ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Python Programming

Αντωνία Τερζίδου

&

Μηνάς Δασυγένης

http://arch.icte.uowm.gr

Κοζάνη,

Ιούνιος 2016

Περιεχόμενα

1. Εισαγωγή……4
2. Ιστορικά στοιχεία……..5
3. Χαρακτηριστικά της Python……...8
4. Πως εγκαθιστώ την Python στον υπολογιστή μου….16
5. Το περιβάλλον της Python 3.3.2……..17
6. Βασικές εισαγωγικές έννοιες της Python…….27
7. Τελεστές και εκφράσεις……38
8. Έλεγχος ροής……...54
9. Χρήση λεξικού……..70
10. Συναρτήσεις……..78
11. Αρθρώματα……100
12. Αλφαριθμητικά……..108
13. Δομές δεδομένων……111
14. Αντικείμενα και κλάσεις……..139
15. Αρχεία……..157
16. Είσοδος- έξοδος……160
17. Εγγραφή αντικειμένων σε αρχεία-σειριοποίηση…….163
18. Φάκελοι……165
19. Εξαιρέσεις……168
20. Γεννήτορες……..177
21. Κανονικές εκφράσεις……189
22. GUI με tkinter……..191
23. Python Tkinter Relief Styles……202
24. Python Tkinter Bitmaps……203
25. Python Tkinter Cursors……204
26. «Χτίσιμο» μιας βασικής GUI εφαρμογής (GUI application) …207
27. Drawing in tkinter……219
28. Multithreading……224
29. Μέτρηση χρόνου εκτέλεσης……226
30. Profilers……..231
31. Αποσφαλμάτωση……237
32. GCI Programming……240
33. Biopython……..244
34. Βιβλιογραφία……253

Υπάρχουν δύο τρόποι για να κατασκευάσετε το σχεδιασμό ενός λογισμικού: ο ένας είναι να το κάνετε τόσο απλό ώστε προφανώς να μην υπάρχουν ελαττώματα, και ο άλλος είναι να το κάνετε τόσο πολύπλοκο ώστε να μην είναι προφανή τα ελαττώματα.

C. A. R. Hoare

- ✓ **Η επιτυχία στη ζωή δεν είναι τόσο θέμα ταλέντου και ευκαιριών όσο συγκέντρωσης και συνέπειας.**

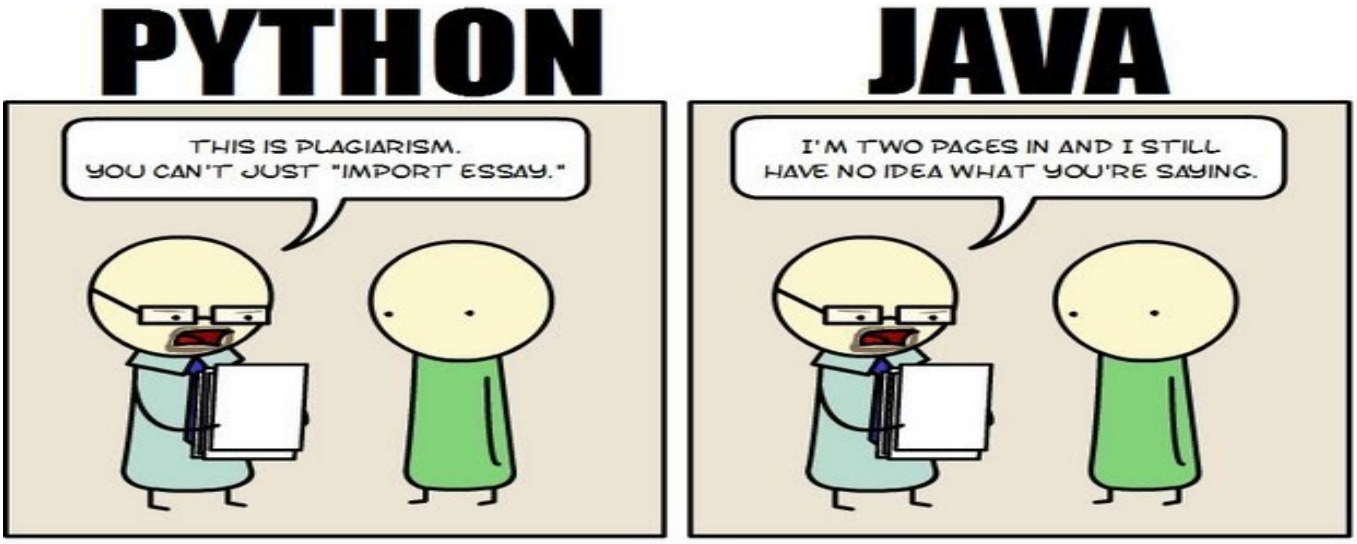
C. W. Wendte

Μία από τις αρχές καθοδήγησης της Python είναι ότι *" Οι ρητές εντολές είναι καλύτερες από αυτές που εξυπακούονται" (Explicit is better than Implicit).*

Εισαγωγή

Η Python είναι πιθανότατα μια από τις ελάχιστες γλώσσες προγραμματισμού που είναι τόσο εύκολες και ταυτόχρονα τόσο ισχυρές. Κατά την διάρκεια της ενασχολησής μου με το παρόν project,διαπίστωσα ότι **είναι διασκεδαστικό να προγραμματίζεις με την Python.**

Γλώσσες προγραμματισμού όπως η Java, η VB, η C++, κλπ. αποτελούν το βαρύ πυροβολικό στον κόσμο του προγραμματισμού. Συχνά όμως, οι προγραμματιστές θέλουμε να γράψουμε μια μικρή εφαρμογή, που να μας καλύπτει κάποια συγκεκριμένη ανάγκη, στα γρήγορα.



Η Python είναι μια απλή αλλά παράλληλα «δυνατή» γλώσσα προγραμματισμού, που θα μας επιτρέψει να αναπτύσσουμε τις μικροεφαρμογές μας εύκολα και γρήγορα.

Η Python είναι μια δυναμική γλώσσα προγραμματισμού, πολύ υψηλού επιπέδου. Κύριος στόχος της είναι η αυξημένη παραγωγικότητα του προγραμματιστή και η αναγνωσιμότητα του κώδικα. Έχει ενσωματωμένες δομές δεδομένων όπως λίστες και πίνακες κατακερματισμού (που αποκαλεί dictionaries) και μια αρκετά μεγάλη βιβλιοθήκη, ακολουθώντας τη φιλοσοφία batteries included.

Δανείζεται στοιχεία από τον προστακτικό, αντικειμενοστρεφή αλλά και συναρτησιακό προγραμματισμό, χωρίς να επιβάλλει κάποιο συγκεκριμένο τρόπο προγραμματισμού (σε αντίθεση π.χ. με τη Java).

Τα χαρακτηριστικά της την καθιστούν καλή επιλογή για prototyping, αλλά και για πληθώρα εφαρμογών στις οποίες η απόδοση δεν είναι τόσο σημαντικός παράγοντας.

Η Python είναι εύκολη στην εκμάθηση και ταυτόχρονα ισχυρή γλώσσα προγραμματισμού. Έχει αποδοτικές δομές δεδομένων υψηλού επιπέδου και μια απλή αλλά αποτελεσματική προσέγγιση στον αντικειμενοστρεφή προγραμματισμό.

Η κομψή σύνταξη της Python και οι δυναμικοί τύποι της, μαζί με τη λειτουργία της ως διερμηνευόμενης (αντί μεταγλωττιζόμενης) γλώσσας, την καθιστούν την ιδανική

γλώσσα για δημιουργία σεναρίων εντολών και για ταχεία ανάπτυξη εφαρμογών σε πολλούς τομείς και στις περισσότερες πλατφόρμες.

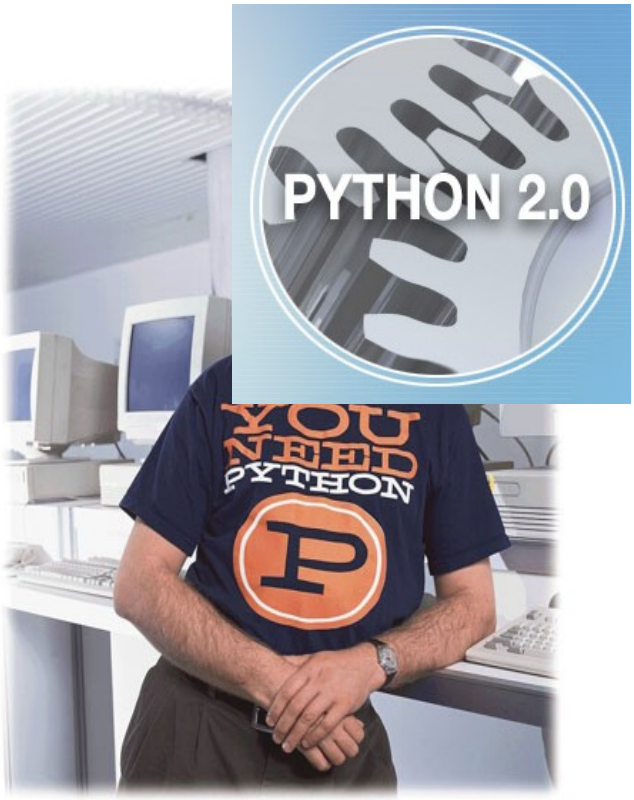
Η Python είναι μια πολύ όμορφη, ιδιαίτερα εκφραστική, αντικειμενοστραφής, υψηλού επιπέδου γλώσσα προγραμματισμού. Το σύστημα τύπων της είναι πλήρως δυναμικό και η διαχείριση μνήμης αυτόματη, η standard υλοποίησή της (CPython) είναι ένας bytecode compiler και interpreter και η βιβλιοθήκη της πολύ πλούσια.

Βασική αρχή της Python είναι ότι ο χρόνος του προγραμματιστή είναι πολύ σημαντικότερος από τους κύκλους μηχανής και τον απαλλάσσει από τις χαμηλού επιπέδου λεπτομέρειες που πρέπει να φροντίσει στη C/C++ ή, ως ένα βαθμό, στη Java.

Ιστορικά στοιχεία

Η Python είναι μια διερμηνευόμενη (interpreted), αλληλεπιδραστική (interactive) και προσανατολισμένη σε αντικείμενα (object-oriented) γλώσσα προγραμματισμού, που αναπτύχθηκε από τον Ολλανδό *Guido van Rossum, το 1990*. Το όνομά της προέρχεται από ένα από τα αγαπημένα τηλεοπτικά θεάματα του van Rossum, το ιπτάμενο τσίρκο των Monty Python's.

Η Python είναι μια γλώσσα που συνδυάζει σημαντική ισχύ με πολύ σαφή σύνταξη. Χρησιμοποιεί ενότητες (modules), τάξεις (classes), εξαιρέσεις (exceptions) καθώς και πολύ υψηλού επιπέδου δυναμικούς τύπους δεδομένων.



Οι διερμηνευτές (interpreters) της Python είναι διαθέσιμοι για τα περισσότερα λειτουργικά συστήματα και ο πηγαίος κώδικας (source code) της Python διατίθεται δωρεάν. Η επίσημη ιστοσελίδα της γλώσσας είναι η **http://www.python.org**. Η ανάπτυξη της Python ξεκίνησε το 1990 στο CWI του Άμστερνταμ και συνεχίζεται στο CNRI του Reston. Η γλώσσα διαθέτει μια κομψή αν και όχι πολύ απλοποιημένη σύνταξη και έναν μικρό αριθμό από ισχυρούς και υψηλού επιπέδου τύπους δεδομένων.

Η Python μπορεί να επεκταθεί προσθέτοντάς της καινούργια modules που να είναι γραμμένα σε μια γλώσσα που μεταγλωττίζεται, όπως είναι η C ή η C++. Αυτά τα modules επέκτασης μπορούν να ορίσουν νέες συναρτήσεις (functions) και μεταβλητές (variables) καθώς επίσης και καινούργιους τύπους δεδομένων (object types). Η Python συγκρίνεται συχνά μ' άλλες διερμηνευόμενες γλώσσες προγραμματισμού, όπως είναι οι Java, JavaScript, Perl, Tcl και η Smalltalk, αλλά συγκρίσεις μπορούν να γίνουν και με τις C++, Common Lisp και Scheme.

Ο κύριος στόχος της είναι η αναγνωσιμότητα του κώδικά της και η ευκολία χρήσης της. Διακρίνεται λόγω του ότι έχει πολλές βιβλιοθήκες που διευκολύνουν ιδιαίτερα αρκετές συνηθισμένες εργασίες και για την ταχύτητα εκμάθησης της.

Η Python αναπτύσσεται ως ανοιχτό λογισμικό (open source) και η διαχείρισή της γίνεται από τον μη κερδοσκοπικό οργανισμό Python Software Foundation. Ο κώδικας διανέμεται με την άδεια Python Software Foundation License η οποία είναι συμβατή με την GPL. Το όνομα της γλώσσας προέρχεται από την ομάδα άγγλων κωμικών Μόντυ Πάιθον.

Αρχικά, η Python ήταν γλώσσα σεναρίων που χρησιμοποιούνταν στο λειτουργικό σύστημα Amoeba, ικανή και για κλήσεις συστήματος.

- ✓ Η **Python 2.0** κυκλοφόρησε στις 16 Οκτωβρίου του 2000. Στις 3 Δεκεμβρίου 2008 κυκλοφόρησε η έκδοση 3.0 (γνωστή και ως py3k ή python 3000). Πολλά από τα καινούργια χαρακτηριστικά αυτής της έκδοσης έχουν μεταφερθεί στις εκδόσεις 2.6 και 2.7 που είναι προς τα πίσω συμβατές.

- ✓ Η **python 3** είναι ιστορικά η πρώτη γλώσσα προγραμματισμού που σπάει την προς τα πίσω συμβατότητα με προηγούμενες εκδόσεις ώστε να διορθωθούν κάποια λάθη που υπήρχαν σε προγενέστερες εκδόσεις και να καταστεί ακόμα πιο σαφής ο απλός τρόπος με τον οποίο μπορούν να γίνουν κάποια πράγματα.

Γιατί Python;

→ Διότι αποτελεί scripting γλώσσα και γλώσσα προγραμματισμού. Αυτό σημαίνει ότι αν και είναι μία διερμηνευόμενη γλώσσα (δεν χρειάζεται να μεταγλωττίσετε τον κώδικά σας, απλά σώζεται και «τρέχετε» την εφαρμογή σας), διατηρεί τα περισσότερα πλεονεκτήματα μιας κλασικής γλώσσας προγραμματισμού (π.χ. Java, VB, κλπ.).

→ Είναι μια «απλή γλώσσα προγραμματισμού» ενώ συχνά την κατηγοριοποιούν στις very – high level languages. Οι δύο αυτοί χαρακτηρισμοί οφείλονται κυρίως στην απλή σύνταξη και τους «γενικούς» τύπους δεδομένων (που της επιτρέπουν μεγάλο πεδίο εφαρμογής).

→ Επιτρέπει τη δημιουργία modules που μπορούν να χρησιμοποιηθούν εύκολα από άλλες εφαρμογές γραμμένες σε Python.

→ Είναι προσανατολισμένη στην συγγραφή μικρότερων και συμπυκνωμένων εφαρμογών. Έτσι τα προγράμματα σε Python είναι αισθητά μικρότερα από τα αντίστοιχά τους σε γλώσσες όπως οι C/C++ ή Java.

→ Το όνομά της σχετίζεται με την show του BBC “Monty Python’s Flying Circus” και όχι με το γνωστό ερπετό.

Ο κώδικας σε γλώσσα προγραμματισμού Python μπορεί:

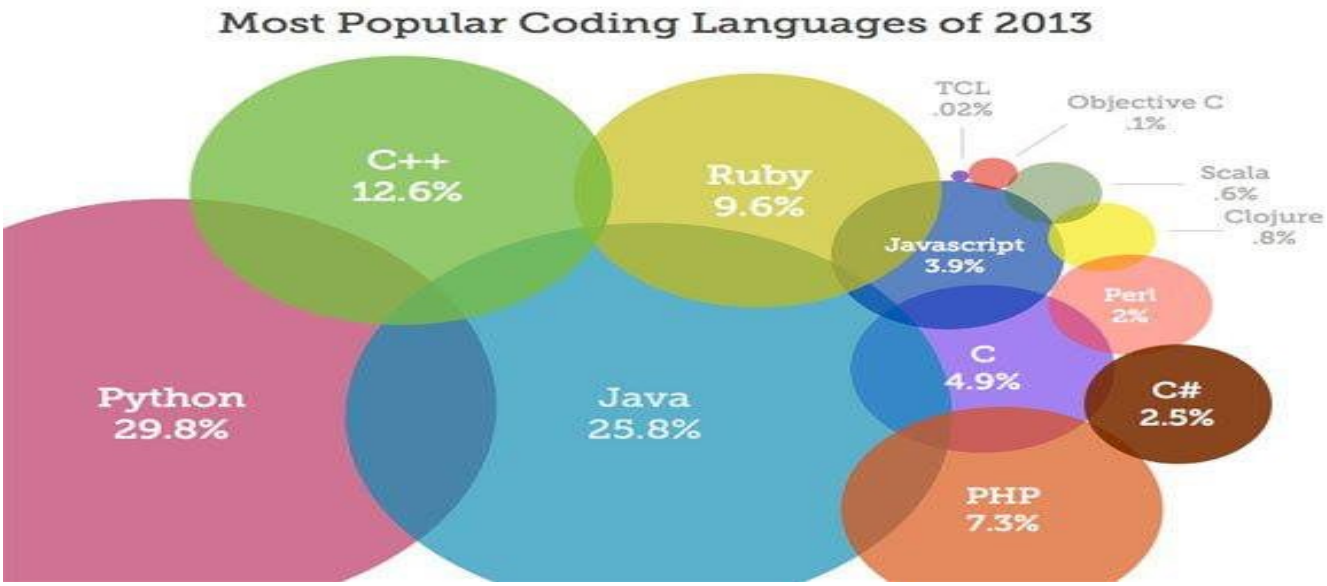
↔ Να χρησιμοποιηθεί σε διάφορα Λειτουργικά Συστήματα.

↔ Να διαβαστεί εύκολα και να τροποποιηθεί/αναπτυχθεί από άλλα άτομα, κάνοντας την ομαδική προσπάθεια πολύ πιο δυνατή και εφικτή.

↔ Να αναγνωριστούν εύκολα τυχόν λάθη στον κώδικα και να διορθωθούν.

↔ Να διευρυνθούν εύκολα οι δυνατότητες του προγράμματος με τα λεγόμενα modules (αρθρώματα) στην Python.

Στο διάγραμμα με τις πιο δημοφιλείς γλώσσες προγραμματισμού το 2013, που ακολουθεί η Python βρίσκεται στην πρώτη θέση.



Χαρακτηριστικά της Python

Απλή

Η Python είναι μια απλή και μινιμαλιστική γλώσσα. Το διάβασμα ενός καλού προγράμματος σε Python είναι σαν το διάβασμα των Αγγλικών, αλλά πολύ αυστηρών Αγγλικών! Αυτή η ομοιότητα της Python με ψευδοκώδικα είναι ένα από τα πιο ισχυρά σημεία της. Σας επιτρέπει να συγκεντρώνεστε στη λύση του προβλήματος αντί στην ίδια τη γλώσσα.

Εύκολη στην εκμάθηση

Είναι εξαιρετικά απλό να ξεκινήσετε με την Python. Η Python έχει μια ασυνήθιστα απλή σύνταξη, όπως έχει ήδη αναφερθεί παραπάνω.

