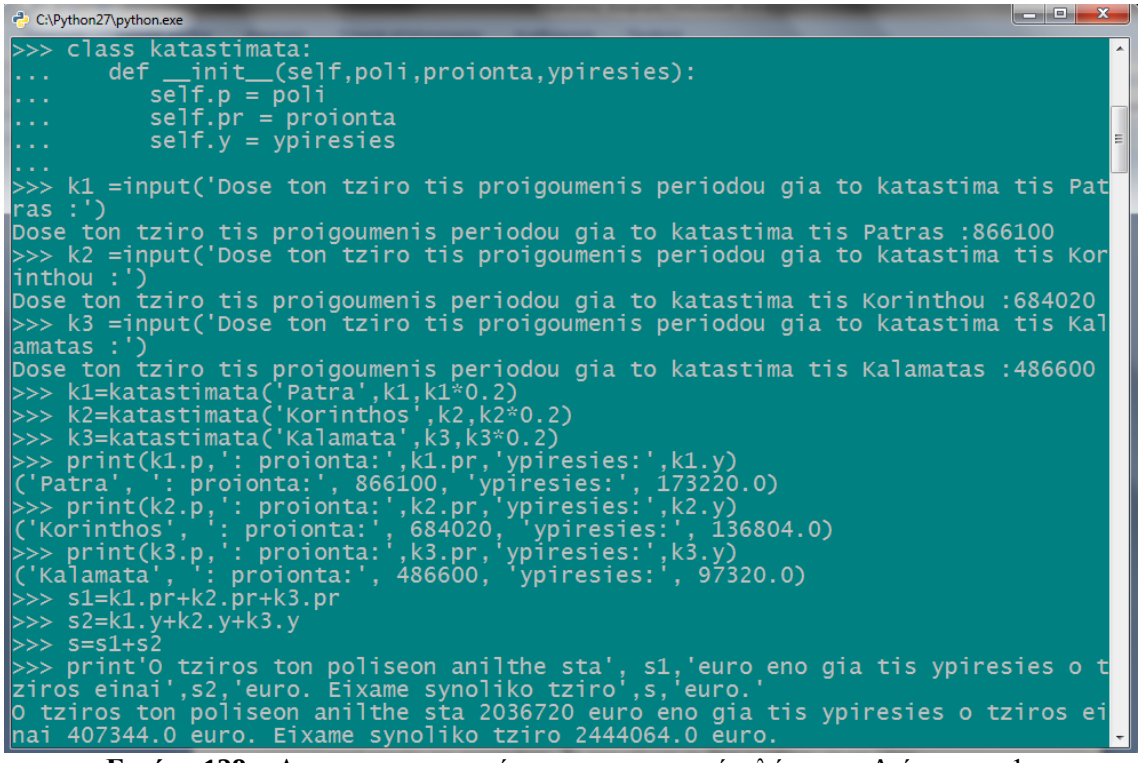
Εικόνα 127 - Δομές δεδομένων: λύση της G άσκησης.

III. Ασκήσεις στον αντικειμενοστραφή προγραμματισμό.

A. Λύση

```
>>> class katastimata:
...     def __init__(self,poli,proionta,ypiresies):
...         self.p = poli
...         self.pr = proionta
...         self.y = ypiresies
...
>>> k1 =input('Dose ton tziro tis proigoumenis periodou gia to katastima tis Patras :')
Dose ton tziro tis proigoumenis periodou gia to katastima tis Patras :866100
>>> k2 =input('Dose ton tziro tis proigoumenis periodou gia to katastima tis Korinthou :')
Dose ton tziro tis proigoumenis periodou gia to katastima tis Korinthou :684020
>>> k3 =input('Dose ton tziro tis proigoumenis periodou gia to katastima tis Kalamatas :')
Dose ton tziro tis proigoumenis periodou gia to katastima tis Kalamatas :486600
>>> k1=katastimata('Patra',k1,k1*0.2)
>>> k2=katastimata('Korinthos',k2,k2*0.2)
>>> k3=katastimata('Kalamata',k3,k3*0.2)
```

```
>>> print(k1.p,': proionta:',k1.pr,'ypiresies:',k1.y)
('Patra', ': proionta:', 866100, 'ypiresies:', 173220.0)
>>> print(k2.p,': proionta:',k2.pr,'ypiresies:',k2.y)
('Korinthos', ': proionta:', 684020, 'ypiresies:', 136804.0)
>>> print(k3.p,': proionta:',k3.pr,'ypiresies:',k3.y)
('Kalamata', ': proionta:', 486600, 'ypiresies:', 97320.0)
>>> s1=k1.pr+k2.pr+k3.pr
>>> s2=k1.y+k2.y+k3.y
>>> s=s1+s2
>>> print'O tziros ton poliseon anilthe sta', s1,'euro eno gia tis ypiresies o tziros einai',s2,'euro. Eixame synoliko tziro',s,'euro.'
O tziros ton poliseon anilthe sta 2036720 euro, eno gia tis ypiresies o tziros einai 407344.0 euro. Eixame synoliko tziro 2444064.0 euro.
>>> s3=s1*(1-0.4)
>>> s4=s2*(1-0.22)
>>> s5=s3+s4
>>> print'Oi anamenomenos tziros gia tin epomeni periodo einai',s5,'euro, diladi:',s3,'gia ta proionta kai',s4,'gia tis ypiresies.'
Oi anamenomenos tziros gia tin epomeni periodo einai 1539760.32 euro, diladi: 1222032.0 gia ta proionta kai 317728.32 gia tis ypiresies.
```



Εικόνα 128 – Αντικειμενοστραφής προγραμματισμός: λύση της Α άσκησης 1.