Ελεύθερη και Ανοικτού Κώδικα

Η Python είναι ένα παράδειγμα ΕΛΛΑΚ (Ελεύθερο Λογισμικό και Λογισμικό Ανοικτού Κώδικα). Με απλά λόγια, μπορείτε να διανείμετε αντίγραφα αυτού του λογισμικού, να διαβάσετε τον πηγαίο κώδικά του, να κάνετε αλλαγές σ' αυτό και να χρησιμοποιήσετε κομμάτια του σε νέα ελεύθερα προγράμματα. Το ΕΛΛΑΚ βασίζεται στην ιδέα μιας κοινότητας που μοιράζεται τη γνώση. Αυτός είναι ένας από τους

λόγους για τους οποίους η Python είναι τόσο καλή -δημιουργήθηκε και βελτιώνεται συνεχώς από μια κοινότητα που το μόνο που θέλει είναι μια καλύτερη Python.

Φορητή

Λόγω του ανοικτού της κώδικα, η Python έχει υλοποιηθεί (δηλαδή αλλάχθηκε για να λειτουργεί) σε πολλές πλατφόρμες. Όλα τα Python προγράμματά σας μπορούν να δουλέψουν σε οποιαδήποτε από αυτές τις πλατφόρμες χωρίς να χρειάζονται καθόλου αλλαγές αν είστε αρκετά προσεκτικοί ώστε να αποφύγετε να χρησιμοποιήσετε χαρακτηριστικά που εξαρτούνται από κάθε σύστημα.

Μπορείτε να χρησιμοποιήσετε την Python στο Linux, στα Windows, στο FreeBSD, σε Macintosh, στο Solaris, στο OS/2, στην Amiga, στο AROS, στο AS/400, στο BeOS, στο OS/390, στο z/OS, στο Palm OS, στο QNX, στο VMS, στο Psion, στο Acorn RISC OS, στο VxWorks, σε PlayStation, στο Sharp Zaurus, στα Windows CE ακόμα και σε PocketPC !

Γλώσσα υψηλού επιπέδου

Όταν γράφετε προγράμματα στην Python, δε χρειάζεται ποτέ να νοιάζεστε για τις χαμηλού επιπέδου λεπτομέρειες όπως η διαχείριση της μνήμης που χρησιμοποιείται από τα προγράμματά σας, κ.λπ.

Διερμηνευόμενη

Εδώ χρειάζεται μια διεξοδική ανάλυση.

Ένα πρόγραμμα που γράφεται σε μια μεταγλωττιζόμενη γλώσσα όπως η C ή η C++ μετατρέπεται από την πηγαία γλώσσα, για παράδειγμα τη C ή τη C++ σε μια γλώσσα που μιλάει ο υπολογιστής σας (δυαδικός κώδικας δηλαδή 0 και 1) χρησιμοποιώντας ένα μεταγλωττιστή με διάφορες σημαίες και επιλογές. Όταν τρέχετε το πρόγραμμα, ο συνδέτης αντιγράφει το πρόγραμμα στη μνήμη και αρχίζει να το τρέχει.

Η Python, από την άλλη, δε χρειάζεται μεταγλώττιση σε δυαδικό αρχείο. Απλά τρέχετε το πρόγραμμα απ' ευθείας από τον πηγαίο κώδικα. Εσωτερικά, η Python μετατρέπει τον πηγαίο κώδικα σε μια ενδιάμεση μορφή που ονομάζεται bytecode και μετά το μεταφράζει στη γλώσσα του υπολογιστή και μετά το τρέχει. Όλο αυτό, στην πραγματικότητα κάνει τη χρήση της Python πολύ πιο εύκολη αφού δε χρειάζεται να ανησυχείτε για τη μεταγλώττιση του προγράμματος, τη σύνδεση με τις κατάλληλες βιβλιοθήκες, κ.λπ, κ.λπ. Αυτό επίσης κάνει τα προγράμματα της Python εξαιρετικά φορητά, αφού μπορείτε απλά να αντιγράψετε το πρόγραμμα Python που φτιάξατε σε έναν άλλο υπολογιστή και να δουλέψει έτσι απλά!

Επεκτάσιμη

Αν χρειάζεστε ένα κρίσιμο κομμάτι κώδικα να τρέχει πολύ γρήγορα ή αν πρέπει να έχετε ένα κομμάτι ενός αλγόριθμου που να μην είναι ανοικτό, τότε μπορείτε να προγραμματίσετε εκείνο το κομμάτι σε C ή C++ και μετά να το χρησιμοποιείτε από το Python πρόγραμμά σας.

Αντικειμενοστρεφής

Η Python υποστηρίζει τόσο το διαδικασιοστρεφή προγραμματισμό (procedure-oriented) όσο και τον αντικειμενοστρεφή προγραμματισμό (object-oriented). Στο διαδικασιοστρεφή προγραμματισμό, το πρόγραμμα δομείται πάνω σε διαδικασίες ή συναρτήσεις οι οποίες δεν είναι τίποτε άλλο από επαναχρησιμοποιήσιμα κομμάτια από προγράμματα. Στις αντικειμενοστρεφείς γλώσσες, το προγράμματα δομούνται πάνω σε αντικείμενα τα οποία συνδυάζουν δεδομένα και λειτουργικότητα. Η Python έχει έναν πολύ ισχυρό αλλά πολύ απλό τρόπο για αντικειμενοστρεφή προγραμματισμό, ειδικά όταν συγκρίνεται με μεγάλες γλώσσες όπως η C++ ή η Java.

Ενσωματώσιμη

Μπορείτε να ενσωματώσετε την Python μέσα στα προγράμματα σε C/C++ για να τους δώσετε δυνατότητες 'scripting' για τους χρήστες σας.

Εκτεταμένες βιβλιοθήκες

Η Πρότυπη βιβλιοθήκη της Python είναι πραγματικά τεράστια. Μπορεί να σας βοηθήσει να κάνετε διάφορα πράγματα σχετικά με κανονικές εκφράσεις, δημιουργία τεκμηρίωσης, δοκιμές μονάδων, νημάτωση, βάσεις δεδομένων, περιηγητές ιστού, CGI, FTP, email, XML, XML-RPC, HTML, αρχεία WAV, κρυπτογράφηση, γραφικές διεπαφές χρήστη (GUI -graphical user interfaces), Tk, και άλλα πράγματα που εξαρτούνται από το σύστημα. Θυμηθείτε ότι όλα αυτά είναι διαθέσιμα όποτε είναι εγκατεστημένη η Python. Αυτό ονομάζεται φιλοσοφία 'Batteries Included' της Python.

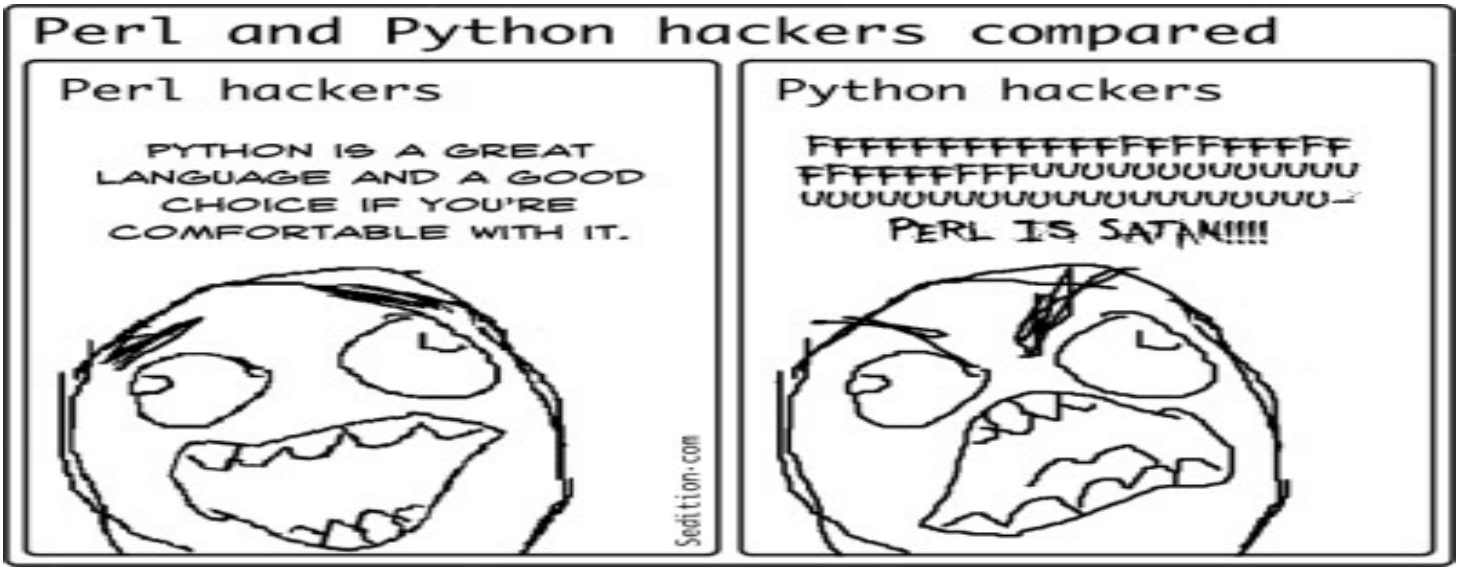
Επιπλεόν από την πρότυπη βιβλιοθήκη, υπάρχουν διάφορες άλλες βιβλιοθήκες υψηλής ποιότητας όπως η wxPython , η Twisted, η Python Imaging Library και πολλές άλλες.

Η Python είναι πραγματικά μια συναρπαστική και ισχυρότατη γλώσσα. Έχει το σωστό συνδυασμό απόδοσης και χαρακτηριστικών που κάνουν τη δημιουργία προγραμμάτων σε Python διασκεδαστική και εύκολη.

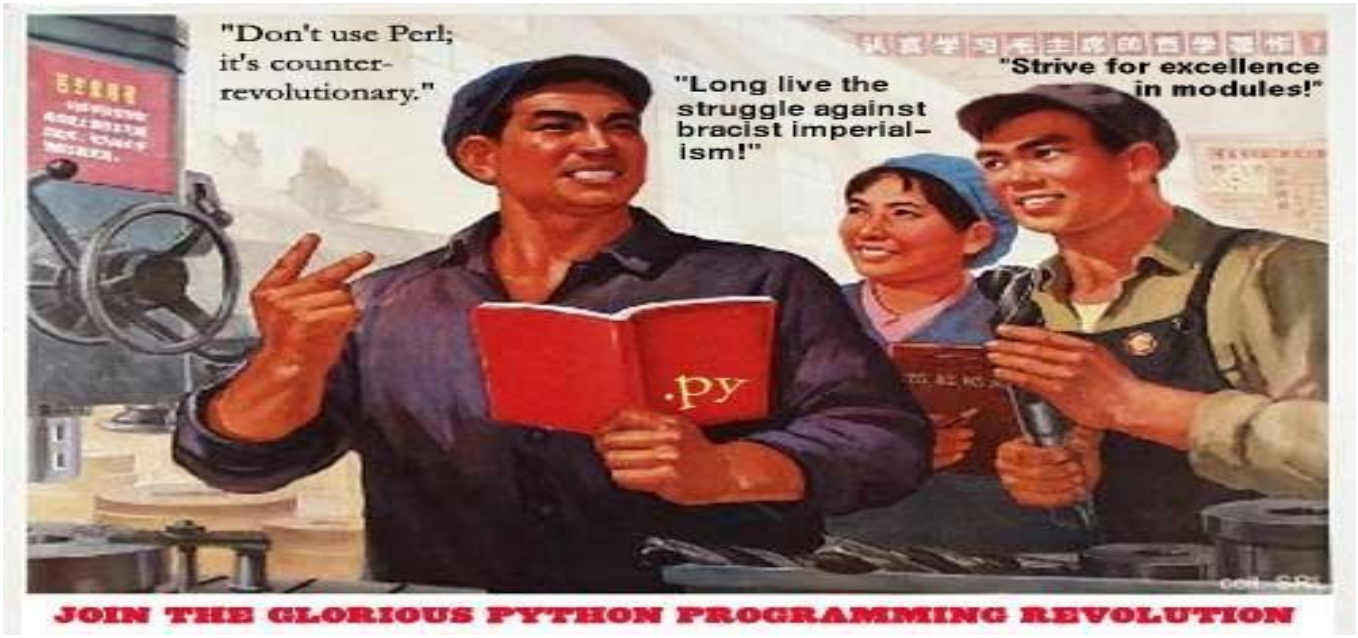
Γιατί Pyhton και όχι Perl;

Η Perl είναι μια ακόμα εξαιρετικά ισχυρή, δημοφιλής, ανοικτού κώδικα, διερμηνεύσιμη γλώσσα προγραμματισμού. Τα προγράμματα σε Perl είναι πολύ εύκολα όταν είναι μικρά, και η γλώσσα διαπρέπει σε μικρά hacks και σενάρια εντολών που "κάνουν τη δουλειά τους". Ωστόσο, γρήγορα γίνεται ανυπόφορη όταν αρχίσετε να γράφετε μεγαλύτερα προγράμματα.

Συγκρινόμενα με την Perl, τα προγράμματα σε Python είναι σίγουρα πιο απλά, πιο καθαρά, πιο εύκολα στη συγγραφή και άρα πιο κατανοητά και εύκολα στη συντήρηση. Πραγματικά θαυμάζω την Perl και τη χρησιμοποιώ καθημερινά για διάφορα πράγματα αλλά όταν γράφω ένα πρόγραμμα πάντα αρχίζω να το σκέφτομαι με τους όρους της Python επειδή έχει γίνει τόσο φυσική για μένα. Η Perl έχει υποστεί τόσα πολλά hacks και αλλαγές, που μοιάζει σαν να είναι η ίδια ένα τεράστιο (αλλά πολύ τεράστιο) hack. Δυστυχώς, η επερχόμενη Perl 6 δε φαίνεται να φέρνει καθόλου βελτιώσεις σ' αυτό τον τομέα.



Το μοναδικό και πολύ σημαντικό πλεονέκτημα που νομίζω ότι έχει η Perl, είναι η τεράστια βιβλιοθήκη CPAN -η Comprehensive Perl Archive Network. Όπως υπαινίσσεται και το όνομα, αυτή είναι μια γιγαντιαία συλλογή από αρθρώματα της Perl και είναι πραγματικά δύσκολο να κατανοήσει κανείς το μέγεθος και το βάθος της -μπορείτε να κάνετε οτιδήποτε σ' έναν υπολογιστή χρησιμοποιώντας αυτά τα αρθρώματα. Ένας από τους λόγους για τους οποίους η Perl έχει περισσότερες βιβλιοθήκες από την Python είναι γιατί υπάρχει εδώ και πολύ περισσότερο καιρό από ότι η Python. Εντούτοις αυτό φαίνεται να αλλάζει με το ολοένα αυξανόμενο Ευρετήριο πακέτων της Python.

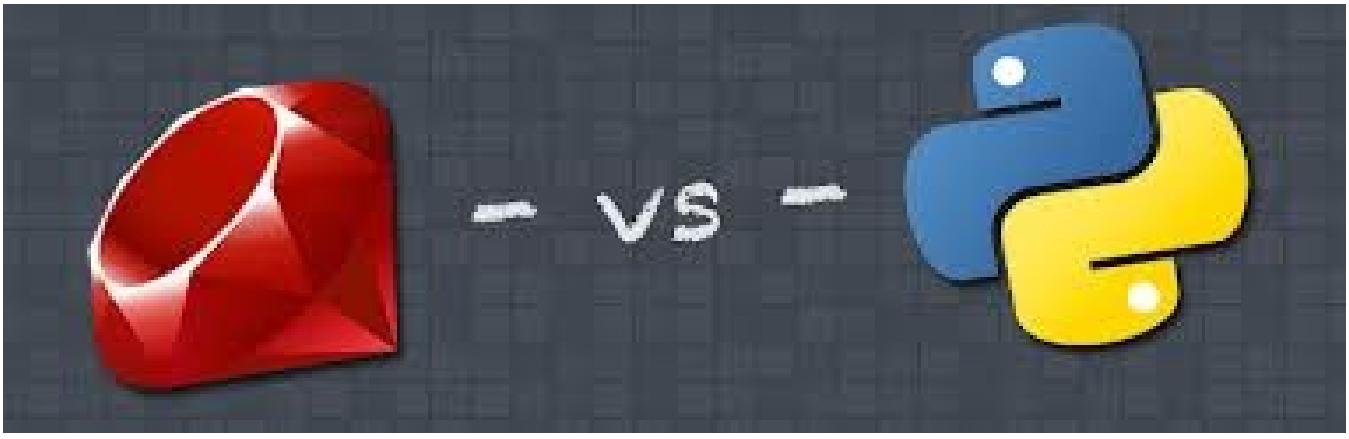


Γιατί Python και όχι Ruby;

Η Ruby είναι μια ακόμα δημοφιλής, ανοικτού κώδικα, διερμηνεύσιμη γλώσσα προγραμματισμού.

Python ή Ruby?? Προτείνεται η Python, καθαρά από την άποψη της ευκολίας εκμάθησης.

Αρκετοί προγραμματιστές θεωρούν δύσκολη στην κατανόηση τη γλώσσα Ruby, αλλά όσοι την καταλαβαίνουν εξυμνούν την ομορφιά της γλώσσας.



Απόψεις σπουδαίων προγραμμαριστών για την Python

- ✓ Ο **Eric S. Raymond** είναι ο συγγραφέας του "Ο καθεδρικός και το παζάρι" και είναι επίσης εκείνος που πρότεινε τον όρο Ανοικτός κώδικας. Λέει ότι η Python έγινε η αγαπημένη του γλώσσα προγραμματισμού.

- ✓ Ο **Bruce Eckel** είναι ο συγγραφέας των διάσημων βιβλίων Thinking in Java και Thinking in C++. Λέει ότι καμιά γλώσσα δεν τον έκανε να νιώσει πιο παραγωγικός από την Python. Λέει ότι η Python είναι ίσως η μόνη γλώσσα η οποία εστιάζει στο να κάνει τα πράγματα πιο εύκολα για τον προγραμματιστή.
- ✓ Ο **Peter Norvig** είναι ένας γνωστός συγγραφέας της Lisp και Διευθυντής ποιότητας αναζητήσεων στο Google.Λέει ότι η Python πάντα ήταν ένα εσωτερικό κομμάτι του Google. Μπορείτε να το επιβεβαιώσετε αυτό κοιτώντας τις αγγελίες Google Jobs στις οποίες αναγράφεται η γνώση της Python ως απαιτούμενη για μηχανικούς λογισμικού.

Python και ταχύτατη ανάπτυξη εφαρμογών

Από την αρχή της ανάπτυξης της γλώσσας αυτής ενθαρρύνεται η ανάπτυξη των εφαρμογών μέσω της Python να είναι όσο πιο απλή γίνεται . Αυτό γίνεται και όσον αφορά την εκμάθηση της ίδιας της γλώσσας, όπου προσπαθείται να υπάρχει μια ομαλή καμπύλη εκμάθησης και όσον αφορά την αναγνωσιμότητα του παραγόμενου κώδικα . Όπως χαρακτηριστικά έχει γραφτεί, για να κάνουμε αποσφαλμάτωση σε ένα κομμάτι κώδικα χρειαζόμαστε διπλάσια εφυΐα από όταν τον γράψαμε. Συνεπώς, αν γράφεις όσο πιο ‘έξυπνο’-δύσκολο κώδικα μπορείς, εκ των πραγμάτων δεν μπορείς να τον αποσφαλματώσεις.