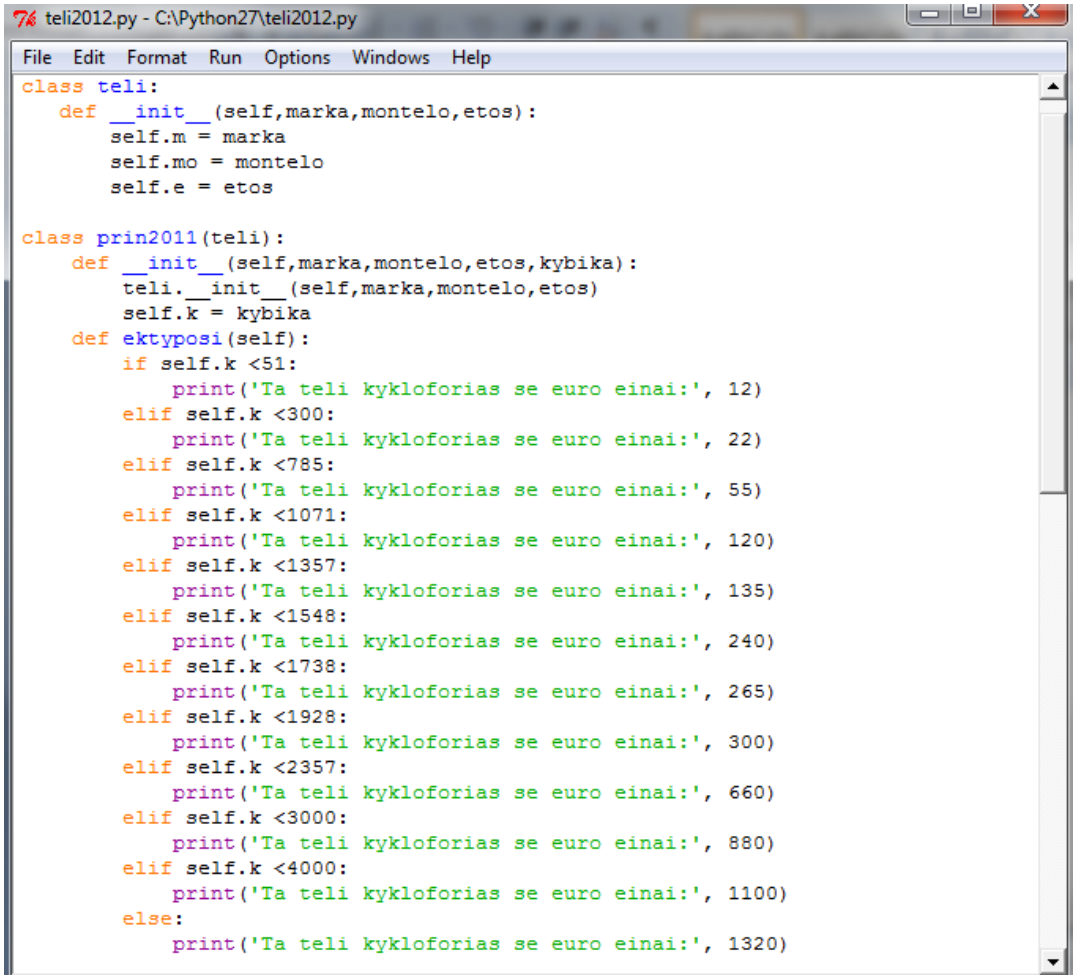
```
>>> s3=s1*(1-0.4)
>>> s4=s2*(1-0.22)
>>> s5=s3+s4
>>> print'Oi anamenomenos tziros gia tin epomeni periodo einai',s5,'euro, diladi:',s3,'gia ta proionta kai',s4,'gia tis ypiresies.'
Oi anamenomenos tziros gia tin epomeni periodo einai 1539760.32 euro, diladi: 1222032.0 gia ta proionta kai 317728.32 gia tis ypiresies.
>>>
```

Εικόνα 129 - Αντικειμενοστραφής προγραμματισμός: λύση της Α άσκησης 2.

Β. Λύση

```
class teli:
  def __init__(self,marka,montelo,etos):
     self.m = marka
     self.mo = montelo
     self.e = etos

class prin2011(teli):
   def __init__(self,marka,montelo,etos,kybika):
      teli.__init__(self,marka,montelo,etos)
      self.k = kybika
   def ektyposi(self):
      if self.k <51:
         print('Ta teli kykloforias se euro einai:', 12)
      elif self.k <300:
         print('Ta teli kykloforias se euro einai:', 22)
      elif self.k <785:
         print('Ta teli kykloforias se euro einai:', 55)
      elif self.k <1071:
         print('Ta teli kykloforias se euro einai:', 120)
      elif self.k <1357:
         print('Ta teli kykloforias se euro einai:', 135)
      elif self.k <1548:
         print('Ta teli kykloforias se euro einai:', 240)
      elif self.k <1738:
         print('Ta teli kykloforias se euro einai:', 265)
      elif self.k <1928:
         print('Ta teli kykloforias se euro einai:', 300)
      elif self.k <2357:
         print('Ta teli kykloforias se euro einai:', 660)
      elif self.k <3000:
         print('Ta teli kykloforias se euro einai:', 880)
      elif self.k <4000:
         print('Ta teli kykloforias se euro einai:', 1100)
      else:
         print('Ta teli kykloforias se euro einai:', 1320)
```



Εικόνα 130 - Αντικειμενοστραφής προγραμματισμός: λύση της Β άσκησης 1.

```
class meta2011(teli):
    def __init__(self,marka,montelo,etos,co2):
        teli.__init__(self,marka,montelo,etos)
        self.r = co2
    def ektyposi(self):
        if self.r >250:
            print('Ta teli kykloforias se euro einai:', self.r*3.4)
        elif self.r >200:
            print('Ta teli kykloforias se euro einai:', self.r*2.8)
        elif self.r >180:
            print('Ta teli kykloforias se euro einai:', self.r*2.55)
        elif self.r >160:
            print('Ta teli kykloforias se euro einai:', self.r*2.25)
        elif self.r >140:
            print('Ta teli kykloforias se euro einai:', self.r*1.7)
        elif self.r >120:
            print('Ta teli kykloforias se euro einai:', self.r*1.1)
        elif self.r >100:
            print('Ta teli kykloforias se euro einai:', self.r*0.9)
        else:
            print('Den plironeis teli kykloforias')
```

```
print('Υπολογισμός τελών κυκλοφορίας IX 2012')
print('Για την καταχώρηση των στοιχείων θα χωρίσετε τα πεδία με κόμμα: μάρκα, μοντέλο, έτος κυκλοφορίας, κυβικά/ρύπους')
print('Αν το όχημα έχει κυκλοφορήσει μετά τις 1/11/2010, να δηλώσετε 2010,1 αντί για 2010 όταν ερωτηθείτε.')
print(' Καταχωρείστε τους ρύπους αντί των κυβικών αν το όχημα σας είναι μετά τις 1/11/2010')
etos = 1
while etos>0:
   etos= input('Δώσε έτος κυκλοφορίας αυτοκινήτου (0 για έξοδο):')
   if etos ==0:
      continue
   a = input('Δώσε στοιχεία αυτοκινήτου :')
   print a
    if etos>2010:
       c = meta2011(a[0],a[1],a[2],a[3])
       c.ektyposi()
   else:
       c = prin2011(a[0],a[1],a[2],a[3])
       c.ektyposi()
print'Ευχαριστούμε πολύ!! Καλή σας ημέρα!'
```

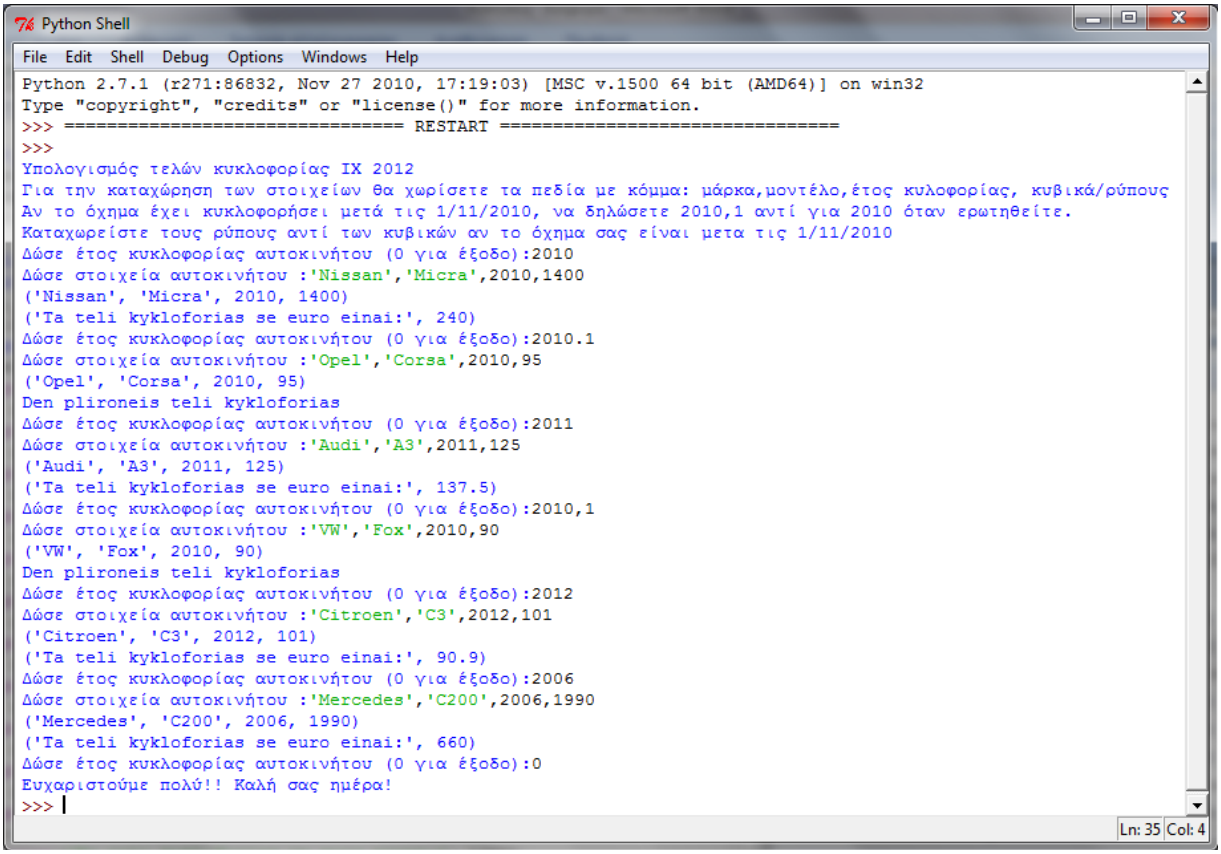
```
class meta2011(teli):
    def __init__(self,marka,montelo,etos,co2):
        teli.__init__(self,marka,montelo,etos)
        self.r = co2
    def ektyposi(self):
        if self.r >250:
            print('Ta teli kykloforias se euro einai:', self.r*3.4)
        elif self.r >200:
            print('Ta teli kykloforias se euro einai:', self.r*2.8)
        elif self.r >180:
            print('Ta teli kykloforias se euro einai:', self.r*2.55)
        elif self.r >160:
            print('Ta teli kykloforias se euro einai:', self.r*2.25)
        elif self.r >140:
            print('Ta teli kykloforias se euro einai:', self.r*1.7)
        elif self.r >120:
            print('Ta teli kykloforias se euro einai:', self.r*1.1)
        elif self.r >100:
            print('Ta teli kykloforias se euro einai:', self.r*0.9)
        else:
            print('Den plironeis teli kykloforias')

print('Υπολογισμός τελών κυκλοφορίας IX 2012')
print('Για την καταχώρηση των στοιχείων θα χωρίσετε τα πεδία με κόμμα: μάρκα,μοντέλο,έτος κυκλοφορίας, κυβικά/ρύπους')
print('Αν το όχημα έχει κυκλοφορήσει μετά τις 1/11/2010, να δηλώσετε 2010,1 αντί για 2010 όταν ερωτηθείτε.')
print('Καταχωρείστε τους ρύπους αντί των κυβικών αν το όχημα σας είναι μετά τις 1/11/2010')
etos = 1
while etos>0:
    etos= input('Δώσε έτος κυκλοφορίας αυτοκινήτου (0 για έξοδο):')
    if etos ==0:
        continue
    a = input('Δώσε στοιχεία αυτοκινήτου :')
    print a
    if etos>2010:
         c = meta2011(a[0],a[1],a[2],a[3])
         c.ektyposi()
    else:
         c = prin2011(a[0],a[1],a[2],a[3])
         c.ektyposi()
print'Ευχαριστούμε πολύ!! Καλή σας ημέρα!'
```

Εικόνα 131 - Αντικειμενοστραφής προγραμματισμός: λύση της Β άσκησης 2.

Υπολογισμός τελών κυκλοφορίας IX 2012
Για την καταχώρηση των στοιχείων θα χωρίσετε τα πεδία με κόμμα: μάρκα, μοντέλο, έτος κυκλοφορίας, κυβικά/ρύπους

Αν το όχημα έχει κυκλοφορήσει μετά τις 1/11/2010, να δηλώσετε 2010,1 αντί για 2010 όταν ερωτηθείτε.
Καταχωρείστε τους ρύπους αντί των κυβικών αν το όχημα σας είναι μετά τις 1/11/2010
Δώσε έτος κυκλοφορίας αυτοκινήτου (0 για έξοδο):2010
Δώσε στοιχεία αυτοκινήτου :'Nissan','Micra',2010,1400
('Nissan', 'Micra', 2010, 1400)
('Ta teli kykloforias se euro einai:', 240)
Δώσε έτος κυκλοφορίας αυτοκινήτου (0 για έξοδο):2010.1
Δώσε στοιχεία αυτοκινήτου :'Opel','Corsa',2010,95
('Opel', 'Corsa', 2010, 95)
Den plironeis teli kykloforias
Δώσε έτος κυκλοφορίας αυτοκινήτου (0 για έξοδο):2011
Δώσε στοιχεία αυτοκινήτου :'Audi','A3',2011,125
('Audi', 'A3', 2011, 125)
('Ta teli kykloforias se euro einai:', 137.5)
Δώσε έτος κυκλοφορίας αυτοκινήτου (0 για έξοδο):2010,1
Δώσε στοιχεία αυτοκινήτου :'VW','Fox',2010,90
('VW', 'Fox', 2010, 90)
Den plironeis teli kykloforias
Δώσε έτος κυκλοφορίας αυτοκινήτου (0 για έξοδο):2012
Δώσε στοιχεία αυτοκινήτου :'Citroen','C3',2012,101
('Citroen', 'C3', 2012, 101)
('Ta teli kykloforias se euro einai:', 90.9)
Δώσε έτος κυκλοφορίας αυτοκινήτου (0 για έξοδο):2006
Δώσε στοιχεία αυτοκινήτου :'Mercedes','C200',2006,1990
('Mercedes', 'C200', 2006, 1990)
('Ta teli kykloforias se euro einai:', 660)
Δώσε έτος κυκλοφορίας αυτοκινήτου (0 για έξοδο):0
Ευχαριστούμε πολύ!! Καλή σας ημέρα!