Όλα τα παραπάνω συνηγορούν στην δυνατότητα της Python να επιτρέπει την ταχύτατη ανάπτυξη εφαρμογών ειδικά σε σχέση με άλλες γώσσες χαμηλότερου επιπέδου (πχ C, C++),ενώ λέγεται ότι συνήθως τα προγράμματα σε Python είναι 3-5 φορές μικρότερα σε σχέση με τα αντίστοιχα σε Java. Όσο πιο υψηλού επιπέδου είναι μια γλώσσα προγραμματισμού, τόσο πιο κοντά στην σκέψη του ανθρώπου βρίσκεται. Αυτό σημαίνει ότι είναι πιο εύκολο να γραφτούν προγράμματα σε υψηλού επιπέδου γλώσσες (υψηλό επίπεδο αφαίρεσης) και συνήθως λειτουργούν σε περισσότερες πλατφόρμες. Αυτό όμως γίνεται θυσιάζοντας μέρος της ταχύτητας των προς εκτέλεση προγραμμάτων. Στις μέρες μας παρατηρείται μια σταδιακή στροφή από γλώσσες που επικεντρωνόντουσαν στην απόδοση των προγραμμάτων (efficiency) , να επικεντρώνουν στην απόδοση του προγραμματιστή (productivity).

Python και segmentation faults

Στην Python ξεχάστε τα segmentation faults. Σε αντίστοιχες περιπτώσεις, ο διερεμηνευτής της Python μας ενημερώνει με μια εξαίρεση που πετάει (γίνεται throw) και πλεόν γνωρίζουμε σε ποια γραμμή υπάρχει το πρόβλημα ώστε να το αντιμετωπίσουμε.

Η αυτόματη διαχείριση μνήμης σημαίνει πως δεν χρειάζεται να ανησυχούμε πλέον για το πότε θα ελευθερώσουμε την μνήμη που δεσμεύουμε όταν φτιάχνουμε αντικείμενα. Επίσης, η Python αντιλαμβάνεται πότε το ίδιο αντικείμενο αναφέρεται πάνω από μια φορές κι έτσι δεν το αποθηκεύει στη μνήμη αν δεν χρειάζεται. Η τεχνική αυτή ονομάζεται μέτρηση αναφορών (reference counting).

Python και αντικείμενα

Στην Python, τα πάντα είναι αντικείμενα. Ακόμα και οι συναρτήσεις,οι γεννήτορε (generators) , οι εξαιρέσεις και ό,τι άλλο μπορείτε να σκεφτείτε. Έτσι, προσφέρεται ένας διαισθητικός τρόπος συγγραφής των προγραμμάτων μας με συνέπεια στην αντικειμενοστράφεια, παρόλο που υποστηρίζονται και άλλοι τρόποι προγραμματισμού (όπως συναρτησιακός, διαδικαστικός κ.α).

Στα βασικά χαρακτηριστικά της γλώσσας θα συναντήσουμε τον χειρισμό εξαιρέσεων, πακέτα, κλάσεις ,συναρτήσεις, γεννήτορες (ειδική περίπτωση συναρτήσεων) με ένα τρόπο όπου κάυε ένα θα συμπληρώνει αρμονικά το άλλο για την διευκόλυνση του προγραμματιστή. Αυτό το δέσιμο καθοδηγείται πάντα από την συνέπεια στην αρχή της Python ότι θα έπρεπε να υπάρχει ένας και μόνο ένας προφανής τρόπος να γίνει κάτι, οδηγώντας έτσι στην απλοποίηση των προβλημάτων που μπορεί να αντιμετωπίσει ενας προγραμματιστής.

Επιπροσθέτως, ένα βασικό χαρακτηριστικό της γλώσσας, που ίσως ξενίσει κάποιους, αλλά αναβαθμίζει την αναγνωρισιμότητα του κώδικα, είναι ότι κάθε μπλοκ κώδικα καθορίζεται από την στοίχισή του. Κατά αυτό τον τρόπο, κάποιος είναι υποχρεωμένος να τηρήσει τους κανόνες «καλής συμπεριφοράς» όπως αυτοί επιβάλλονται από τις υπόλοιπες γλώσσες προγραμματισμού καθώς θα πρέπει να ενσωματώσει στον τρόπο που γράφει τον κώδικά του μια συνέπεια στον καθορισμό της στοίχισής του. Για κόμα καλύτερα αποτελέσματα έχουν γραφτεί ειδικές προτάσεις (Python Enhancement Proposal (PEP)) τα οποία διευκρινίζουν τα συγκεκριμένα θέματα και καθορίζουν έναν επίσημο τρόπο συγγραφής του κώδικα (παραπομπή στα PEP 8 (http://www.python.org/dev/peps/pep-0008/) και PEP 257 (http://www.python.org/dev/peps/pep-0257/)).

Python 3

Η Python όπως αναφέρθηκε και παραπάνω αναπτύχθηκε το 1990. Αυτό σημαίνει πως εκτός από την εμπειρία και την ωρίμανση στο κώδικα που είχαν αποφέρει όλα αυτά τα χρόνια, υπήρχαν και βάρη από το παρελθόν που την εμπόδιζαν να προχωρήσει μπροστά όπως επιτάσσουν οι αρχές πάνω στις οποίες είναι δημιουργημένη και ταυτόχρονα να καλύπτει τις σύγχρονες ανάγκες.Στη συνέχεια παραθέτω κάποια παραδείγματα περίεργων συμπεριφορών:

- ⇨ 7/4 = 1 επηρεασμένη από την αριθμητική ακεραίων στη C
- ⇨ Range και γεννήτορες (άσκοπη χρήση μνήμης)
- ⇨ Εκτύπωση print statement όπου αν χρειαζόταν κάτι παραπάνω από την τυπική συμπεροφορά γινόταν σχετικά περίπλοκο.
- ⇨ Οργάνωση κύριας βιβλιοθήκης η οποία πλέον ξέφευγε από τον κύριο τρόπο ονοματολογίας και περιείχε συναρτήσεις που προτείνονται να μην χρησιμοποιούνται πια.
- ⇨ Η υποστήριξη Unicode ήθελε κάποια (μικρή έστω) προσπάθεια.

Για να ξεπεραστούν τα προβλήματα που αναφέρονται παραπάνω, έπρεπε να παρθεί μια τολμηρή απόφαση. Το μεγάλο βήμα λοιπόν για την Python έγινε με την έκδοση 3 (ή αλλιώς py3k ή 3000) . Το μεγαλύτερ μέρος της γλώσσας είναι σχεδόν το ίδιο, αλλά πλέον έμειναν μια και καλή στο παρελθόν ιδιάζουσες συμπεριφορές ενώ εισήχθηκαν καλύτεροι τρόποι επίλυσης ορισμένων προβλημάτων.Πιο συγκεκριμένα οι αλλαγές που έγιναν περιλαμβάνουν :

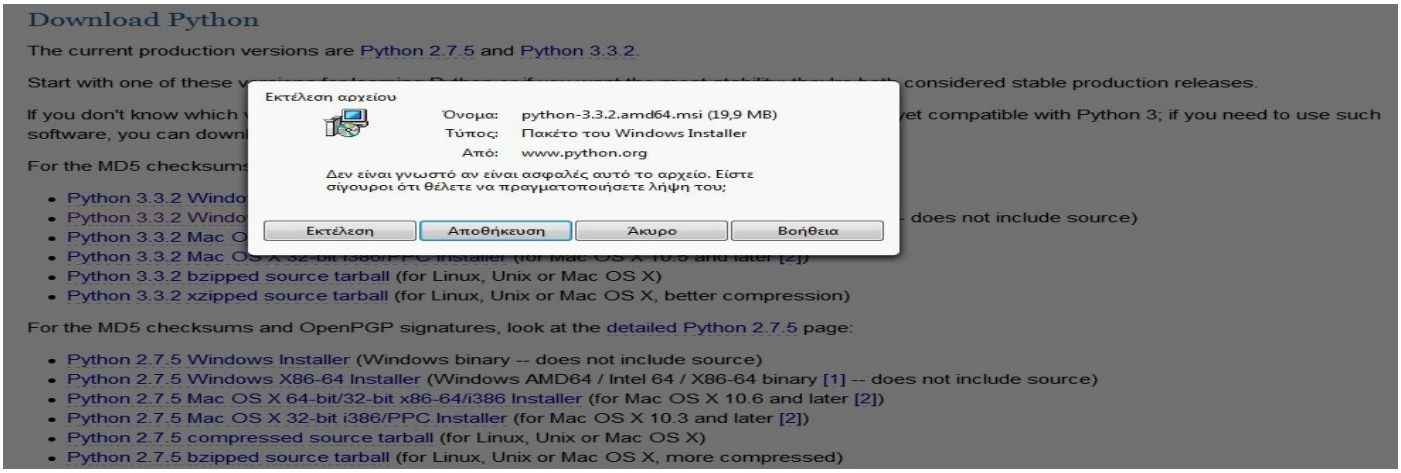
- ⇨ 7/4=1,75 (και ειδικός τελεστής // πλέον αφορά αριθμητική ακεραίων)
- ⇨ Εκτεταμένη χρήση γεννητόρων. Προτιμήθηκε η ευρεία χρήση γεννητόρων στην βασική βιβλιοθήκη ώστε για παράδειγμα η range να παράγει τους αριθμούς που χρειάζονται μόνο όταν χρειάζονται και να μην γεμίζει άσκοπα η μνήμη.
- ⇨ Η print έγινε συνάρτηση
- ⇨ Οι βιβλιοθήκες αναδιοργανώθηκαν
- ⇨ Εύκολη υποστήριξη Unicode, χωρίς να απαιτείται σχεδόν καθόλου προσπάθεια

Ας ξεκινήσουμε λοιπόν ένα ενδιαφέρον ταξίδι…

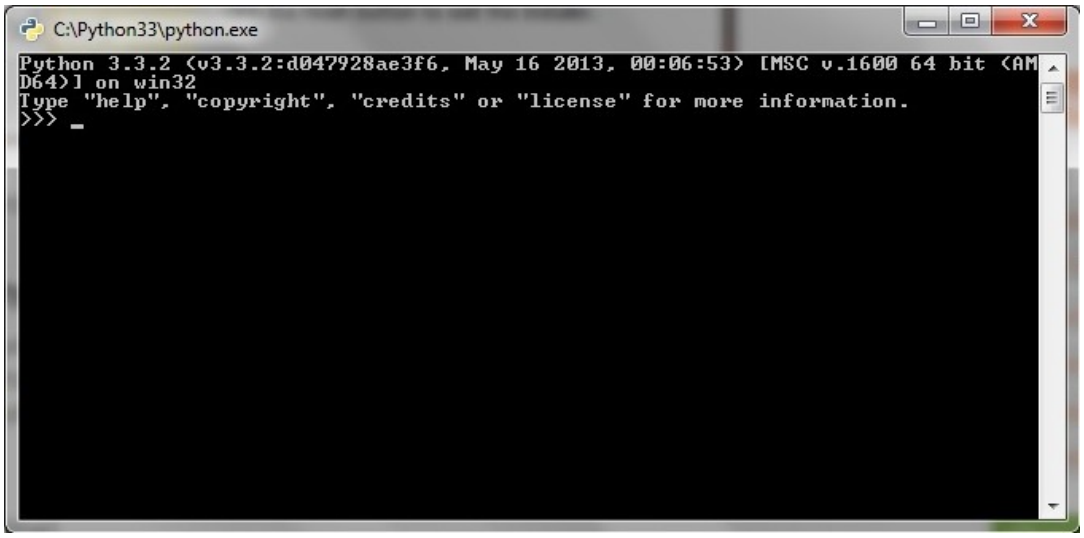
Πως εγκαθιστώ την Python στον υπολογιστή μου;

Για χρήστες Windows

Επισκεπτόμαστε το επίσημο site της Python **http://www.python.org/download/** και κατεβάζουμε το αρχείο εγκατάστασης που ταιριάζει στο λειτουργικό μας σύστημα και τα χαρακτηριστικά του προσωπικού μας υπολογιστή. (Θα εγκαταστήσουμε την Python 3.3.2 σε σύστημα 64bit που φιλοξενεί Windows λειτουργικό σύστημα, επομένως κατεβάζουμε το αρχείο εγκατάστασης python-3.1.1.amd64.msi)



Ακολουθούμε τις οδηγίες του οδηγού εγκατάστασης χωρίς να αλλάξουμε καμία απολύτως ρύθμιση. Σε λίγα λεπτά η εγκατάσταση έχει ολοκληρωθεί και είμαστε έτοιμοι να γράψουμε το πρώτο μας πρόγραμμα.



Το περιβάλλον της Python 3.3.2

Στον φάκελο Έναρξη → Όλα τα προγράμματα → Python (που αποτελεί το βασικό μενού εικονιδίων της Python) θα βρούμε τις εξής επιλογές:

- ✓ IDLE (Python GUI)
- ✓ Module Docs

- ✓ Python (command line)

IDLE (Python GUI)

Αποτελεί μια διεπαφή χρήστη που, εκτός από το γεγονός ότι ανοίγει τον διερμηνευτή της Python, προσφέρει κάποιες ευκολίες στον προγραμματιστή όπως να ανοίγει και να επεξεργάζεται εφαρμογές python, να κάνει debugging και πολλές από τις ευκολίες που παρέχει ένας μέσος επεξεργαστής κειμένου (εύρεση / αντικατάσταση, λειτουργίες αντιγραφής / αποκοπής / επικόλλησης, κλπ.). Αποτελεί το εργαλείο που θα χρησιμοποιήσουμε για να δημιουργούμε, να τρέχουμε, να δοκιμάζουμε και να εκτελούμε λειτουργίες debugging στην τρέχουσα σειρά σεμιναρίων. Φυσικά, για την συγγραφή των Python εφαρμογών μας μπορούμε να χρησιμοποιήσουμε εναλλακτικά οποιονδήποτε άλλο επεξεργαστή κειμένου μας βολεύει.

Module Docs

Πρόκειται για ένα εργαλείο που επιτρέπει στον προγραμματιστή να περιηγηθεί στα modules που περιέχονται εγγενώς στην Python καθώς και να εμφανίσει πληροφορίες για τις λειτουργίες τους. Τέλος δίνει τη δυνατότητα στον χρήστη να δει τα πάντα συγκεντρωμένα μέσω του αγαπημένου του web browser.

Python (command line)

Ουσιαστικά ανοίγει τον διερμηνευτεί της Python σε ένα περιβάλλον DOS ώστε να μπορούμε να «τρέξουμε» τις εφαρμογές μας. Ωστόσο δεν προσφέρει καμία επιπλέον δυνατότητα. Χρησιμοποιείτε συνήθως για να τρέχουμε προγράμματα τα οποία έχουν ήδη δοκιμαστεί για σφάλματα.

DOS Prompt

Αν θέλετε να μπορείτε να χρησιμοποιήσετε την Python από τη γραμμή εντολών των Windows π.χ. το DOS prompt, τότε θα πρέπει να ορίσετε τη μεταβλητή συστήματος PATH κατάλληλα.Για τα Windows 2000, XP, 2003, πατήστε στο Control Panel -> System -> Advanced ->Environment Variables. Επιλέξτε τη μεταβλητή που

ονομάζεται PATH στην ενότητα 'System Variables', μετά επιλέξτε Edit και προσθέστε το ;C:\Python30 μετά από ό,τι είναι ήδη γραμμένο εκεί. Φυσικά, χρησιμοποιήστε το κατάλληλο όνομα φακέλου, εκεί όπου εγκαταστήσατε την Python.

Για παλαιότερες εκδόσεις των Windows, προσθέστε την ακόλουθη γραμμή στο αρχείο C:\AUTOEXEC.BAT :'PATH=%PATH%;C:\Python30' (χωρίς τα εισαγωγικά) και επανεκκινήστε το σύστημα. Για τα Windows NT, χρησιμοποιήστε το αρχείο AUTOEXEC.NT.

Για χρήστες Linux και BSD

Αν χρησιμοποιείτε μια διανομή Linux όπως το Ubuntu, το Fedora, το OpenSUSE ή ένα σύστημα BSD όπως το FreeBSD, τότε πιθανότατα θα έχετε ήδη εγκατεστημενη την Python στο σύστημά σας.

Για να ελέγξετε αν έχετε ήδη εγκατεστημένη την Python στο Linux σας, ανοίξτε ένα πρόγραμμα κελύφους (όπως το konsole ή το gnome-terminal) και πληκτρολογήστε την εντολή python -V όπως φαίνεται παρακάτω.

```
$ python -V
Python 3.0b1
```

Σημαντική παρατήρηση !

Το $ είναι το prompt του κελύφους, δηλαδή εκεί που πληκτρολογείτε τις εντολές. Θα είναι διαφορετικό ανάλογα με τις ρυθμίσεις του λειτουργικού σας συστήματος, οπότε θα υποδεικνύω πάντα το prompt απλά με το σύμβολο $.

Αν δείτε μερικές πληροφορίες έκδοσης όπως αυτές που φαίνονται πιο πάνω, τότε έχετε ήδη την Python εγκατεστημένη.

Ωστόσο, αν πάρετε ένα μήνυμα σαν κι αύτο:

```
$ python -V
bash: Python: command not found
```

Τότε δεν έχετε εγκατεστημένη την Python. Αυτό είναι εξαιρετικά απίθανο αλλά όχι αδύνατο.

Σημείωση :

Αν έχετε ήδη εγκατεστημένη την Python 2.x, τότε δοκιμάστε **python3 -V**.

Σ' αυτή την περίπτωση, υπάρχουν δύο τρόποι για να εγκαταστήσετε την Python 3.x στον υπολογιστή σας.

1) Μπορείτε να μεταγλωττίσετε την Python από τον πηγαίο κώδικα [1] και να την εγκαταστήσετε. Οι οδηγίες μεταγλώττισης παρέχονται στον ιστότοπο.
2) Μπορείτε να εγκαταστήσετε τα δυαδικά πακέτα χρησιμοποιώντας το διαχειριστή πακέτων λογισμικού που συνοδεύει το λειτουργικό σας σύστημα, όπως το apt-get στο Ubuntu/Debian και στις άλλες διανομές Linux που βασίζονται στο Debian, το yum στο Fedora Linux, το pkg_add στο FreeBSD, κ.λπ. Σημειώστε ότι θα χρειαστείτε μια σύνδεση στο διαδίκτυο για να χρησιμοποιήσετε αυτή τη μέθοδο. Εναλλακτικά, μπορείτε να κατεβάσετε τα δυαδικά αρχεία από κάπου αλλού, να τα αντιγράψετε μετά στον υπολογιστή σας και να τα εγκαταστήσετε.

Για χρήστες Mac OS X

Οι περισσότεροι χρήστες Mac OS X θα βρουν την Python ήδη εγκατεστημένη στα συστήματά τους. *Ανοίξτε το **Terminal.app** και πληκτρολογήστε **python -V** και ακολουθήστε τις οδηγίες στο πιο πάνω τμήμα για το Linux.*

Για ένα σύστημα Linux, πιθανότατα έχουμε ήδη την Python εγκατεστημένη στο σύστημά μας. Σε αντίθετη περίπτωση, μπορούμε να την εγκαταστήσουμε από το διαχειριστή πακέτων λογισμικού που συνοδεύει τη διανομή μας. Για ένα σύστημα Windows, η εγκατάσταση της Python είναι τόσο εύκολη όσο το κατέβασμα του αρχείου εγκατάστασης και το διπλό κλικ πάνω του. Στο εξής, θα υποθέσουμε ότι έχουμε την Python εγκατεστημένη το σύστημά μας.