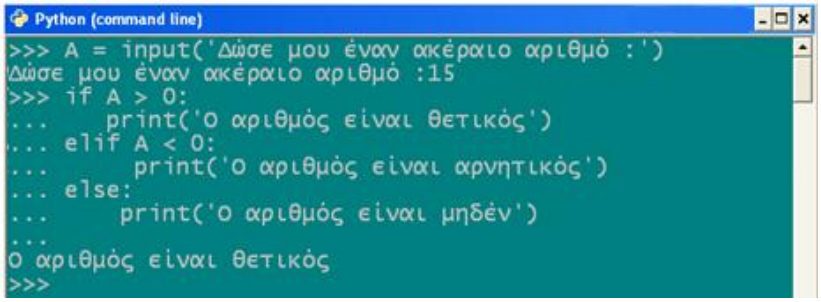


Εικόνα 132 – Τρέξιμο του προγράμματος τελών κυκλοφορίας.

IV. Ασκήσεις στον έλεγχο ροής.

A. Λύση

A = input(‘Δώσε μου έναν ακέραιο αριθμό :΄)
Δώσε μου έναν ακέραιο αριθμό :15
>>> if A > 0:
… Print(΄Ο αριθμός είναι θετικός)
… Elif A < 0:
… Print(΄Ο αριθμός είναι αρνητικός)
… Else:
… Print(΄Ο αριθμός είναι μηδέν)
Ο αριθμός είναι θετικός

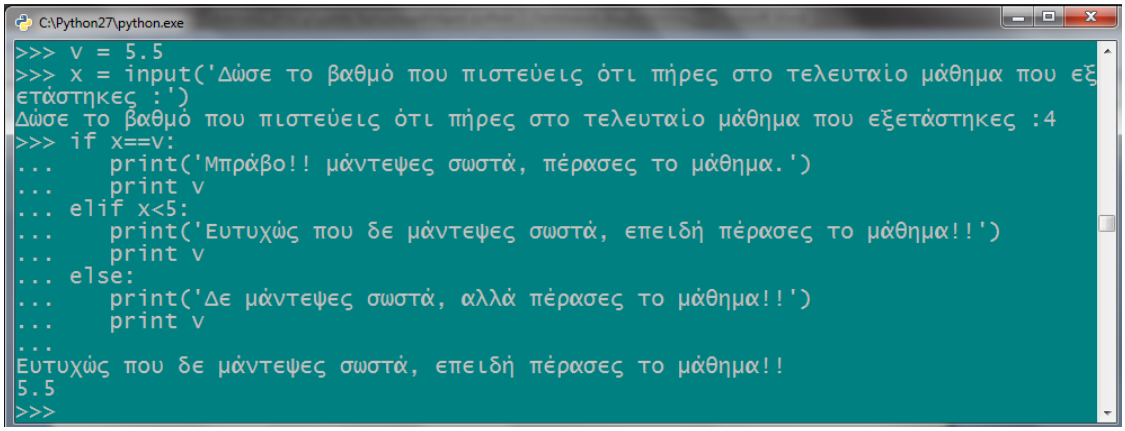


Εικόνα 133 – Έλεγχος ροής: λύση της Α άσκησης.

B. Λύση

>>> v = 5.5

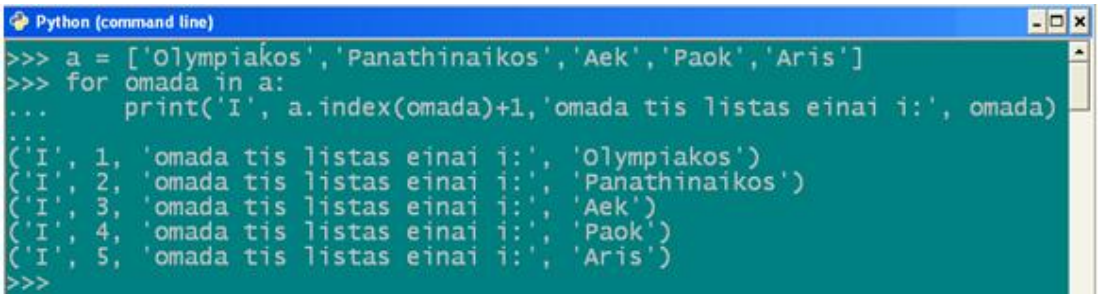
```
>>> x = input('Δώσε το βαθμό που πιστεύεις ότι πήρες στο τελευταίο μάθημα που εξετάστηκες :')
Δώσε το βαθμό που πιστεύεις ότι πήρες στο τελευταίο μάθημα που εξετάστηκες :4
>>> if x == v:
...     print('Μπράβο!! μάντεψες σωστά, πέρασες το μάθημα.')
...     print v
... elif x<5:
...     print('Ευτυχώς που δε μάντεψες σωστά, επειδή πέρασες το μάθημα!!')
...     print v
... else:
...     print('Δε μάντεψες σωστά, αλλά πέρασες το μάθημα!!')
...     print v
...
Ευτυχώς που δε μάντεψες σωστά, επειδή πέρασες το μάθημα!!
5.5
```



Εικόνα 134 - Έλεγχος ροής: λύση της Β άσκησης.

C. Λύση

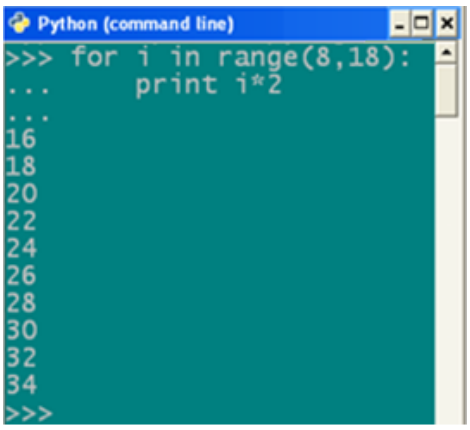
```
>>> a = [´Olympiakos´,´Panathinaikos´,´Aek´,´Paok´,´Aris´]
>>> for omada in a:
…         print(´I´, a.index(omada)+1, ´omada tis listas einai i:´, omada)
…
(´I´ , 1, ´omada tis listas einai i:´,  ´Olympiakos´)
(´I´ , 2, ´omada tis listas einai i:´,  ´Panathinaikos´)
(´I´ , 3, ´omada tis listas einai i:´,  ´Aek´)
(´I´ , 4, ´omada tis listas einai i:´,  ´Paok´)
(´I´ , 5, ´omada tis listas einai i:´,  ´Aris´)
```



Εικόνα 135 - Έλεγχος ροής: λύση της C άσκησης.