Επιλογή ενός επεξεργαστή κώδικα (Editor)

Προτού προχωρήσουμε στη συγγραφή προγραμμάτων Python σε αρχεία πηγαίου κώδικα, χρειαζόμαστε έναν **επεξεργαστή κώδικα** για να γράψουμε τον κώδικά μας. *Η επιλογή ενός επεξεργαστή είναι εξαιρετικά σημαντική*. Πρέπει να επιλέξετε τον επεξεργαστή που θα χρησιμοποιήσετε όπως θα επιλέγατε ένα σπίτι που θα αγοράζατε. Ένας καλός επεξεργαστής θα σας βοηθήσει να γράψετε προγράμματα

Python εύκολα, κάνοντας το ταξίδι σας πιο άνετο και σας βοηθά να φθάσετε στον προορισμό σας (την επίτευξη του στόχου σας) με έναν πολύ πιο εύκολο και ασφαλή τρόπο.

Μία από τις πολύ βασικές απαιτήσεις είναι η **χρωματική επισήμανση σύνταξης** όπου όλα τα διαφορετικά τμήματα του Python προγράμματος χρωματίζονται κατάλληλα έτσι ώστε να μπορείτε να δείτε το πρόγραμμα και να έχετε μία εικόνα της εκτέλεσής του. Επίσης, η επισήμανση μας βοηθά να εντοπίζουμε τυχόν συντακτικά λάθη που έχουμε κάνει στον κώδικα επειδή δεν θα χρωματίζεται σωστά όπως θα αναμενόταν μια λέξη ή μια εντολή.

Για χρήστες Windows, προτείνεται το IDLE. Το IDLE κάνει συντακτική επισήμανση του κώδικα και πολλά περισσότερα όπως το ότι σας επιτρέπει να τρέχετε τα προγράμματά σας μέσα από το IDLE ανάμεσα σε άλλα πράγματα. Σημαντική σημείωση: Μη χρησιμοποιήσετε το Notepad –είναι μία κακή επιλογή επειδή δεν κάνει συντακτική επισήμανση και επίσης δεν υποστηρίζει τη στοίχιση του κειμένου, που είναι ένα πολύ σημαντικό χαρακτηριστικό στην περίπτωσή μας όπως θα δούμε αργότερα. Καλοί επεξεργαστές όπως το IDLE (και ο VIM επίσης) θα σας βοηθήσουν αυτόματα σε αυτό.

Για χρήστες Linux/FreeBSD, υπάρχει πλήθος επιλογών για τον επεξεργαστή που θα χρησιμοποιήσετε. Εάν ξεκινάτε μόλις τώρα τον προγραμματισμό, πιθανόν θα θέλετε να χρησιμοποιήσετε το ***Geany***. Έχει μία φιλική γραφική διεπαφή και κουμπιά για την εύκολη μεταγλώττιση και εκτέλεση των Python προγραμμάτων σας.

Ωστόσο, εάν θέλετε, μπορείτε να αρκεστείτε στους επεξεργαστές κειμένου Gedit και Kate, οι οποίοι υπάρχουν εγκατεστημένοι από προεπιλογή στις περισσότερες διανομές Linux που χρησιμοποιούν το GNOME ή το KDE, αντίστοιχα. Σημειώστε ότι οι επεξεργαστές αυτοί χρωματίζουν τον κώδικα για να τον κάνουν πιο ευανάγνωστο, αλλά δεν παρέχουν εργαλεία εκτέλεσής του. Θα πρέπει να εκτελείτε τα προγράμματά σας έξω από τον επεξεργαστή κειμένου.

Οι έμπειροι προγραμματιστές, χρησιμοποιούν το Vim ή τον Emacs. Πρόκειται για δύο από τους πιο ισχυρούς επεξεργαστές κώδικα και θα οφεληθείτε από τη χρήση τους για τη συγγραφή των Python προγραμμάτων σας. Σε περίπτωση που είστε διατεθειμένοι να αφιερώσετε χρόνο ώστε να μάθετε το Vim ή τονEmacs, τότε συνιστάται ανεπιφύλακτα να μάθετε να χρησιμοποιείτε οποιονδήποτε από τους δύο καθώς θα σας φανούν πολύ χρήσιμοι στην πορεία.

- ✓ Σε αυτή την εργασία θα χρησιμοποιήσω το **IDLE**, το IDE (Integrated Development Environment -Ολοκληρωμένο Περιβάλλον Ανάπτυξης) και

επεξεργαστή κώδικα που επέλεξα. Το IDLE εγκαθίσταται εξ' ορισμού από το πρόγραμμα εγκατάστασης της Python σε Windows και OS X. Είναι επίσης διαθέσιμο για εγκατάσταση σε Linux και BSD στα αντίστοιχα αποθετήρια (repositories).

Αν παρ' όλα αυτά θέλετε να εξερευνήσετε άλλες επιλογές όσον αφορά τον επεξεργαστή κώδικα, δείτε τη λίστα επεξεργαστών κώδικα για Python και κάντε την επιλογή σας. (http:/ / www. python. org/ cgi-bin/ moinmoin/ PythonEditors)Μπορείτε επίσης να επιλέξετε ένα IDE (Ολοκληρωμένο Περιβάλλον Ανάπτυξης) για Python. Δείτε την ολοκληρωμένη λίστα IDE που υποστηρίζουν την Python για περισσότερες λεπτομέρειες (http:/ / www. python. org/ cgi-bin/ moinmoin/ IntegratedDevelopmentEnvironments). Αφού ξεκινήσετε να γράφετε μεγάλα προγράμματα σε Python, τα ολοκληρωμένα περιβάλλοντα ανάπτυξης μπορούν να φανούν εξαιρετικά χρήσιμα.

Ξανατονίζω ότι είναι πολύ σημαντική η επιλογή ενός κατάλληλου επεξεργαστή για Python, διότι μπορεί να κάνει τη συγγραφή προγραμμάτων σε Python πιο διασκεδαστική και εύκολη.

Για τους χρήστες του Vim

Υπάρχει μία καλή εισαγωγή στο πώς να μετατρέψετε το Vim σε ένα ισχυρό Python IDE από τον John M. Anderson . (http:/ / blog. sontek. net/ 2008/ 05/ 11/ python-with-a-modular-ide-vim/)

Για τους χρήστες του Emacs

Υπάρχει μία καλή εισαγωγή στο πώς να μετατρέψετε τον Emacs σε ένα ισχυρό Python IDE από τον Ryan McGuire. (http:/ / www. enigmacurry. com/ 2008/ 05/ 09/ emacs-as-a-powerful-python-ide/)

Χρησιμοποιώντας ένα αρχείο πηγαίου κώδικα

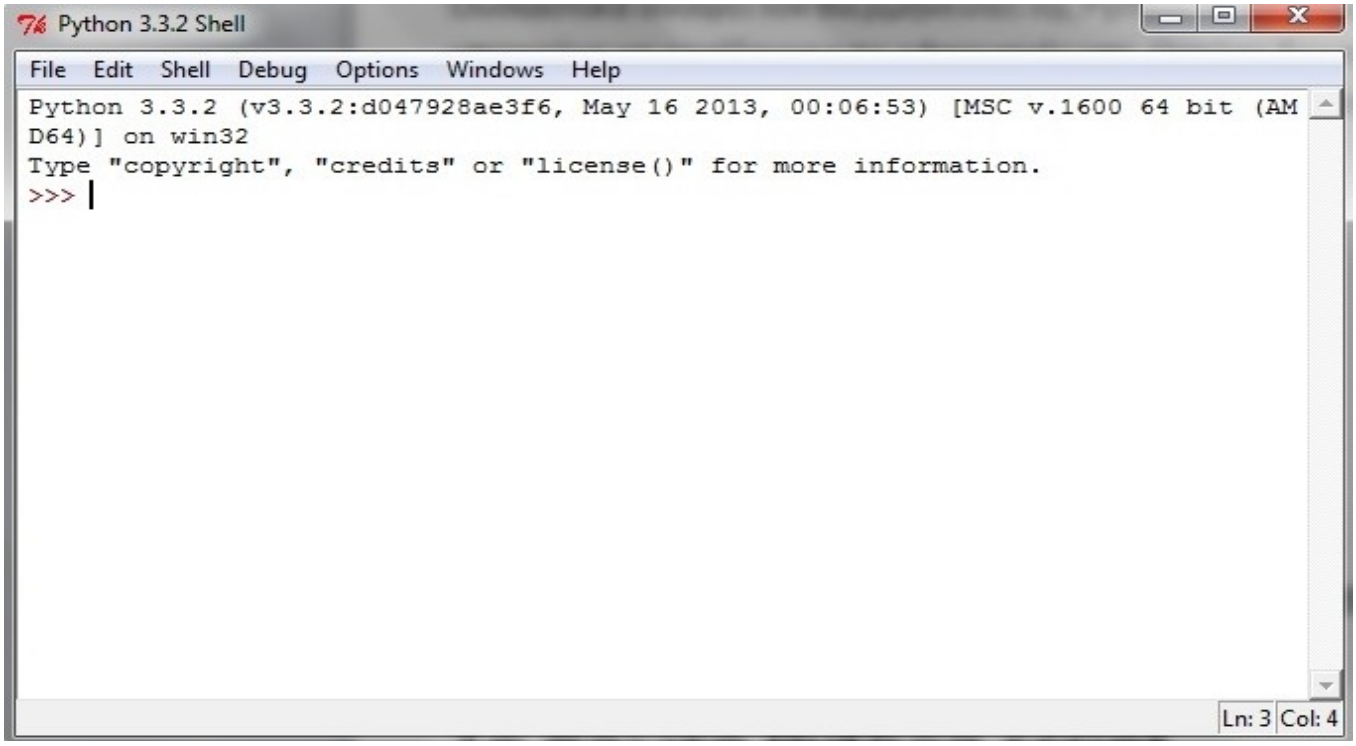
Το πρώτο Python script

Είναι παράδοση όποτε μαθαίνουμε μία καινούργια γλώσσα προγραμματισμού, το πρώτο πρόγραμμα που γράφουμε και τρέχουμε να είναι το **πρόγραμμα «Hello, World!» («Χαίρε, Κόσμε!»)** -το μόνο που κάνει είναι να λέει «Χαίρε, Κόσμε!» όταν το τρέξουμε. Όπως έχει πει και ο Simon Cozens , είναι «το παραδοσιακό ξόρκι προς

τους θεούς του προγραμματισμού για να μας βοηθήσουν να μάθουμε τη γλώσσα καλύτερα» . Πάραυτα εδώ θα τρέξουμε το πρόγραμμα .

Εδώ θα πρωτοτυπήσουμε και θα εμφανίσουμε στο πρώτο μας script το μήνυμα «Welcome to Python world!»

→ Ανοίγουμε το IDLE (Python GUI).

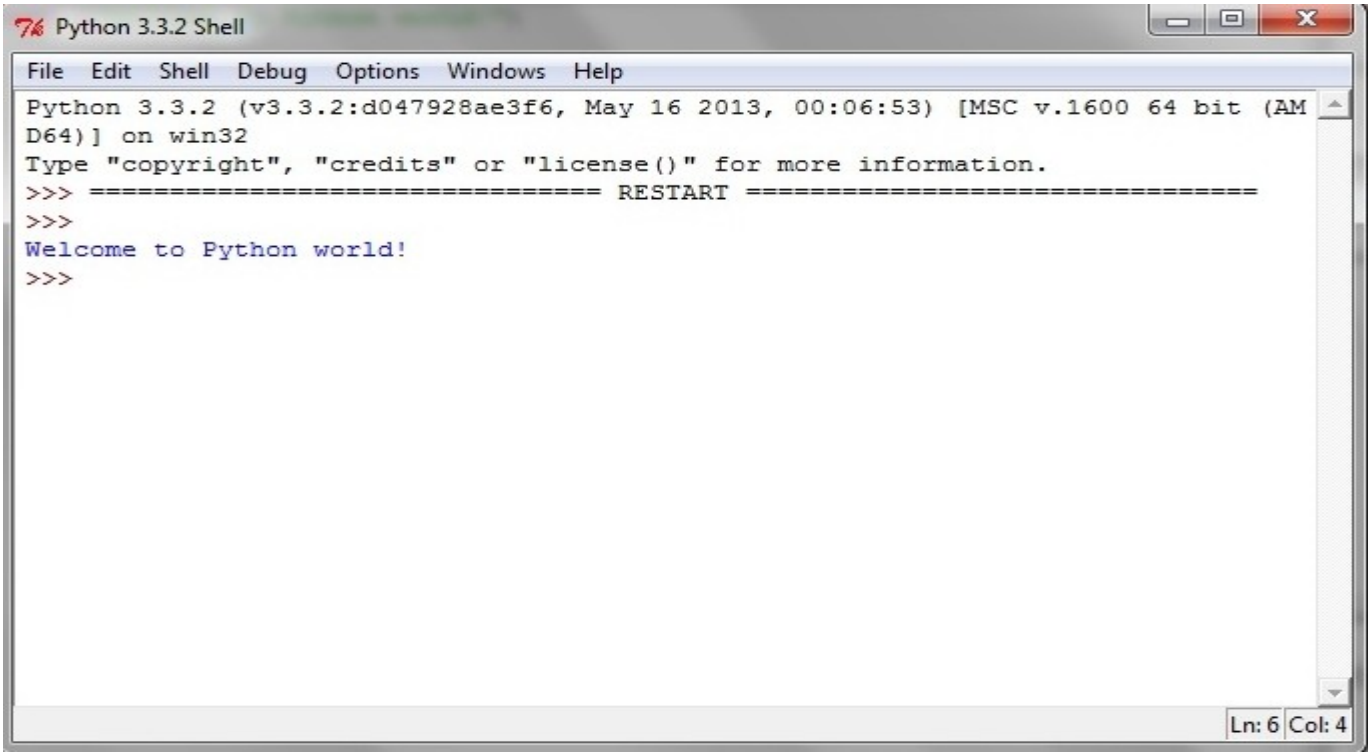


→ Επιλέξτε File → New Window ή εκτελούμε Ctrl + N και εισάγετε πρόγραμμα.
print ('Welcome to Python world').
Έπειτα επιλέξτε File → Save ή Ctrl+S και αποθηκεύστε το ως welcometopython.py

→ Στη συνέχεια, μένοντας στο παράθυρο του κώδικα, επιλέγουμε Run module ή πατούμε το F5.

→ Μπορούμε επίσης να τρέξουμε αυτό το πρόγραμμα ανοίγοντας ένα κέλυφος (τερματικό στο Linux ή γραμμή εντολών στο DOS) και εισάγοντας την **εντολή python3.3.2 welcometopython.py**.

→ Σημειώστε ότι πριν δώσετε την εντολή για την εκτέλεση του προγράμματος helloworld.py θα πρέπει να έχετε μεταβεί στο φάκελο όπου είναι αποθηκευμένο το αρχείο welocometopython.py.



Παρατηρήσεις :

Η Python σας δίνει την έξοδο (output) της γραμμής άμεσα! Αυτό που μόλις γράψατε είναι μία εντολή (statement) σε Python. Χρησιμοποιούμε την print (τύπωσε) για να τυπώσουμε οποιαδήποτε τιμή της παρέχουμε.

- Κλείσιμο της κονσόλας του διερμηνευτή : Για να κλείσετε την κονσόλα, πιέστε ctrl-d (σύμβολο EOF - End Of File) αν χρησιμοποιείτε το IDLE ή κάποιο κέλυφος Linux/BSD. Αν χρησιμοποιείτε τη γραμμή εντολών των Windows, πιέστε ctrl-z και μετά πιέστε Enter.
- Σε περίπτωση που πήρατε κάποιο μήνυμα λάθους, ξαναγράψτε το παραπάνω πρόγραμμα ακριβώς όπως το βλέπετε πιο πάνω και τρέξτε το ξανά.
- Σημειώνουμε οτι η Python κάνει διάκριση πεζών-κεφαλαίων,για παράδειγμα το print δεν είναι το ίδιο με το Print.
- Επίσης, βεβαιωθείτε οτι δεν υπάρχουν κενά ή στηλοθέτες (tabs) πριν τον πρώτο χαρακτήρα (γράμμα) σε κάθε γραμμή -θα δούμε γιατί αυτό είναι σημαντικό αργότερα.
- Ας σταθούμε στις δύο πρώτες γραμμές του προγράμματος..
  Αυτές ονομάζονται σχόλια -οτιδήποτε δεξιά του συμβόλου # είναι σχόλιο. Τα σχόλια χρησιμοποιούνται κυρίως ως σημειώσεις για τον αναγνώστη του προγράμματος.
- Η Python δε λαμβάνει υπ' όψη της τα σχόλια, εκτός από την ειδική περίπτωση της πρώτης γραμμής όπως εδώ. Αυτή αποκαλείται **γραμμή shebang** -

οποτεδήποτε οι πρώτοι δύο χαρακτήρες του αρχείου πηγαίου κώδικα είναι #! ακολουθούμενοι από την τοποθεσία ενός προγράμματος, αυτό πληροφορεί το Linux/Unix σύστημα ότι αυτό το πρόγραμμα θα πρέπει να εκτελεστεί με αυτό το διερμηνευτή όταν θα έρθει η ώρα της εκτέλεσής του.

```
#!/usr/bin/python3.3
#Filename: welcometopython.py
print ('Welcome to Python world!')
```

- Σημειώστε ότι μπορείτε πάντα να τρέξετε το πρόγραμμα σε οποιαδήποτε πλατφόρμα καθορίζοντας το διερμηνευτή απ' ευθείας στη γραμμή εντολών όπως για παράδειγμα με την εντολή python3.3.2 welcometopython.py.
- Χρησιμοποιείτε τα σχόλια στα προγράμματά σας για να εξηγήσετε κάποιες σημαντικές λεπτομέρειες του προγράμματος -είναι χρήσιμο για τους αναγνώστες του προγράμματός σας έτσι ώστε να μπορούν εύκολα να καταλάβουν τι κάνει το πρόγραμμα. Θυμηθείτε, αυτό το άτομο μπορεί να είστε εσείς ο ίδιος μετά από έξι μήνες!

- Τα σχόλια ακολουθούνται από μία εντολή Python. Εδώ καλούμε τη συνάρτηση (function) print που απλά τυπώνει το κείμενο 'Welcome to Python world!'. Αργότερα θα αναλύσουμε τις συναρτήσεις -αυτό που πρέπει να κατανοήσετε αυτή τη στιγμή είναι πως οτιδήποτε δώσετε ανάμεσα στις παρενθέσεις θα τυπωθεί στην οθόνη. Σε αυτή την περίπτωση, δίνουμε το 'Welcome to Python world!' το οποίο αποκαλούμε συμβολοσειρά (string).

Εκτελέσιμα προγράμματα Python

Τα παρακάτω έχουν ισχύ μόνο για χρήστες Linux/Unix αλλά οι χρήστες Windows μπορεί να είναι εξίσου περίεργοι σχετικά με την πρώτη γραμμή του προγράμματος. Πρώτα, πρέπει να δώσουμε στο πρόγραμμα δικαίωμα εκτέλεσης χρησιμοποιώντας την **εντολή chmod** και μετά να τρέξουμε το πηγαίο πρόγραμμα.

```
$ chmod a+x welcometopython.py

$ ./welcometopython.py

Welcome to Python world!
```

- ⇨ Η **εντολή chmod** χρησιμοποιείται εδώ για να αλλάξουμε (change) την κατάσταση (mode) του αρχείου δίνοντας δικαιώματα εκτέλεσης (execute) σε όλους (all) τους χρήστες του συστήματος.
- ⇨ Έπειτα, εκτελούμε το πρόγραμμα απ' ευθείας καθορίζοντας την τοποθεσία του αρχείου πηγαίου κώδικα. Χρησιμοποιούμε το **./** για να υποδηλώσουμε οτι το πρόγραμμα βρίσκεται στην τρέχουσα διαδρομή (στο φάκελο που είμαστε).