D. Λύση

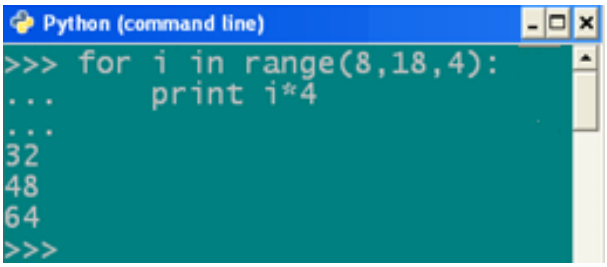
```
>>> for I in range(8,18):
…         Print i*2
…
16
18
20
22
24
26
28
30
32
34
```



Εικόνα 136 - Έλεγχος ροής: λύση της D άσκησης.

E. Λύση

```
>>> for I in range(8,18,4):
…         print i*4
…
32
48
64
```

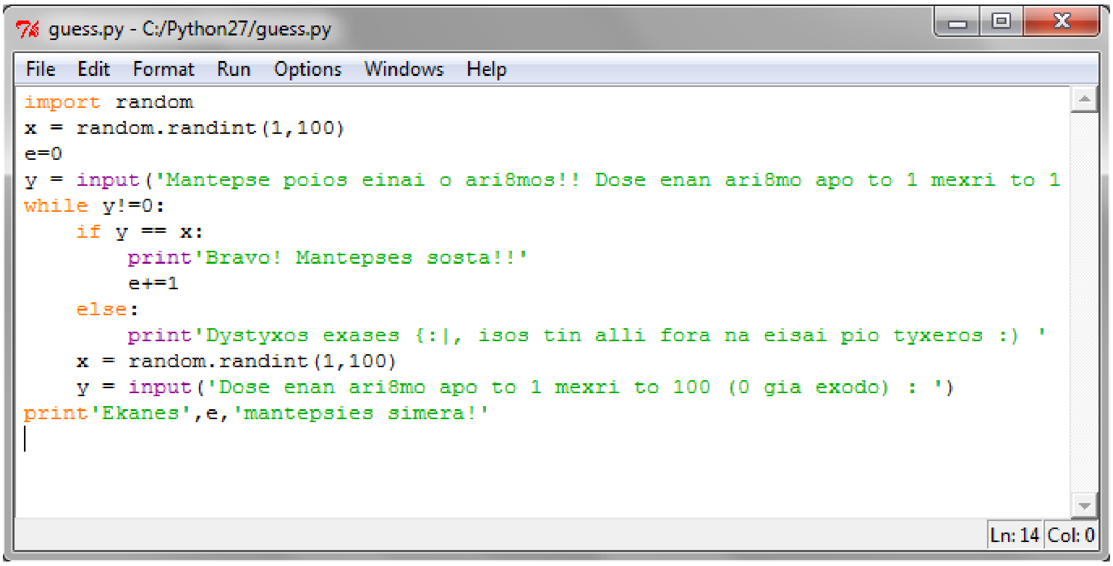


Εικόνα 137 - Έλεγχος ροής: λύση της E άσκησης.

F. Λύση

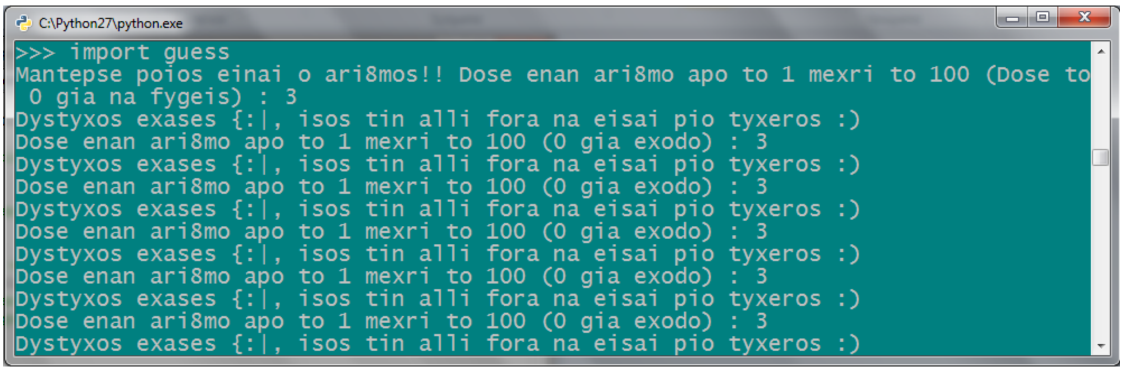
```
import random
x = random.randint(1,100)
e=0
```

```
y = input('Mantepse poios einai o ari8mos!! Dose enan ari8mo apo to 1 mexri to 100 (Dose to 0 gia na fygeis) : ')
while y!=0:
    if y == x:
        print'Bravo! Mantepses sosta!!'
        e+=1
    else:
        print'Dystyxos exases {:|, isos tin alli fora na eisai pio tyxeros :) '
    x = random.randint(1,100)
    y = input('Dose enan ari8mo apo to 1 mexri to 100 (0 gia exodo) : ')
print'Ekanes',e,'mantepsies simera!'
```



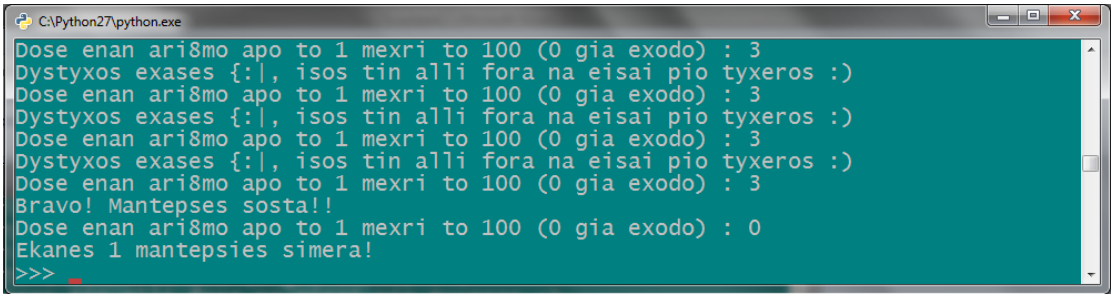
Εικόνα 138 - Έλεγχος ροής: λύση της F άσκησης 1.

Εισάγεται το module και γίνεται μια δοκιμή.



Εικόνα 139 - Έλεγχος ροής: λύση της F άσκησης 2.

Ύστερα από 28 προσπάθειες, έγινε μια σωστή πρόβλεψη.

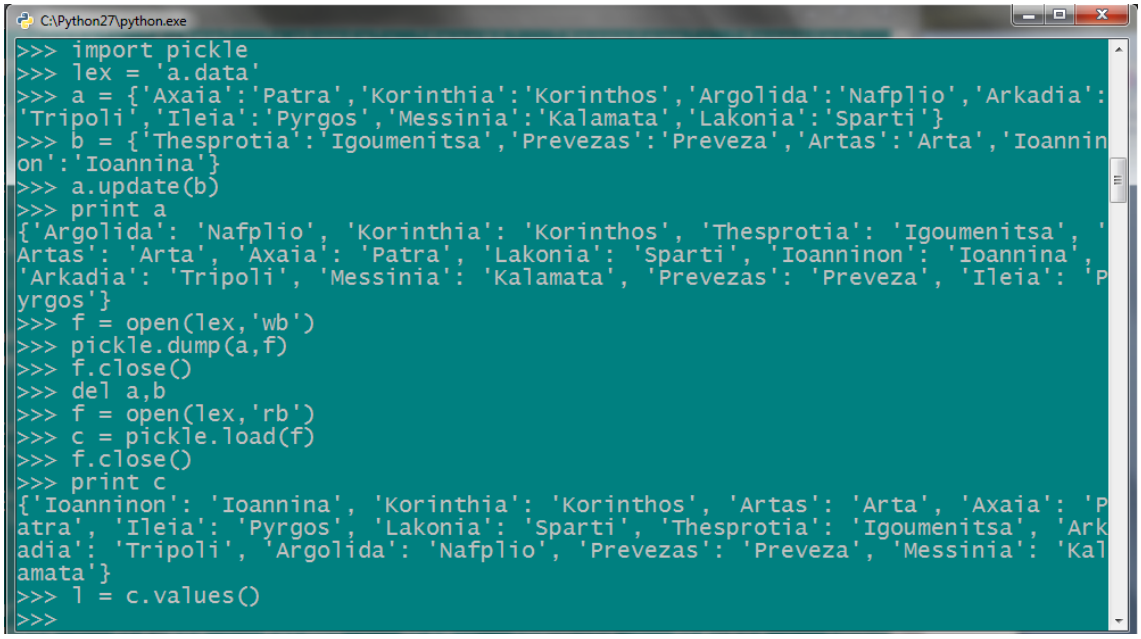


Εικόνα 140 - Έλεγχος ροής: λύση της F άσκησης 3.

V. Άσκηση στα αρχεία.

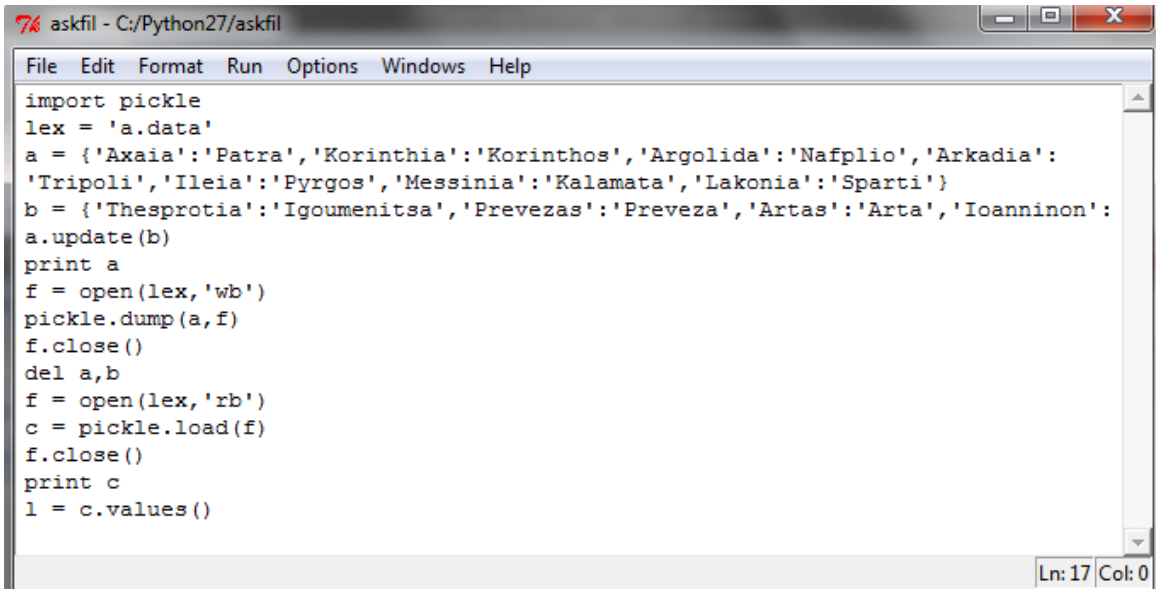
Λύση

```
>>> import pickle
>>> lex = 'a.data'
>>> a = {'Axaia':'Patra','Korinthia':'Korinthos','Argolida':'Nafplio','Arkadia':'Tripoli','Ileia':'Pyrgos','Messinia':'Kalamata','Lakonia':'Sparti'}
>>>b={'Thesprotia':'Igoumenitsa','Prevezas':'Preveza','Artas':'Arta','Ioanninon':'Ioannina'}
>>> a.update(b)
>>> print a
{'Argolida': 'Nafplio', 'Korinthia': 'Korinthos', 'Thesprotia': 'Igoumenitsa', 'Artas': 'Arta', 'Axaia': 'Patra', 'Lakonia': 'Sparti', 'Ioanninon': 'Ioannina', 'Arkadia': 'Tripoli', 'Messinia': 'Kalamata', 'Prevezas': 'Preveza', 'Ileia': 'Pyrgos'}
>>> f = open(lex,'wb')
>>> pickle.dump(a,f)
>>> f.close()
>>> del a,b
>>> f = open(lex,'rb')
>>> c = pickle.load(f)
>>> f.close()
>>> print c
{'Ioanninon': 'Ioannina', 'Korinthia': 'Korinthos', 'Artas': 'Arta', 'Axaia': 'Patra', 'Ileia': 'Pyrgos', 'Lakonia': 'Sparti', 'Thesprotia': 'Igoumenitsa', 'Arkadia': 'Tripoli', 'Argolida': 'Nafplio', 'Prevezas': 'Preveza', 'Messinia': 'Kalamata'}
>>> l = c.values()
```



Εικόνα 141 – Αρχεία: εισαγωγή του pickle και δημιουργία λεξικών.