Τι γίνεται όμως αν δε γνωρίζουμε πού βρίσκεται ο διερμηνευτής της Python;

Τότε, μπορείτε να χρησιμοποιήσετε το **ειδικό πρόγραμμα env** που είναι διαθέσιμο σε συστήματα Linux/Unix. Απλάτροποποιήστε την πρώτη γραμμή του προγράμματος ώς εξής:

```
#!/usr/bin/env python3
```

Το πρόγραμμα env θα αναζητήσει το διερμηνευτή της Python με τον οποίο θα εκτελεστεί το πρόγραμμά μας.

Μέχρι τώρα, μπορούσαμε να τρέξουμε το πρόγραμμά μας εφ' όσον γνωρίζαμε ακριβώς τη διαδρομή προς αυτό. Αν όμως θέλαμε να μπορούμε να το τρέξουμε από οπουδήποτε; Αυτό μπορούμε να το επιτύχουμε αποθηκεύοντας το πρόγραμμα σε μία από τις διαδρομές που περιέχονται στη μεταβλητή περιβάλλοντος PATH. Κάθε φορά που τρέχετε οποιοδήποτε πρόγραμμα, το σύστημα το αναζητά σε κάθε μία από της διαδρομές που περιέχονται στη μεταβλητή περιβάλλοντος PATH και, αν βρεθεί, το πρόγραμμα εκτελείται. Μπορούμε να κάνουμε το πρόγραμμά μας διαθέσιμο από παντού απλά αντιγράφοντας το αρχείο πηγαίου κώδικα σε μία από τις διαδρομές που περιέχονται στην PATH.

```
$ echo $PATH
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games
```

```
$ cp welcometopython.py /usr/local/bin/welcometopython
```

```
$ welcome to Python
```

```
Welcome to Python world!
```

Σημειώνουμε ότι οι φάκελοι που περιέχονται στην PATH είναι συνήθως φάκελοι του συστήματος. Γι' αυτό χρειάζεστε δικαιώματα υπερχρήστη για να αντιγράψετε σ' αυτούς ένα αρχείο, π.χ. βάζοντας μπροστά στην εντολή σας το sudo, δηλαδή: sudo cp helloworld.py /usr/local/bin/helloworld ή άλλη εντολή που σας δίνει δικαιώματα υπερχρήστη (συμβουλευτείτε την τεκμηρίωση του συστήματός σας γι' αυτό το θέμα).

Μπορούμε να δούμε τα περιεχόμενα της PATH χρησιμοποιώντας την **εντολή echo** (σε τερματικό) και προσθέτοντας πριν το όνομα της μεταβλητής το **σύμβολο $** για να δείξουμε στο κέλυφος ότι θέλουμε την τιμή της συγκεκριμένης μεταβλητής. Μπορείτε να προσθέσετε μία διαδρομή στη μεταβλητή PATH δίνοντας για

παράδειγμα σε τερματικό την εντολή PATH=$PATH:/tonia/pythonproject/programs όπου '/tonia/pythonproject/programs' είναι η διαδρομή που θέλετε να προσθέσετε.

Σημαντική παρατήρηση

Η προσθήκη μιας διαδρομής στην $PATH με την εντολή `PATH=$PATH:/tonia/pythonproject/programs` λειτουργεί προσωρινά, και το αποτέλεσμά της χάνεται μετά από επανεκκίνηση του συστήματος. Συμβουλευτείτε την τεκμηρίωση του συστήματός σας για μόνιμη προσθήκη διαδρομής στην $PATH. Αυτή η μέθοδος είναι πολύ χρήσιμη αν θέλετε να γράψετε χρήσιμα scripts που θέλετε να μπορείτε να τα τρέχετε από οπουδήποτε, οποτεδήποτε. Είναι σα να δημιουργείτε τις δικές σας εντολές όπως π.χ. η εντολή cd ή οποιαδήποτε άλλη εντολή χρησιμοποιείτε σε ένα τερματικό στο Linux ή στη γραμμή εντολών του DOS.

“Βοήθεια” στην Python

Αν χρειάζεστε γρήγορα πληροφορίες σχετικά με μία συνάρτηση ή εντολή στην Python, τότε μπορείτε να χρησιμοποιήσετε την **ενσωματωμένη συνάρτηση help**. Είναι ιδιαίτερα χρήσιμη ειδικά όταν χρησιμοποιείτε την κονσόλα του διερμηνευτή. Για παράδειγμα, δώστε *help(print)* -θα εμφανίσει την τεκμηρίωση για τη συνάρτηση print που χρησιμοποιείται για την εκτύπωση στην οθόνη.

Πιέστε το q για να κλείσετε τη βοήθεια.Με τον ίδιο τρόπο, μπορείτε να βρείτε πληροφορίες σχετικά με σχεδόν οτιδήποτε στην Python. Χρησιμοποιήστε τη **help()** για να μάθετε περισσότερα για τη χρήση της ίδιας της help!

Σε περίπτωση που χρειάζεστε βοήθεια σχετικά με τελεστές (operators) όπως το return, τότε πρέπει να τους τοποθετήσετε ανάμεσα σε απλά εισαγωγικά π.χ. help('return') έτσι ώστε να μη μπερδευτεί η Python σχετικά με το τι προσπαθείτε να κάνετε.

Βασικές εισαγωγικές έννοιες στην Python

Κυριολεκτικές σταθερές (Literal Constants)

Ένα παράδειγμα κυριολεκτικής σταθεράς είναι ένας αριθμός όπως 4, 6.23, 5.25e-1 ή μια συμβολοσειρά όπως 'Αυτή είναι μια συμβολοσειρά' ή "Μια συμβολοσειρά!".

Αποκαλείται κυριολεκτική διότι κυριολεκτεί -η τιμή της ορίζεται κυριολεκτικά. Ο αριθμός 2 αναπαριστά τον εαυτό του και τίποτα άλλο – είναι σταθερά διότι η τιμή της δε μπορεί να αλλάξει. Άρα, όλα αυτά αναφέρονται ως κυριολεκτικές σταθερές.

Αριθμοί

Οι αριθμοί στην Python είναι τριών τύπων:

1) οι ακέραιοι (integers)
2) οι αριθμοί κινητής υποδιαστολής (floating point) και
3) οι μιγαδικοί αριθμοί (complex numbers)

- ✓ Ένα παράδειγμα ακέραιου είναι το 4 και είναι απλά ένας ακέραιος αριθμός.
- ✓ Παραδείγματα αριθμών κινητής υποδιαστολής (ή floats για συντομία) είναι οι 3.23 και 52.3E-4. Το σύμβολο E δηλώνει δύναμη του 10. Σε αυτή την περίπτωση, το 52.3E-4 σημαίνει 52.3 * 10-4.
- ✓ Παραδείγματα μιγαδικών αριθμών είναι οι (-3+j) και (3.3 - 2.6j).

Δεν υπάρχει ξεχωριστός 'long int' τύπος. Ο ακέραιος μπορεί να έχει κάθε τιμή.

Συμβολοσειρές (Strings)

Μια συμβολοσειρά είναι μια *ακολουθία από χαρακτήρες*. Οι συμβολοσειρές είναι βασικά ένα μάτσο λέξεις. Οι λέξεις μπορεί να είναι στην αγγλική ή σε κάθε γλώσσα που υποστηρίζεται από το πρότυπο Unicode, ουσιαστικά σε σχεδόν κάθε γλώσσα του κόσμου.

Δεν υπάρχουν "ASCII-only" συμβολοσειρές, διότι το πρότυπο Unicode αποτελεί υπερσύνολο του ASCII. Αν απαιτείται ένα ASCII-encoded byte-stream, τότε χρησιμοποιείστε str.encode("ascii"). Από προεπιλογή, όλες οι συμβολοσειρές είναι σε Unicode.

Μονά εισαγωγικά

Μπορείτε να ορίσετε συμβολοσειρές με μονά εισαγωγικά, για παράδειγμα 'Με λένε Τόνια και μαθαίνω να προγραμματίζω σε Python'. Η μορφοποίηση, π.χ κενοί χαρακτήρες (Whitespace) και στηλοθέτες (Tabs), παραμένει ως έχει.

Διπλά εισαγωγικά

Οι συμβολοσειρές με διπλά εισαγωγικά λειτουργούν ακριβώς όπως και με μονά εισαγωγικά. Για παράδειγμα, "Πώς σου φαίνεται η Python;"

Τριπλά εισαγωγικά

Μπορείτε να ορίσετε συμβολοσειρές πολλαπλών γραμμών με τριπλά εισαγωγικά - (""" ή '''). Χρησιμοποιήστε μονά εισαγωγικά και διπλά εισαγωγικά ελεύθερα μέσα σε τριπλά εισαγωγικά. Για παράδειγμα:

```
'''Μια συμβολοσειρά πολλών γραμμών. Αυτή είναι η πρώτη γραμμή.

Αυτή είναι η 2η γραμμή.

"Πώς σου φαίνεται η Python;" το ρώτησα.

Μου είπε: 'Πολύ ενδιαφέρουσα.'

'''
```

Ακολουθίες διαφυγής (Escape Sequences)

Υποθέστε ότι θα θέλατε μια συμβολοσειρά που να περιέχει ένα μονό εισαγωγικό ('), πώς θα το ορίσετε; Για παράδειγμα, η συμβολοσειρά είναι'Δε σ'αγνοώ !. Δεν μπορείτε να το ορίσετε ως 'Δε σ'αγνοώ !' διότι η Python θα μπερδευτεί με την αρχή και το τέλος της συμβολοσειράς. Γι'αυτό, πρέπει να ορίσετε ξεχωριστά ότι το μονό εισαγωγικό δε δηλώνει το τέλος της συμβολοσειράς. Η λύση είναι με τη χρήση του αποκαλούμενου χαρακτήρα διαφυγής. Ορίζετε το μονό εισαγωγικό ως \' -προσέξτε την *αριστερή πλάγια κάθετο*. Τώρα, μπορείτε να ορίσετε τη συμβολοσειρά ως

'Δε σ\'αγνοώ!'.

Ένας άλλος τρόπος ορισμού της συγκεκριμένης συμβολοσειράς είναι π.χ. "Δε σ' αγνοώ!", με τη χρήση διπλών εισαγωγικών. Παρομοίως, πρέπει να χρησιμοποιήσετε ένα χαρακτήρα διαφυγής για ένα διπλό εισαγωγικό εντός μιας συμβολοσειράς διπλών εισαγωγικών. Επίσης, πρέπει να υποδείξετε με χαρακτήρα διαφυγής την *αριστερή πλάγια κάθετο* \\.

Κι αν θέλετε να ορίσετε μια συμβολοσειρά δύο γραμμών; Ένας τρόπος είναι με τη χρήση τριπλών εισαγωγικών όπως δείξαμε προηγουμένως, ή με τη χρήση ενός χαρακτήρα διαφυγής - \n στην αρχή της νέας γραμμής. Για παράδειγμα :

```
Αυτή είναι η πρώτη γραμμή\nΚι αυτή είναι η δεύτερη γραμμή.
```

Ένας ακόμα χαρακτήρας διαφυγής που είναι χρήσιμο να γνωρίζετε είναι ο **στηλοθέτης (Tab) \t**. Υπάρχουν και πολλοί άλλοι χαρακτήρες διαφυγής αλλά εδώ ανέφερα τους χρησιμότερους. Αξίζει να σημειωθεί ότι εντός μιας συμβολοσειράς, μία αριστερή πλάγια κάθετος στο τέλος της γραμμής δηλώνει ότι η συμβολοσειρά συνεχίζεται στην επόμενη γραμμή, αλλά δεν προσθέτει νέα γραμμή. Για παράδειγμα η συμβολοσειρά:

```
"Αυτή είναι η πρώτη πρόταση.\

Κι αυτή είναι η δεύτερη πρόταση."
```

ισοδυναμεί με "Αυτή είναι η πρώτη πρόταση. Κι αυτή είναι η δεύτερη πρόταση.".

Ανεπεξέργαστες συμβολοσειρές (Raw Strings)

Αν πρέπει να ορίσετε κάποιες συμβολοσειρές και δε θέλετε ειδική επεξεργασία, π.χ. για το χειρισμό των χαρακτήρων διαφυγής, ορίστε μια ανεπεξέργαστη (raw) τιμή r ή R στη συμβολοσειρά. Ένα παράδειγμα γι' αυτό είναι:

```
r"It was made by \n".
```

Σ' αυτή τη συμβολοσειρά, δε θα τυπωθεί μια νέα γραμμή, παρότι υπάρχει το \n γιατί στις ανεπεξέργαστες συμβολοσειρές δεν αναλύονται οι χαρακτήρες διαφυγής.

Οι συμβολοσειρές είναι αμετάβλητες (Immutable)

Αυτό σημαίνει ότι αφού δημιουργήσατε μια συμβολοσειρά, δε μπορείτε να την αλλάξετε. Παρόλο που φαίνεται ως κάτι κακό, στην πραγματικότητα δεν είναι. Θα δούμε γιατί δεν είναι περιορισμός !

Συνένωση (Concatenation) κυριολεκτικών συμβολοσειρών

Αν τοποθετήσετε δίπλα δίπλα δύο κυριολεκτικές συμβολοσειρές, συνενώνονται αυτόματα από την Python. Για παράδειγμα, η συμβολοσειρά

```
'What\'s ' 'your name?'
```
μετατρέπεται αυτομάτως σε `"What's your name?"`

Σημείωση για προγραμματιστές C/C++

Δεν υπάρχει ξεχωριστός char τύπος δεδομένων στην Python.

Σημείωση για προγραμματιστές Perl/PHP

Θυμηθείτε ότι τα μονά και τα διπλά εισαγωγικά είναι το ίδιο - δε διαφέρουν σε καμία περίπτωση.

Σημείωση για χρήστες κανονικών εκφράσεων (Regular Expression).

Χρησιμοποιήστε πάντα ανεπεξέργαστες συμβολοσειρές για το χειρισμό κανονικών εκφράσεων. Διαφορετικά θα απαιτηθεί υπερβολική χρήση του χαρακτήρα διαφυγής. Για παράδειγμα, οι αναφορές μπορούν να ορισθούν ως *'\\1' ή r'\1'*.

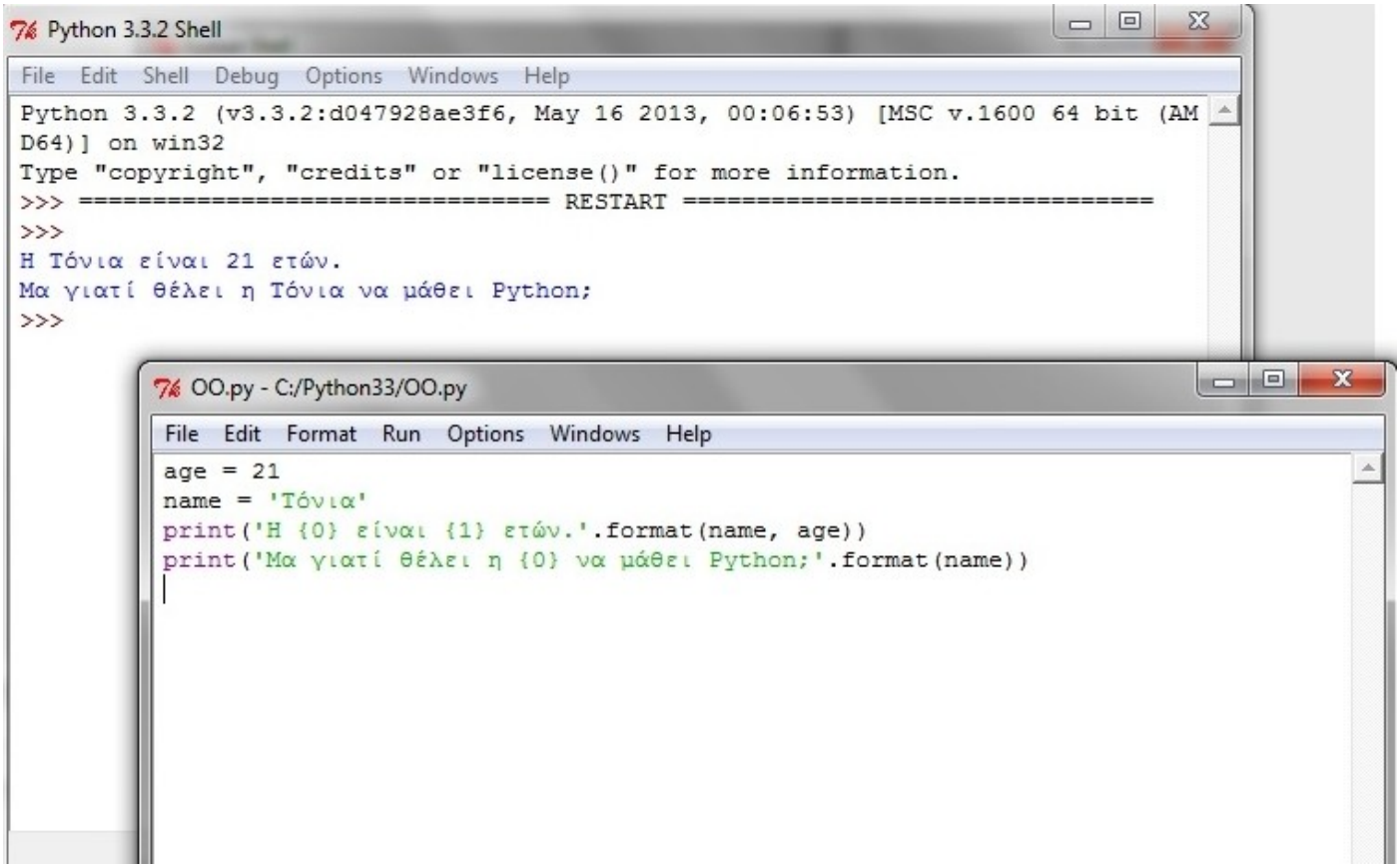
Η μέθοδος format

Μερικές φορές ίσως χρειαστεί να δημιουργήσουμε συμβολοσειρές από άλλες πληροφορίες. Εδώ είναι που χρησιμεύει η ***μέθοδος format().***

```
#!/usr/bin/python3.3

# Filename: example_format.py

age = 21

name = 'Τόνια'

print('Η {0} είναι {1} ετών.'.format(name, age))

print('Μα γιατί θέλει η {0} να μάθει Python;'.format(name))
```

Έξοδος →

```
Η Τόνια είναι 21 ετών.

Μα γιατί θέλει η Τόνια να μάθει Python;
```



Μια συμβολοσειρά μπορεί να χρησιμοποιεί ορισμένες προδιαγραφές (specifications) και κατά συνέπεια, μπορεί να κληθεί η μέθοδος format για να τις αντικαταστήσει με αντίστοιχα ορίσματα (arguments) της μεθόδου format.