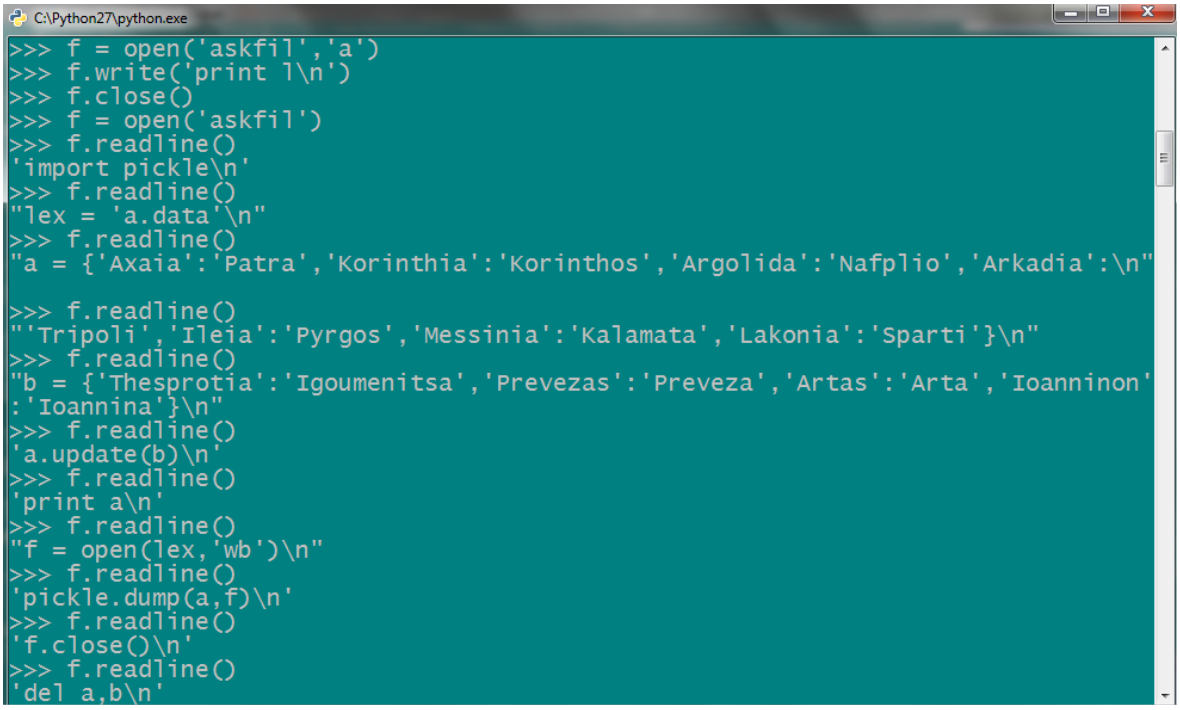
Στη συνέχεια αποθηκεύονται οι παραπάνω εντολές, σε ένα αρχείο κειμένου:



Εικόνα 142 – Αρχεία: δημιουργία και αποθήκευση του αρχείου.

Έχει αποθηκευθεί το αρχείο ως askfil. Τώρα, θα ανοίξει μόνο για εγγραφή και θα εισαχθεί η εντολή που ζητείται στην εκφώνηση. Αφού προστεθεί η εντολή εμφάνισης της λίστας, θα ανοίξει πάλι το αρχείο, για ανάγνωση αυτή τη φορά και θα διαβαστούν όλες οι σειρές μία προς μία.

```
>>> f = open('askfil','a')
>>> f.write('print l\n')
>>> f.close()
```



Εικόνα 143 – Αρχεία: λύση της άσκησης (..συνέχεια).

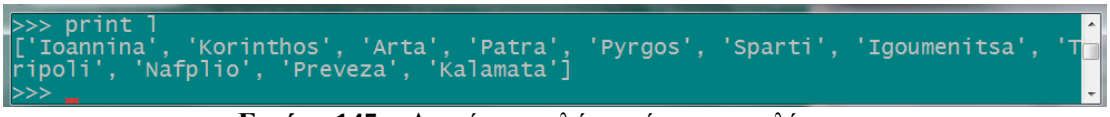
```
>>> f = open('askfil')
>>> f.readline()
'import pickle\n'
>>> f.readline()
"lex = 'a.data'\n"
>>> f.readline()
"a = {'Axaia':'Patra','Korinthia':'Korinthos','Argolida':'Nafplio','Arkadia':\n"
>>> f.readline()
"'Tripoli','Ileia':'Pyrgos','Messinia':'Kalamata','Lakonia':'Sparti'}\n"
>>> f.readline()
"b = {'Thesprotia':'Igoumenitsa','Prevezas':'Preveza','Artas':'Arta','Ioanninon'
:'Ioannina'}\n"
>>> f.readline()
'a.update(b)\n'
>>> f.readline()
'print a\n'
>>> f.readline()
"f = open(lex,'wb')\n"
>>> f.readline()
'pickle.dump(a,f)\n'
>>> f.readline()
'f.close()\n'
>>> f.readline()
'del a,b\n'
>>> f.readline()
"f = open(lex,'rb')\n"
>>> f.readline()
'c = pickle.load(f)\n'
>>> f.readline()
```

'f.close()\n'
>>> f.readline()
'print c\n'
>>> f.readline()
'l = c.values()\n'
>>> f.readline()
'print l\n'
>>> f.readline()
''

Εικόνα 144 – Αρχεία: ανάγνωση του αρχείου (..συνέχεια).

Με την εκτέλεση της εντολής εκτύπωσης, θα εμφανιστεί η παρακάτω λίστα:

>>> print l
['Ioannina', 'Korinthos', 'Arta', 'Patra', 'Pyrgos', 'Sparti', 'Igoumenitsa', 'Tripoli', 'Nafplio', 'Preveza', 'Kalamata']

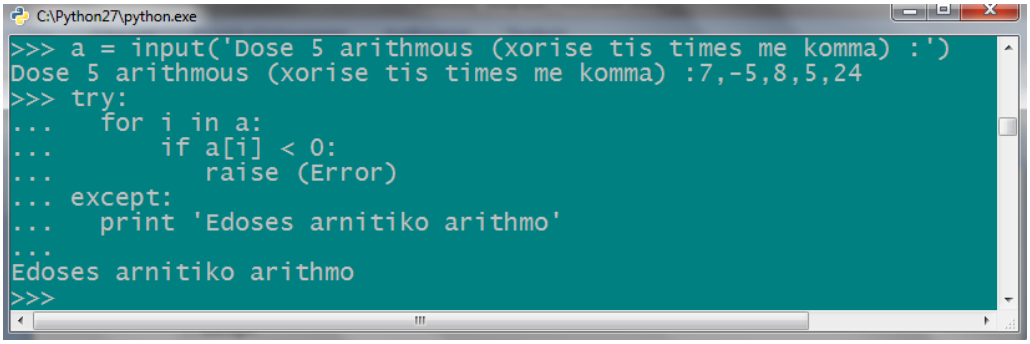


Εικόνα 145 – Αρχεία: εντολή εμφάνισης της λίστας.

VI. Ασκήσεις στις εξαιρέσεις.

A. Λύση

```
>>> a = input('Dose 5 arithmous (xorise tis times me komma) :')
Dose 5 arithmous (xorise tis times me komma) :7,-5,8,5,24
>>> try:
...   for i in a:
...       if a[i] < 0:
...           raise (Error)
... except:
...   print 'Edoses arnitiko arithmo'
...
Edoses arnitiko arithmo
```



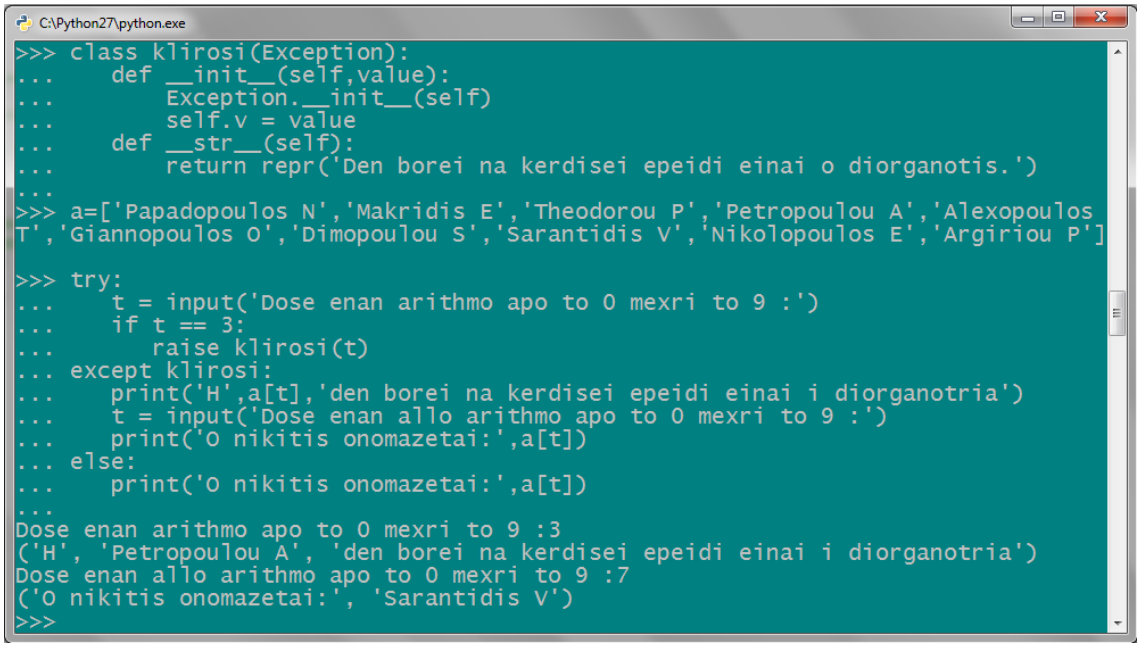
Εικόνα 146 - Εξαιρέσεις: λύση της Α άσκησης.

B. Λύση

Θα δημιουργηθεί η εξαίρεση και θα εισαχθεί μια λίστα με ονόματα.

```
class klirosi(Exception):
  def __init__(self,value):
     Exception.__init__(self)
     self.v = value
  def __str__(self):
     return repr('Den borei na kerdisei epeidi einai o diorganotis.')

a=['Papadopoulos N','Makridis E','Theodorou P','Petropoulou A','Alexopoulos T','Giannopoulos O','Dimopoulou S','Sarantidis V','Nikolopoulos E','Argiriou P']
try:
  t = input('Dose enan arithmo apo to 0 mexri to 9 :')
  if t == 3:
    raise klirosi(t)
except klirosi:
  print('H',a[t],'den borei na kerdisei epeidi einai i diorganotria')
  t = input('Dose enan allo arithmo apo to 0 mexri to 9 :')
  print('O nikitis onomazetai:',a[t])
else:
  print('O nikitis onomazetai:',a[t])
Dose enan arithmo apo to 0 mexri to 9 :3
('H', 'Petropoulou A', 'den borei na kerdisei epeidi einai i diorganotria')
Dose enan allo arithmo apo to 0 mexri to 9 :7
('O nikitis onomazetai:', 'Sarantidis V')
```

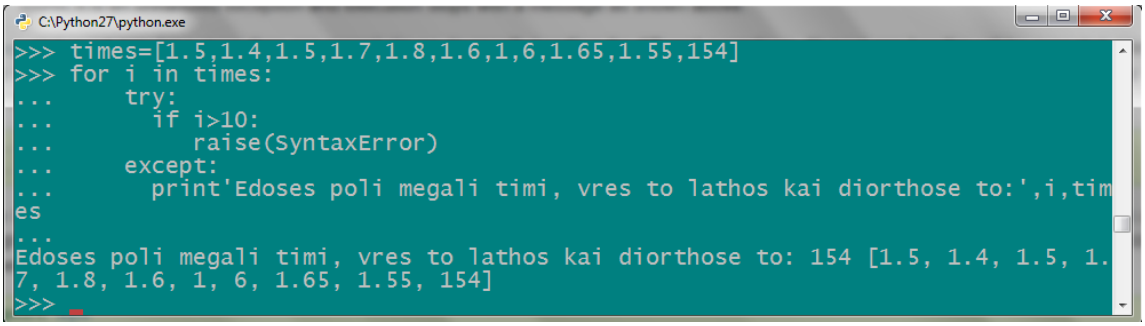


Εικόνα 147 - Εξαιρέσεις: λύση της Β άσκησης.

C. Λύση

```
>>> times=[1.5,1.4,1.5,1.7,1.8,1.6,1,6,1.65,1.55,154]
>>> for i in times:
...     try:
...       if i>10:
...          raise(SyntaxError)
...     except:
...       print'Edoses poli megali timi, vres to lathos kai diorthose to',i,times
...
```

Edoses poli megali timi, vres to lathos kai diorthose to: 154 [1.5, 1.4, 1.5, 1.7, 1.8, 1.6, 1, 6, 1.65, 1.55, 154]



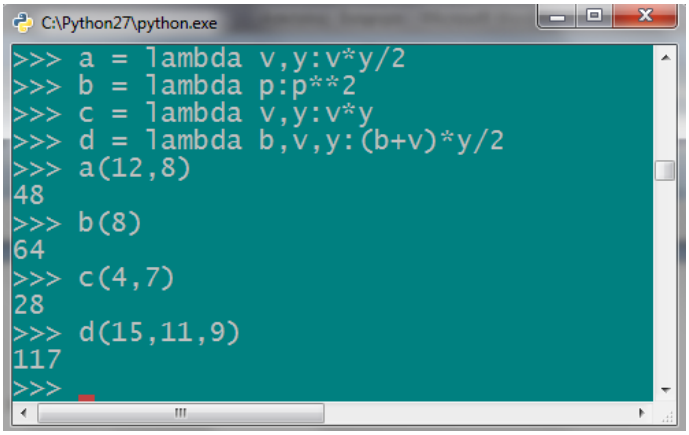
Εικόνα 148 - Εξαιρέσεις: λύση της C άσκησης.

VII. Άσκηση στις εκφράσεις λάμδα.

Λύση

```
>>> a = lambda v,y:v*y/2
>>> b = lambda p:p**2
>>> c = lambda v,y:v*y
>>> d = lambda b,v,y:(b+v)*y/2
>>> a(12,8)
```

48
>>> b(8)
64
>>> c(4,7)
28
>>> d(15,11,9)
117



Εικόνα 149 - Λύση της άσκησης στις εκφράσεις λ.