- Παρατηρήστε την πρώτη δήλωση {0} που αντιστοιχεί στη μεταβλητή name η οποία είναι το πρώτο όρισμα στη μέθοδο μορφοποίησης. Παρομοίως, η δεύτερη δήλωση {1} αντιστοιχεί στη μεταβλητή age ως το δεύτερο όρισμα στη μέθοδο μορφοποίησης.

✓ Θα είχαμε το ίδιο αποτέλεσμα και με αλληλουχία συμβολοσειρών: *'Η ' + name + ' είναι ' +str(age) + ' ετών.'* , παρατηρήστε όμως την ασχήμια και την επιρρέπεια σε σφάλματα.

✓ Δεύτερον, η μετατροπή συμβολοσειράς θα είχε επιτευχθεί αυτόματα στη μέθοδο format αντί για τη ρητή μετατροπή εδώ.

✓ Τρίτον, με τη χρήση της μεθόδου format, μπορούμε να αλλάξουμε το μήνυμα, χωρίς να χρειάζεται να ασχοληθούμε με τις μεταβλητές που χρησιμοποιήθηκαν, και το αντίστροφο.

Μεταβλητές (Variables)

Η χρήση μόνο κυριολεκτικών σταθερών καταντά σύντομα βαρετή - χρειαζόμαστε κάποιο τρόπο αποθήκευσης και επεξεργασίας κάθε πληροφορίας. Εδώ είναι που εφαρμόζονται οι ***μεταβλητές***. Οι μεταβλητές είναι ακριβώς ότι το όνομα συνεπάγεται - οι τιμές τους ποικίλλουν, δηλαδή μπορείτε να αποθηκεύσετε οτιδήποτε με τη χρήση των μεταβλητών. Οι μεταβλητές είναι απλά τμήματα της μνήμης του Η/Υ σας όπου μπορείτε να αποθηκεύσετε πληροφορία. Αντίθετα με τις κυριολεκτικές σταθερές, χρειάζεστε κάποια μέθοδο (method) για να καλέσετε αυτές τις μεταβλητές, αλλά και να τους δώσετε ονόματα.

Ονοματοδοσία αναγνωριστικού

Οι μεταβλητές είναι παραδείγματα αναγνωριστικών. Τα ***αναγνωριστικά*** είναι δοθέντα ονόματα για να αναγνωρίσουν κάτι. Υπάρχουν ορισμένοι κανόνες που πρέπει να ακολουθήσετε για την ονοματοδοσία αναγνωριστικών:

- ⇨ Ο πρώτος χαρακτήρας του αναγνωριστικού πρέπει να είναι ένα γράμμα της αλφαβήτου (κεφαλαίο ASCII ή πεζό ASCII ή χαρακτήρας Unicode) ή μια κάτω παύλα ('_').
- ⇨ Το υπόλοιπο του ονόματος του αναγνωριστικού μπορεί να αποτελείται από γράμματα (κεφαλαία ASCII ή πεζά ASCII ή Unicode χαρακτήρες), κάτω παύλες ('_') ή αριθμούς (0-9).
- ⇨ Στο όνομα ενός αναγνωριστικού διακρίνονται τα πεζά από τα κεφαλαία (case-sensitive). Για παράδειγμα, το myname και το myName δεν είναι το ίδιο. Παρατηρήστε το πεζό n στο πρώτο παράδειγμα και το κεφαλαίο N στο δεύτερο.
- ⇨ Παραδείγματα ορθής ονοματοδοσίας αναγνωριστικού: i, __my_name, name_23, a1b2_c3 και random_utf8_characters_ვძჲ٣ㄆㄘㄩ ڨ ៗ𝓃𝓊ςεζ ढ ᱶᱲᱥᱞᱝᱛᱜᱚ.
- ⇨ Παραδείγματα εσφαλμένης ονοματοδοσίας αναγνωριστικού: 2things, this is spaced out,my-name, και "this_is_in_quotes".
- ⇨ Μπορείτε να χρησιμοποιείτε ελληνικές λέξεις ως ονόματα αναγνωριστικών, π.χ. α, το_όνομά_μου.

Τύποι δεδομένων

Οι μεταβλητές μπορούν να διατηρούν τιμές διαφορετικών τύπων που ονομάζονται τύποι δεδομένων. Οι βασικοί τύποι είναι αριθμοί και συμβολοσειρές, όπως ήδη αναφέραμε. Στη συνέχεια, θα δούμε πώς μπορούμε να δημιουργήσουμε δικούς μας τύπους χρησιμοποιώντας → Κλάσεις.

Αντικείμενα

Πρέπει να θυμόμαστε ότι, η Python αντιλαμβάνεται οτιδήποτε χρησιμοποιείται σε ένα πρόγραμμα ως ένα αντικείμενο, με μια γενική έννοια της λέξης. Αντί να πούμε 'το κάτι', λέμε 'το αντικείμενο'.

Σημείωση για χρήστες αντικειμενοστρεφούς προγραμματισμού

Η ***Python είναι έντονα αντικειμενοστρεφής*** με την έννοια ότι οτιδήποτε είναι ένα αντικείμενο συμπεριλαμβάνοντας αριθμούς, συμβολοσειρές και συναρτήσεις.

Πώς να γράφετε προγράμματα Python

Η τυπική διαδικασία εφεξής για την αποθήκευση και εκτέλεση ενός προγράμματος Python είναι η ακόλουθη:

1. Ανοίξτε τον επεξεργαστή κώδικα της προτίμισής σας.

2. Γράψτε τον κώδικα του προγράμματος που δίνεται στο παράδειγμα.

3. Αποθηκεύστε το ως αρχείο με το όνομα που αναφέρεται στο σχόλιο. Ακολουθώ τη σύμβαση αποθήκευσης όλων των προγραμμάτων Python με την κατάληξη .py.

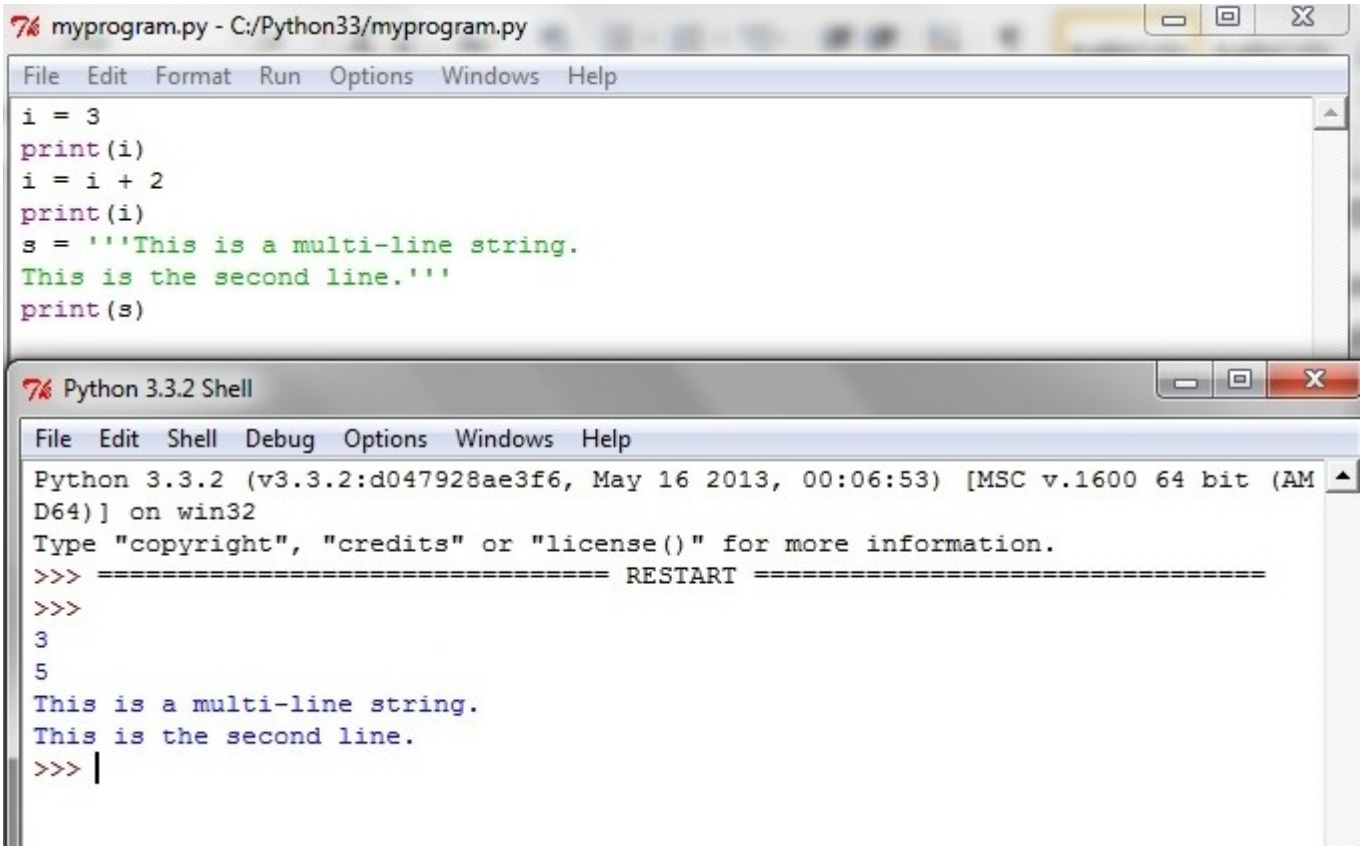
4. Εκτελέστε το διερμηνευτή με την εντολή python var.py ή απλούστερα χρησιμοποιήστε το IDLE για την εκτέλεση των προγραμμάτων. Μπορείτε να κάνετε επίσης τα αρχεία εκτελέσιμα όπως αναφέρθηκε παραπάνω.

Στη συνέχεια, θα δούμε τη χρήση μεταβλητών μαζί με κυριολεκτικές σταθερές μέσω ενός απλού παραδείγματος.

```
# Filename : example.py

i = 3

print(i)

i = i + 2

print(i)

s = '''This is a multi-line string.

This is the second line.'''

print(s)
```

Έξοδος

```
3

5

This is a multi-line
string.

This is the second
line.
```



Πώς λειτουργεί το παραπάνω πρόγραμμα example.py :

Να πώς λειτουργεί το πρόγραμμα. Πρώτα, εκχωρούμε την τιμή της κυριολεκτικής σταθεράς 3 στη μεταβλητή i με τη χρήση του τελεστή εκχώρησης (=). Αυτή η γραμμή ονομάζεται εντολή (statement) διότι δηλώνει ότι κάτι πρέπει να γίνει και στην περίπτωσή μας, συνδέουμε το όνομα της μεταβλητής i με την τιμή 3. Στη συνέχεια, εκτυπώνουμε την τιμή της μεταβλητής i με τη χρήση της εντολής print η οποία, φυσικά, απλά εκτυπώνει την τιμή της μεταβλητής στην οθόνη.

Μετά προσθέτουμε 2 στην αποθηκευμένη τιμή του i και αποθηκεύεται ενημερωμένη εκ νέου. Κατόπιν την εκτυπώνουμε και όπως περιμένουμε, μας επιστρέφει την τιμή 5. Παρομοίως, εκχωρούμε την κυριολεκτική σταθερά στη μεταβλητή s και εκτυπώνουμε στην οθόνη.

Σημείωση για τους προγραμματιστές στατικών γλωσσών: Οι μεταβλητές χρησιμοποιούνται μόνο με την εκχώρηση σε αυτές μιας τιμής. Δεν απαιτείται/χρησιμοποιείται κάποια δήλωση (declaration) ή ο ορισμός τύπου δεδομένων (data type definition).

Λογικές και φυσικές γραμμές πηγαίου κώδικα

Μια φυσική γραμμή είναι αυτό που εσείς βλέπετε όταν γράφετε το πρόγραμμα. Μια λογική γραμμή είναι αυτό που βλέπει η Python ως μια ενιαία εντολή. Η Python υποθέτει σιωπηρά ότι κάθε φυσική γραμμή αντιστοιχεί σε μια λογική γραμμή.

Ένα παράδειγμα λογικής γραμμής είναι μια εντολή όπως print('Hello World') -αν αυτή ήταν σε μια γραμμή από μόνη της (όπως τη βλέπετε στον επεξεργαστή κώδικα), τότε αυτή αντιστοιχεί επίσης σε μια φυσική γραμμή.

Έμμεσα, η Python ενθαρρύνει τη χρήση μιας και μόνο εντολής ανά γραμμή ώστε ο κώδικας να είναι ευανάγνωστος.

Εάν θέλετε να καθορίσετε περισσότερες από μία λογικές γραμμές σε μια μόνο φυσική γραμμή, τότε πρέπει να το ορίσετε ρητά με τη χρήση του ερωτηματικού (semicolon) (;) και δείχνει το τέλος μιας λογικής γραμμής/εντολής.

Ακολουθεί παράδειγμα :

```
i = 5
print(i)
```

Είναι το ίδιο με

```
i = 5;
print(i);
```

και το ίδιο μπορεί να γραφτεί ως…

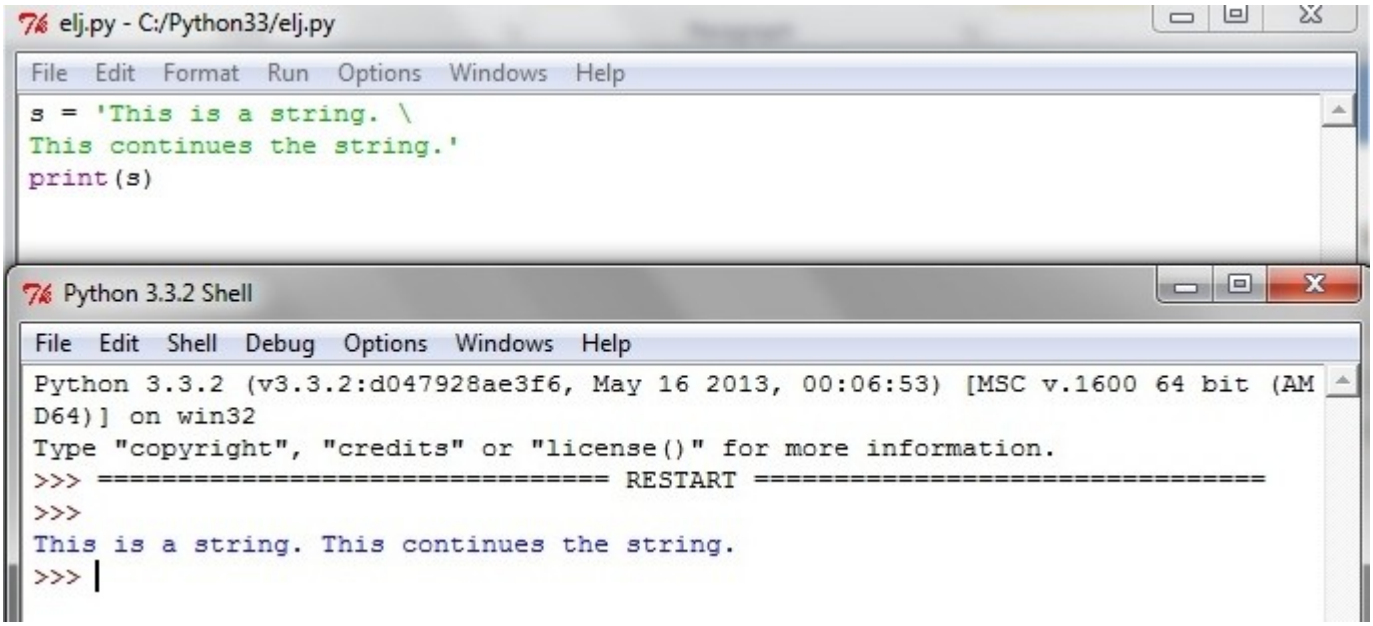
```
i = 5; print(i);
```

ή και

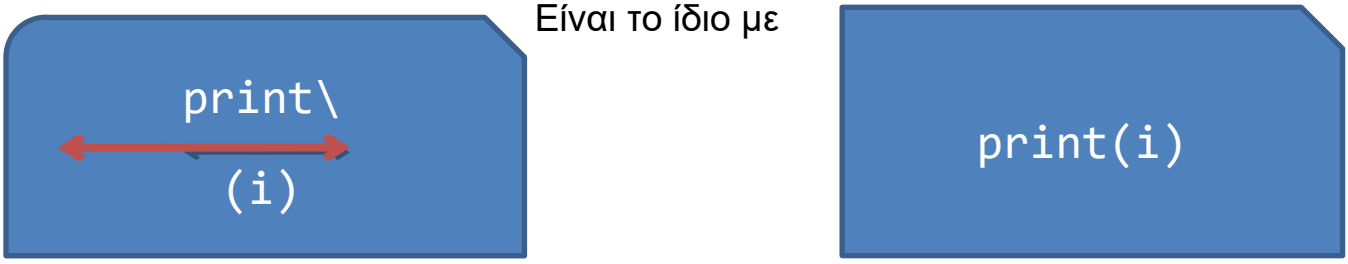
```
i = 5; print(i)
```

Ωστόσο, συνηθίζεται να προγραμματίζετε γράφοντας μία και μόνο λογική γραμμή σε μια ενιαία φυσική γραμμή μόνο. Χρησιμοποιήστε περισσότερες φυσικές γραμμές για μία μόνο λογική γραμμή, αν αυτή είναι όντως μεγάλη. Η ιδέα είναι να αποφευχθεί το ερωτηματικό, όσο το δυνατόν περισσότερο, ώστε ο κώδικας να είναι ευανάγνωστος. Στην πραγματικότητα, οι έμπειροι προγραμματιστές της Python δεν χρησιμοποιούν ποτέ ερωτηματικό σε ένα πρόγραμμα Python.

- Ακολουθεί ένα παράδειγμα σύνταξης μιας λογικής γραμμής που εκτείνεται σε πολλές φυσικές γραμμές. Αυτό αναφέρεται ως ***explicit line joining***.



Ομοίως…



Μερικές φορές, υπάρχει μια σιωπηρή παραδοχή όπου δε χρειάζεται να χρησιμοποιήσετε μια αριστερή πλάγια κάθετο. Αυτή είναι η περίπτωση όπου η λογική γραμμή χρησιμοποιεί παρενθέσεις (), αγκύλες [] ή άγκιστρα {}. Αυτό αναφέρεται ως ***implicit line joining***.

Εσοχή κώδικα (Indentation)

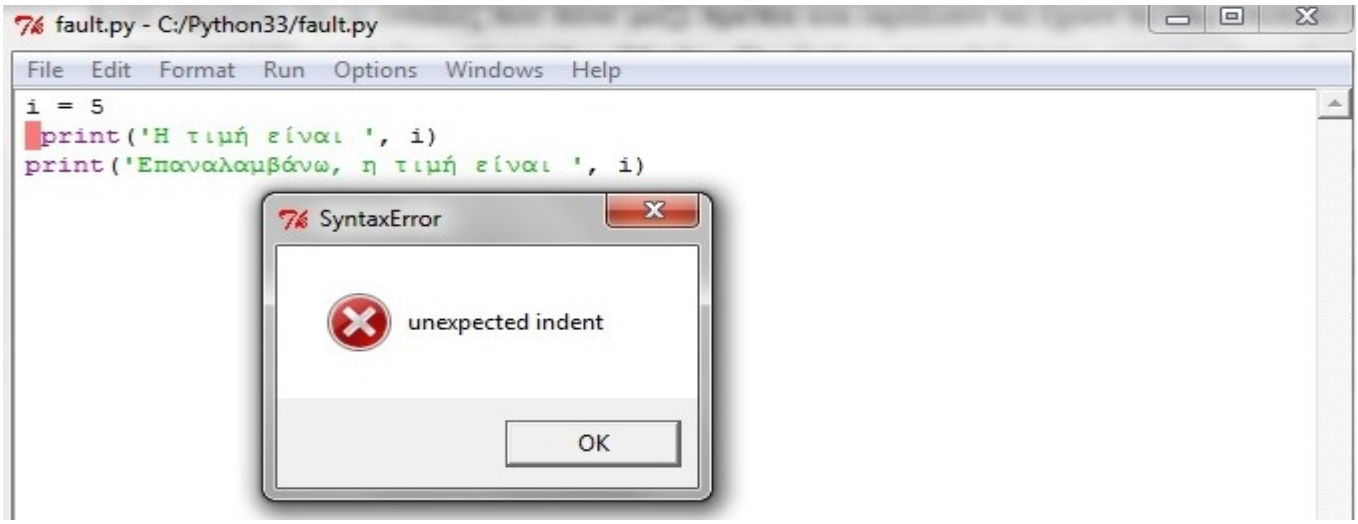
Ο κενός χώρος είναι σημαντικός στην Python. Πράγματι, ***ο κενός χαρακτήρας στην αρχή της γραμμής είναι σημαντικός. Αυτό αποκαλείται εσοχή κώδικα***. Οι αρχικοί κενοί χαρακτήρες (κενά και στηλοθέτες) στην αρχή της λογικής γραμμής καθορίζουν το επίπεδο εσοχής της λογικής γραμμής, και αυτό με τη σειρά του προσδιορίζει την ομαδοποίηση των εντολών.

Αυτό σημαίνει ότι οι εντολές που πάνε μαζί πρέπει και οφείλουν να έχουν το ίδιο επίπεδο εσοχής. Κάθε τέτοια ομάδα εντολών, καλείται **πλοκάδα (block)**.

ΠΡΟΣΟΧΗ!

Είναι πολύ σημαντικό να θυμόμαστε ότι οι λάθος εσοχές μπορεί να προκαλέσουν σφάλματα

Ακολουθεί παράδειγμα :



Σφάλμα! Προσέξτε ένα διάστημα στην αρχή της γραμμής

Προσέξτε τον κενό χαρακτήρα στην αρχή της δεύτερης γραμμής. Το σφάλμα που υποδεικνύει η Python μας λέει ότι η σύνταξη του προγράμματος είναι άκυρη, δηλαδή, ο κώδικας του προγράμματος δε γράφτηκε σωστά. Που σημαίνει για σας ότι δε μπορείτε αυθαίρετα να ξεκινάτε νέες πλοκάδες εντολών (εκτός φυσικά από την προεπιλεγμένη κύρια πλοκάδα που χρησιμοποιούσατε συνέχεια).

Προσοχή : Στην Python μετράνε οι χαρακτήρες διαστήματος!! Αν λοιπόν συναντήσετε ποτέ το ακόλουθο σφάλμα :

```
IndentationError : expected an indented block
```

Σημαίνει ότι ξεχάσατε να τοποθετήσετε τους χαρακτήρες διαστήματος.

Χρήση της εσοχής

Να μην αναμιγνύετε στηλοθέτες και κενά στις εσοχές διότι θα προκληθεί ασυμβατότητα μεταξύ διαφορετικών λειτουργικών. Σας συνιστώ τη χρήση είτε ενός μονού στηλοθέτη (1 tab) ή τεσσάρων κενών (4 spaces) για κάθε επίπεδο εσοχής. Επιλέξτε ένα από τα δύο στυλ εσοχών. Και σημαντικότερο είναι να επιλέξετε ένα και να το χρησιμοποιείτε σταθερά , δηλαδή συνηθίστε αυτό το στυλ μόνο.

Σημείωση → Η Python θα έχει πάντα εσοχή για πλοκάδες και ποτέ αγκύλες (braces).

Τελεστές και εκφράσεις

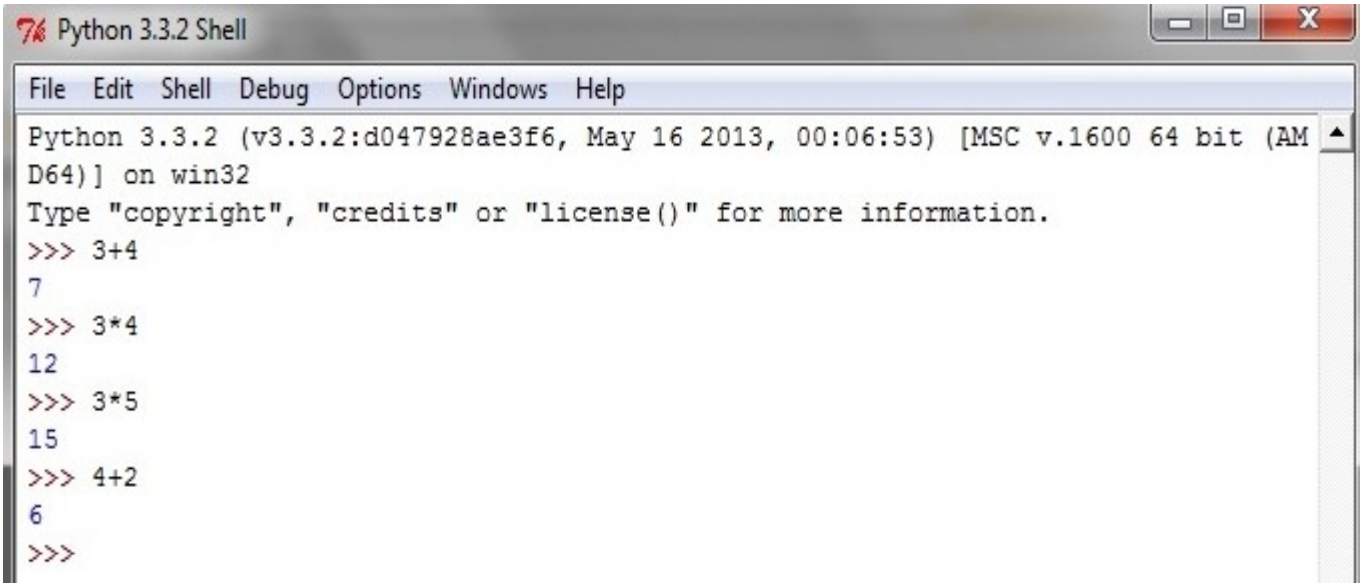
Οι περισσότερες εντολές που γράφετε θα περιέχουν **εκφράσεις (expressions)**. Ένα απλό παράδειγμα μίας έκφρασης είναι 3 + 4. Μία έκφραση μπορεί να διαχωριστεί σε τελεστές (operators) και τελεστέους (operands).

Οι τελεστές είναι λειτουργίες που κάνουν κάτι και μπορούν να αναπαρασταθούν με σύμβολα όπως το + ή με ειδικές λέξεις-κλειδιά. Οι τελεστές απαιτούν κάποια δεδομένα πάνω στα οποία θα λειτουργήσουν και αυτά τα δεδομένα ονομάζονται τελεστέοι. Στη συγκεκριμένη περίπτωση, οι τελεστέοι είναι το 3 και το 4.

Τελεστές

Ας δούμε τους τελεστές και τη χρήση τους:

Μπορείτε να βρείτε την τιμή των εκφράσεων που δίνονται στα παραδείγματα χρησιμοποιώντας το διερμηνευτή διαδραστικά. Για παράδειγμα, για τον έλεγχο της έκφρασης 3 + 4, χρησιμοποιήστε την κονσόλα του διαδραστικού διερμηνευτή της Python.



Στη συνέχεια παραθέτω έναν ολοκληρωμένο πίνακα των τελεστών με εξήγηση του καθένα και παραδείγματα.

Τελεστής	Όνομα	Εξήγηση	Παραδείγματα
+	Συν	Προσθέτει δύο αντικείμενα.	Το 3 + 2 δίνει 5. Το 'a' + 'b' δίνει 'ab'.
-	Μείον	Είτε δίνει έναν αρνητικό αριθμό, ή αφαιρεί έναν αριθμό από έναν άλλο.	Το -2.5 δίνει έναν αρνητικό αριθμό. Το 50 - 27 δίνει 23.
*	Επί	Δίνει το γινόμενο δύο αριθμών ή μια συμβολοσειρά (string) επαναλαμβανόμενη τόσες φορές.	Το 3 * 3 δίνει 9. Το 'la' * 3 δίνει 'lalala'.
**	Δύναμη	Επιστρέφει το x υψωμένο στη δύναμη y.	Το 3 ** 2 δίνει 9 (δηλαδή 3 * 3).
/	Δια	Διαιρεί το x με το y.	Το 12/4 δίνει 3
//	Διαίρεση στρογγυλοποιημένη προς τα κάτω (Floor Division)	Επιστρέφει τον κοντινότερο (προς τα κάτω) ακέραιο στο πηλίκο.	Το 4 // 3 δίνει 1.
%	Υπόλοιπο	Επιστρέφει το υπόλοιπο της διαίρεσης.	Το 8 % 3 δίνει 2. Το -25.5 % 2.25 δίνει 1.5.
<<	Αριστερή μετάθεση	Μεταθέτει τα δυαδικά ψηφία (bits) του αριθμού προς τα αριστερά κατά το πλήθος των θέσεων που καθορίστηκε. (Κάθε αριθμός αναπαρίσταται στη μνήμη με δυαδικά ψηφία (bits, binary digits) -δηλαδή με 0 και 1).	Το 2 << 2 δίνει 8. Το 2 αναπαριστάται ως σε bits ως 10. Η μετάθεση προς τα αριστερά κατά 2 bits μας δίνει 1000 που παριστάνει το δεκαδικό 8.
>>	Δεξιά μετάθεση	Μεταθέτει τα bits του αριθμού προς τα δεξιά κατά το πλήθος των θέσεων που καθορίστηκε.	Το 11 >> 1 δίνει 5. Το 11 αναπαριστάται σε bits ως 1011 που όταν μετατεθούν δεξιά κατά 1 bit μας δίνει 101 το οποίο είναι το δεκαδικό 5.

&	Δυαδικό ΚΑΙ	Δυαδικό ΚΑΙ των αριθμών.	Το 5 & 3 δίνει 1.
|	Δυαδικό Ή	Δυαδικό Ή των αριθμών	Το 5 | 3 δίνει 7.
^	Δυαδικό αποκλειστικό ‘Η	Δυαδικό αποκλειστικό Ή των αριθμών	Το 5 ^ 3 δίνει 6.
~	Δυαδική αντιστροφή	Το δυαδικό αντίστροφο του x είναι -(x+1).	Το ~5 δίνει -6.
<	Μικρότερο από	Επιστρέφει το αν το x είναι μικρότερο από το y. Όλοι οι τελεστές σύγκρισης επιστρέφουν True (Αληθής) ή False (Ψευδής). Σημειώστε οτι τα ονόματα αυτά ξεκινούν με κεφαλαίο.	Το 5 < 3 δίνει False και το 3 < 5 δίνει True. Οι συγκρίσεις μπορούν να συνδυαστούν αλυσιδωτά κατά βούληση: Το 3 < 5 < 7 δίνει True.
>	Μεγαλύτερο από	Επιστρέφει το αν το x είναι μεγαλύτερο από το y.	Το 5 > 3 επιστρέφει True. Αν και οι δύο τελεστέοι είναι αριθμοί, πρώτα μετατρέπονται σε έναν κοινό τύπο. Αλλιώς, επιστρέφει πάντα False.
<=	Μικρότερο ή ίσο	Επιστρέφει το αν το x είναι μικρότερο από ή ίσο με το y.	Το x = 3; y = 6; x <= y επιστρέφει True.
>=	Μεγαλύτερο ή ίσο	Επιστρέφει το αν το x είναι μεγαλύτερο από ή ίσο με το y.	Το x = 4; y = 3; x >= 3 επιστρέφει True.
==	Ίσο	Συγκρίνει αν τα αντικείμενα είναι ίσα.	Το x = 2; y = 2; x == y επιστέφει True. Το x = 'str'; y = 'stR'; x == y επιστρέφει False. Το x = 'str'; y = 'str'; x == y επιστρέφει True.
!=	Διαφορετικό	Συγκρίνει αν τα αντικείμενα ΔΕΝ είναι ίσα.	Το x = 2; y = 3; x != y επιστρέφει True.
not	Λογικό ΟΧΙ	Αν το x είναι True,επιστρέφει False. Αν το x είναι False, επιστρέφει True	x = True; not x επιστρέφει False.
and	Λογικό ΚΑΙ	Το x and y επιστρέφει False αν το x είναι False, αλλιώς επιστρέφει υπολογίζει και	x = False; y = True; x and y επιστρέφει False

		επιστρέφει την τιμή του y.	αφού το x είναι False. Σε αυτή την περίπτωση, η Python δε θα ελέγξει την τιμή του y αφού γνωρίζει οτι η αριστερή πλευρά της έκφρασης 'and' είναι False που υποδηλώνει οτι ολόκληρη η έκφραση θα είναι False ανεξάρτητα από τις άλλες τιμές. Αυτή η τεχνική αποκαλείται **short-circuit evaluation.**
or	Λογικό Ή	Αν το x είναι True, επιστρέφει True, αλλιώς υπολογίζει και επιστρέφει την τιμή του y.	Το x = True; y = False; x or y επιστρέφει True. Η Short-circuit evaluation εφαρμόζεται και εδώ.

Συντόμευση για μαθηματικές πράξεις και την ανάθεση

Είναι συνήθες φαινόμενο η εκτέλεση μίας μαθηματικής πράξης στα περιεχόμενα μίας μεταβλητής και στη συνέχεια η ανάθεση του αποτελέσματος πίσω στη μεταβλητή, γι' αυτό υπάρχει μία συντόμευση για αυτού του είδους τις εκφράσεις.

Μπορείς να γράψεις το:

```
a = 2; a = a * 3        ως      a = 2; a *= 3
```

Παρατήρηση → ***η δήλωση μεταβλητή = μεταβλητή τέλεση έκφραση γίνεται μεταβλητή τέλεση = έκφραση***.

Σειρά υπολογισμού (προτεραιότητα τελεστών)

Ας υποθέσουμε ότι έχουμε μία έκφραση πχ 3 + 2 * 4. Ποιο γίνεται πρώτα, η πρόσθεση ή ο πολλαπλασιασμός; **Τα μαθηματικά μας λένε ότι πρώτα πρέπει να γίνει ο πολλαπλασιασμός. Αυτό σημαίνει ότι ο τελεστής του πολλαπλασιασμού έχει υψηλότερη προτεραιότητα από τον τελεστή της πρόσθεσης.**

Ο πίνακας που ακολουθεί μας δίνει τον πίνακα προτεραιοτήτων για την Python, από τη χαμηλότερη προς την υψηλότερη προτεραιότητα. Αυτό σημαίνει ότι σε μία δεδομένη έκφραση, η Python θα υπολογίσει τους τελεστές και τις εκφράσεις που βρίσκονται χαμηλότερα στον πίνακα πριν από αυτούς που βρίσκονται ψηλότερα.

Ο παρακάτω πίνακας, αντιγραμμένος από το εγχειρίδιο αναφοράς της Python, παρέχεται χάριν πληρότητας. Είναι πολύ καλύτερο να χρησιμοποιείτε παρενθέσεις για να ομαδοποιείτε σωστά τελεστές και τελεστέους ώστε να ορίσετε ξεκάθαρα την προτεραιότητα. Έτσι το πρόγραμμα γίνεται πιο ευανάγνωστο.

Τελεστής	Περιγραφή
Lambda	Έκφραση Lambda
Or	Λογικό Ή
And	Λογικό ΚΑΙ
not x	Λογικό ΟΧΙ
in, not in	Έλεγχοι συμμετοχής
is, is not	Έλεγχοι ταυτότητας
<, <=, >, >=, !=,==	Συγκρίσεις
|	Δυαδικό Ή
^	Δυαδικό αποκλειστικό Ή
&	Δυαδικό ΚΑΙ
<<, >>	Μεταθέσεις
+, -	Πρόσθεση και αφαίρεση
***, /, //, %**	Πολλαπλασιασμός, Διαίρεση, Διαίρεση στρογγυλοποιημένη προς τα κάτω, Υπόλοιπο
+x, -x	Θετικό, Αρνητικό
~x	Δυαδικό ΟΧΙ
**	Ύψωση σε δύναμη
x.attribute	Αναφορά σε χαρακτηριστικό
x[index]	Subscription
x[index1:index2]	Κομμάτιασμα (Slicing)
f(arguments ...)	Κλήση συνάρτησης
(expressions, ...)	Συνένωση ή εμφάνιση πλειάδας
[expressions, ...]	Προβολή λίστας
{key:datum, ...}	Προβολή λεξικού

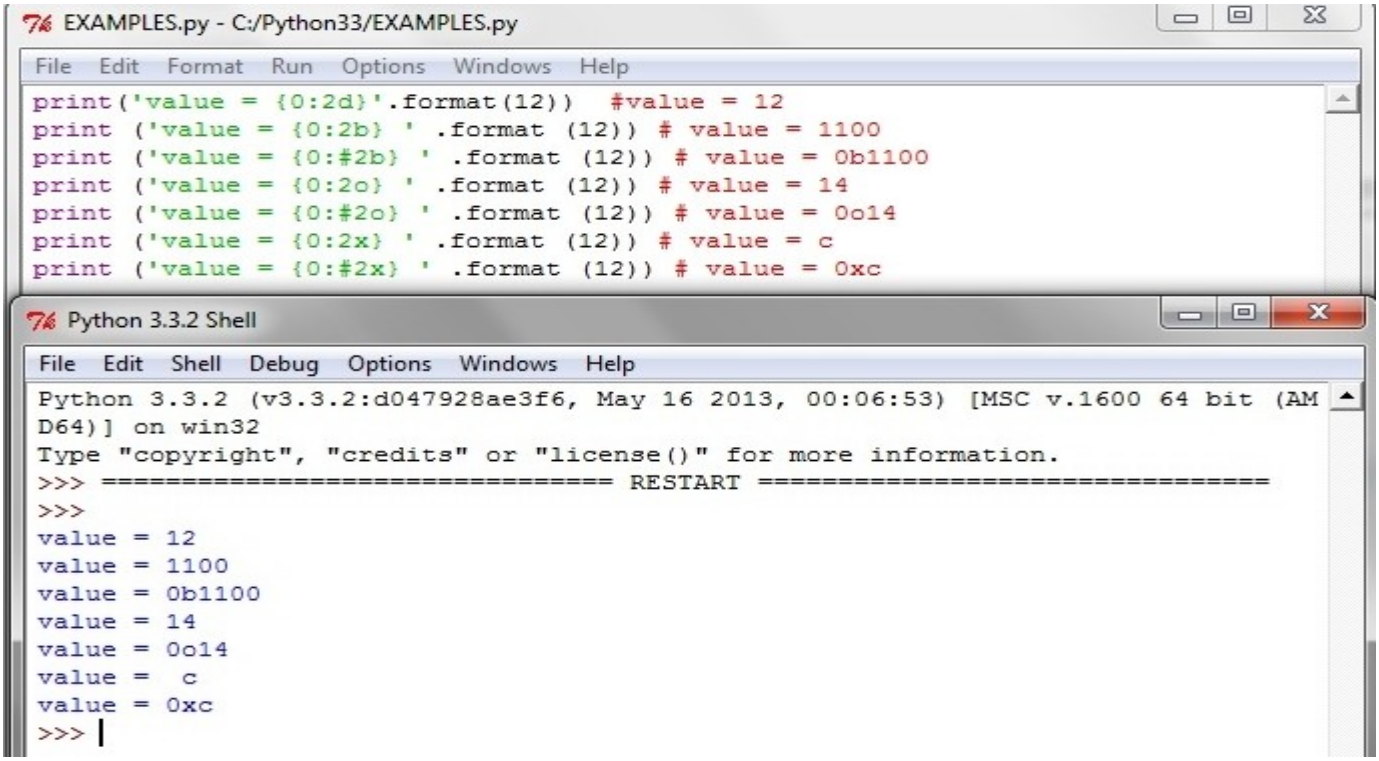
Σημείωση → Τελεστές με την ίδια προτεραιότητα βρίσκονται στην ίδια γραμμή στον παραπάνω πίνακα. Για παράδειγμα, το + και το - έχουν την ίδια προτεραιότητα.

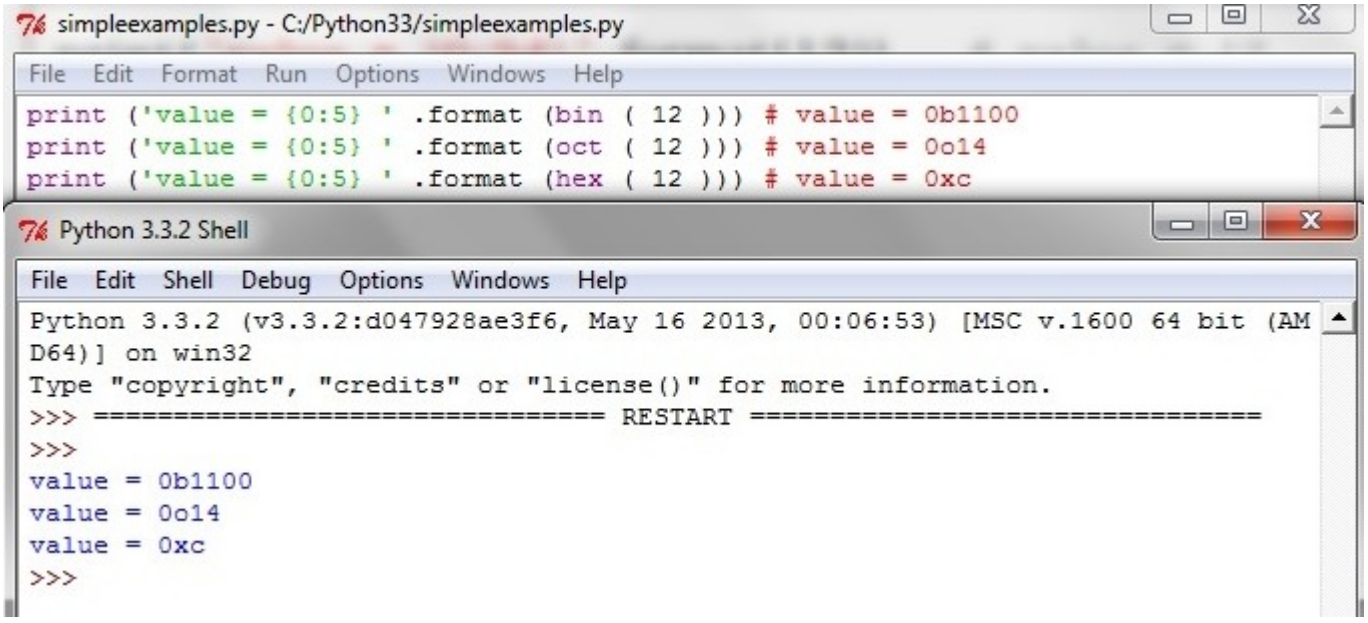
Ακέραιοι

Μπορούμε να τυπώνουμε ακεραίους σε διάφορα συστήματα, με την χρήση των τελεστών :

1) d : Για δεκαδικούς αριθμούς
2) b : Για δυαδικούς αριθμούς (ή εναλλακτικά bin())
3) o : Για οκταδικούς αριθμούς (ή εναλλακτικά oct())
4) x : Για δεκαεξαδικούς αριθμούς (ή εναλλακτικά hex ())

Προσθέτωντας τον χαρακτήρα της δίεσης , εκτυπώνεται επιπλέον πληροφορία που δείχνει σε ποιο σύστημα ανήκει ο εκτυπωμένος αριθμός.





Αριθμοί κινητής υποδιαστολής

Για την μορφοποίηση αριθμών κινητής υποδιαστολής, χρησιμοποιούμε τον προσδιοριστή (specifier) f. Μπορούμε να πούμε πόσα ψηφία επιθυμούμε να εκτυπωθούν από το δεκαδικό μέρος. Επίσης, με την χρήση του τελεστή +, φαίνεται το πρόσημο και στην περίπτωση θετικών αριθμών.

Ακολουθούν παραδείγματα :

someexamples.py - C:/Python33/someexamples.py

File Edit Format Run Options Windows Help

```
print('x = {0:5.3f}'.format(1234.567))      # x = 1234.567
print('x = {0:12.6f}'.format(1234.567))   # x = 1234.567000
print('x = {0:-12.9f}'.format(1234.567))    # x = 1234.567000
print('x = {0:+12.9f}'.format(1234.567))    # x = +1234.567000
print('x = {0:+12.9f}'.format(-1234.567))   # x = -1234.567000
print('x = {0:012.3f}'.format(1234.567))    # x = 00001234.567
print()
print('x = {0:1.5e}'.format(1234.567))      # x = 1.234567e+03
print('x = {0:1.5e}'.format(0.000001234))   # x = 1.23400e-06
print('x = {0:1.5g}'.format(0.000001234))   # x = 1.234e-06
print()
print('x = {0:2.3%}'.format(0.337))         # x = 33.700%
print()
```

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
x = 1234.567
x =  1234.567000
x = 1234.567000000
x = +1234.567000000
x = -1234.567000000
x = 00001234.567

x = 1.23457e+03
x = 1.23400e-06
x = 1.234e-06

x = 33.700%

>>>
```

Αλλαγή της σειράς υπολογισμού

Για να κάνουμε τις εκφράσεις πιο ευανάγνωστες, μπορούμε να χρησιμοποιήσουμε παρενθέσεις. Για παράδειγμα, το 2 + (4 * 3) είναι σίγουρα πιο εύκολο να το καταλάβουμε από το 2 + 4 * 3 το οποίο απαιτεί τη γνώση των προτεραιοτήτων των τελεστών. Όπως και όλα τα άλλα, οι παρενθέσεις θα πρέπει να χρησιμοποιούνται με σύνεση (μην το παρακάνετε) και να μην είναι άχρηστες (όπως π.χ. στο 2 + (4 + 3)).

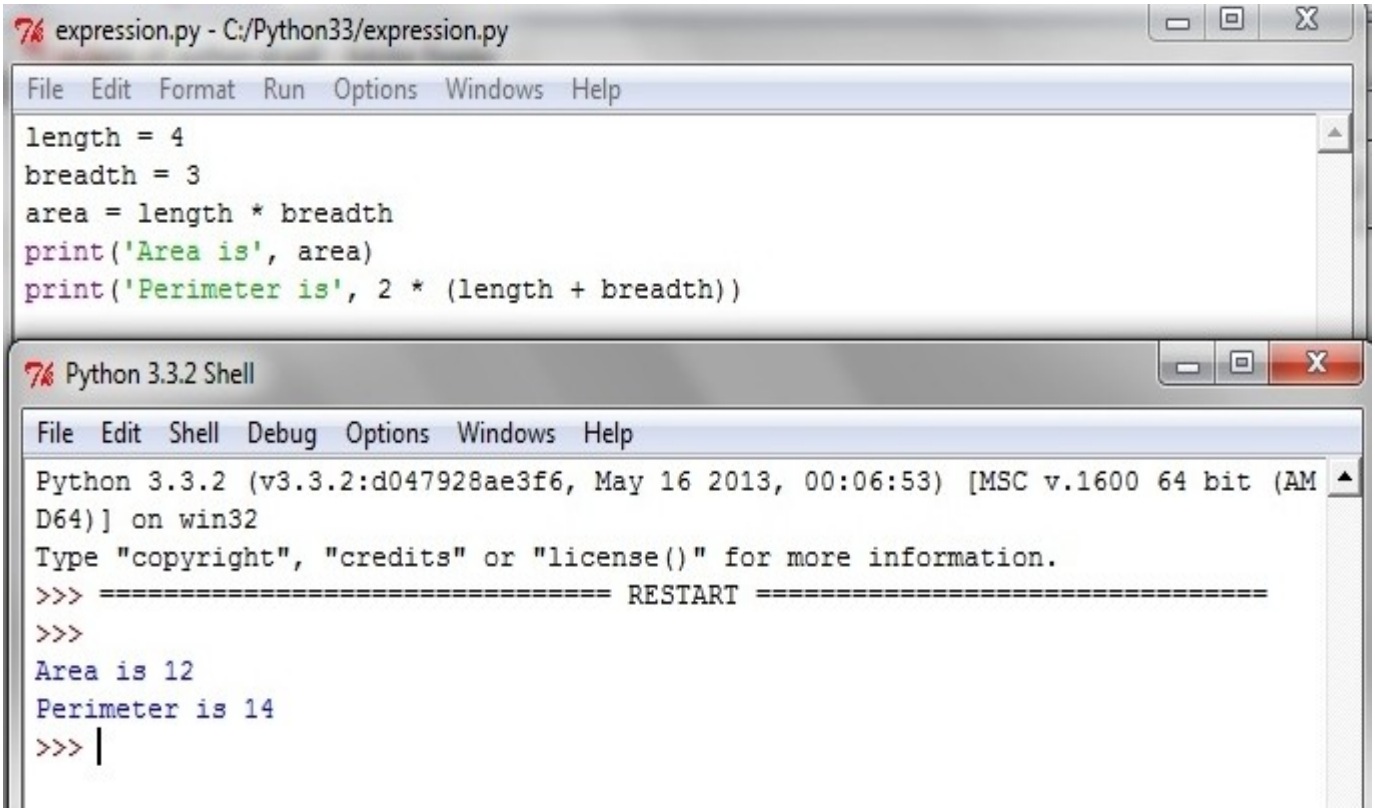
Υπάρχει ένα επιπλέον πλεονέκτημα στη χρήση παρενθέσεων -μας βοηθά να αλλάξουμε τη σειρά υπολογισμού. Για παράδειγμα, αν θέλετε να γίνει η πρόσθεση πριν τον πολλαπλασιασμό σε μία έκφραση, τότε μπορείτε να γράψετε κάτι όπως (2 + 3) * 4.

Συσχέτιση (Associativity)

Οι τελεστές συνήθως συσχετίζονται από τα αριστερά προς τα δεξιά δηλαδή οι τελεστές με την ίδια προτεραιότητα υπολογίζονται από τα αριστερά προς τα δεξιά. Για παράδειγμα, το 2 + 3 + 4 υπολογίζεται ως (2 + 3) + 4. Μερικοί τελεστές όπως οι τελεστές ανάθεσης έχουν συσχέτιση από τα δεξιά προς τα αριστερά δηλ. το a = b = c αντιμετωπίζεται ως a = (b = c).

Εκφράσεις (Expressions)

Ακολουθεί ένα παράδειγμα :



Πώς δουλεύει το παραπάνω πρόγραμμα :

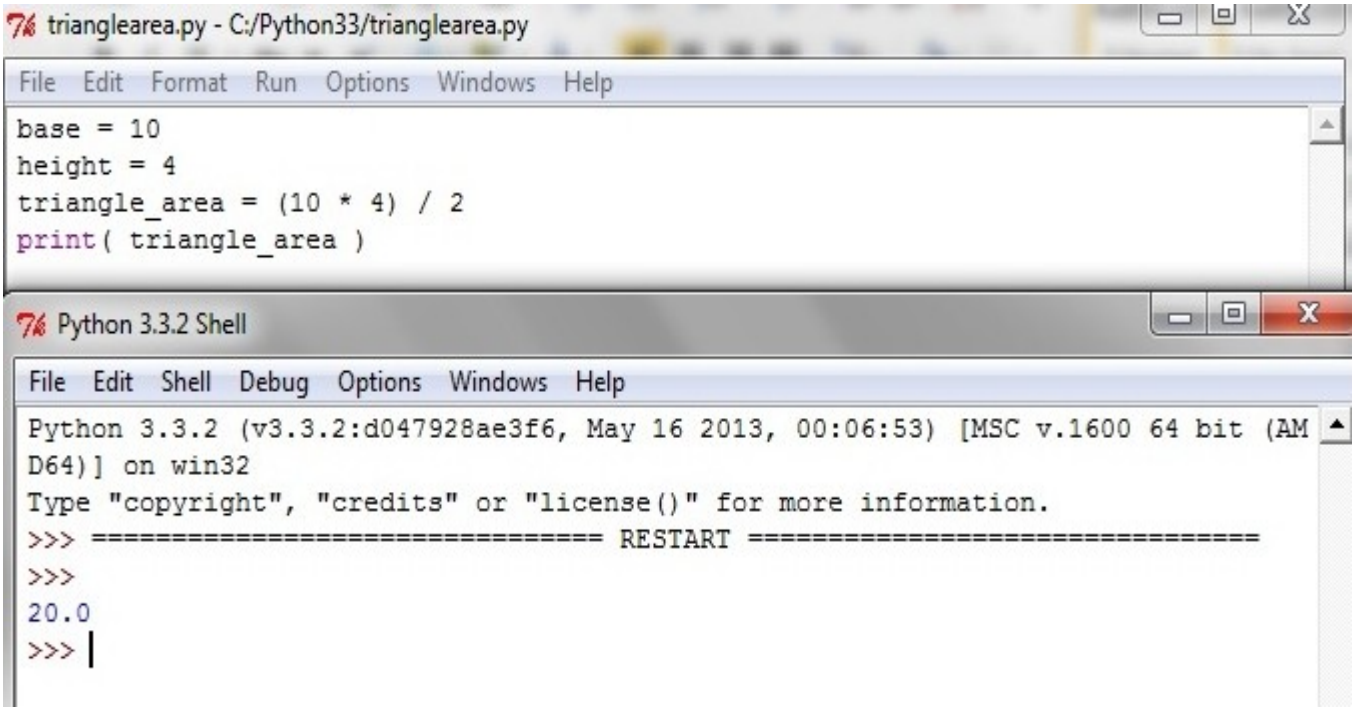
Το μήκος (length) και το πλάτος (breadth) του ορθογωνίου αποθηκεύονται σε μεταβλητές με το ίδιο όνομα. Χρησιμοποιούμε αυτές τις μεταβλητές για να υπολογίσουμε το εμβαδόν και την περίμετρο του ορθογωνίου με τη βοήθεια εκφράσεων. Αποθηκεύουμε το αποτέλεσμα της έκφρασης length * breadth στη μεταβλητή (variable) area και έπειτα το τυπώνουμε χρησιμοποιώντας τη συνάρτηση print. Στη δεύτερη περίπτωση, χρησιμοποιούμε απ' ευθείας την τιμή της έκφρασης 2 * (length + breadth) στη συνάρτηση print.

Επίσης, παρατηρήστε τον τρόπο με τον οποίο η Python "τυπώνει όμορφα" τα αποτελέσματα. Παρ' όλο που δεν έχουμε καθορίσει ένα κενό ανάμεσα στο 'Area is' και στη μεταβλητή area, η Python το τοποθετεί για εμάς έτσι ώστε να πάρουμε μία πιο καθαρή όμορφη έξοδο και το πρόγραμμα είναι πολύ πιο ευανάγνωστο με αυτό τον τρόπο (αφού δε χρειάζεται να ανησυχούμε για τα κενά στις συμβολοσειρές που χρησιμοποιούμε για έξοδο). Αυτό είναι ένα παράδειγμα του πώς η Python κάνει τη ζωή του προγραμματιστή εύκολη.

Ανάθεση τιμών σε μεταβλητές

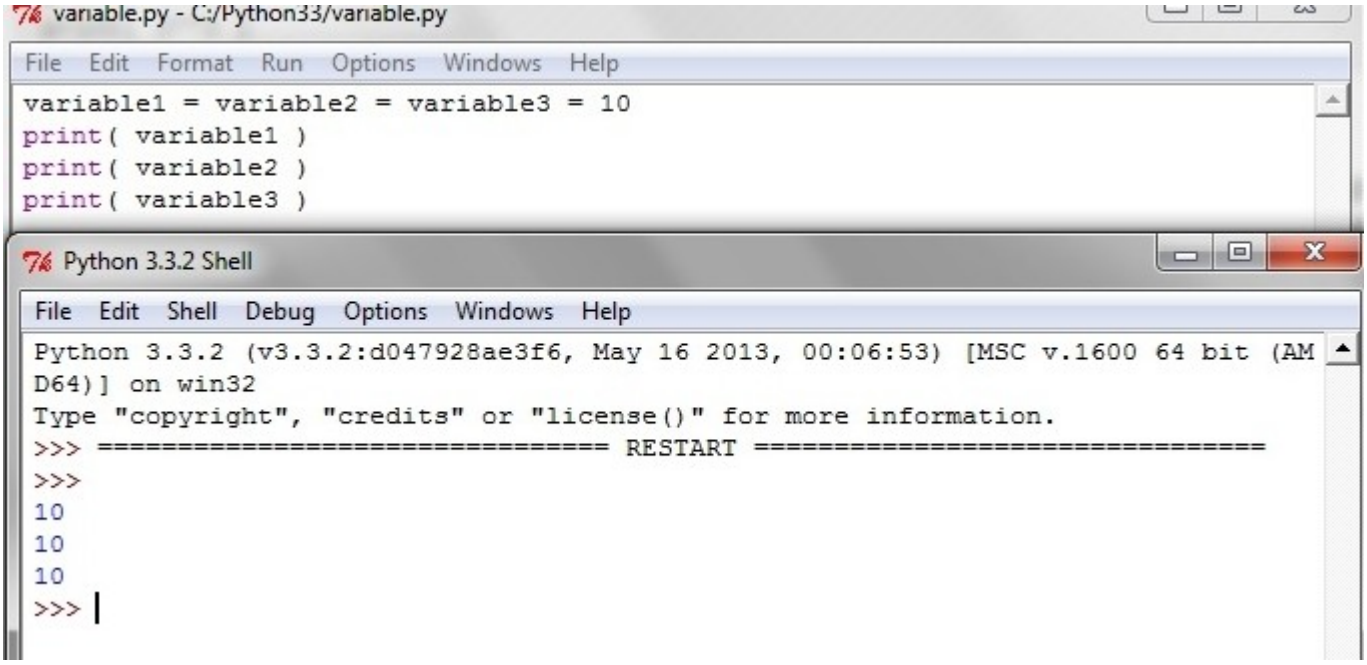
Οι μεταβλητές στην python δεν έχουν τύπους. Αυτό σημαίνει ότι σε μια μεταβλητή μπορεί να γίνει εκχώρηση μιας τιμής χωρίς να μας ενδιαφέρει αν πρόκειται για ακέραιο, πραγματικό, αλφαριθμητικό, λίστα, αντικείμενο, κλπ. Γράφουμε το όνομα της μεταβλητής και η τιμή ακολουθεί μετά το σύμβολο ‘=’. Στο παράδειγμα που

ακολουθεί μπορείτε να δείτε ένα προγραμματάκι python το οποίο υπολογίζει το εμβαδό τριγώνου.

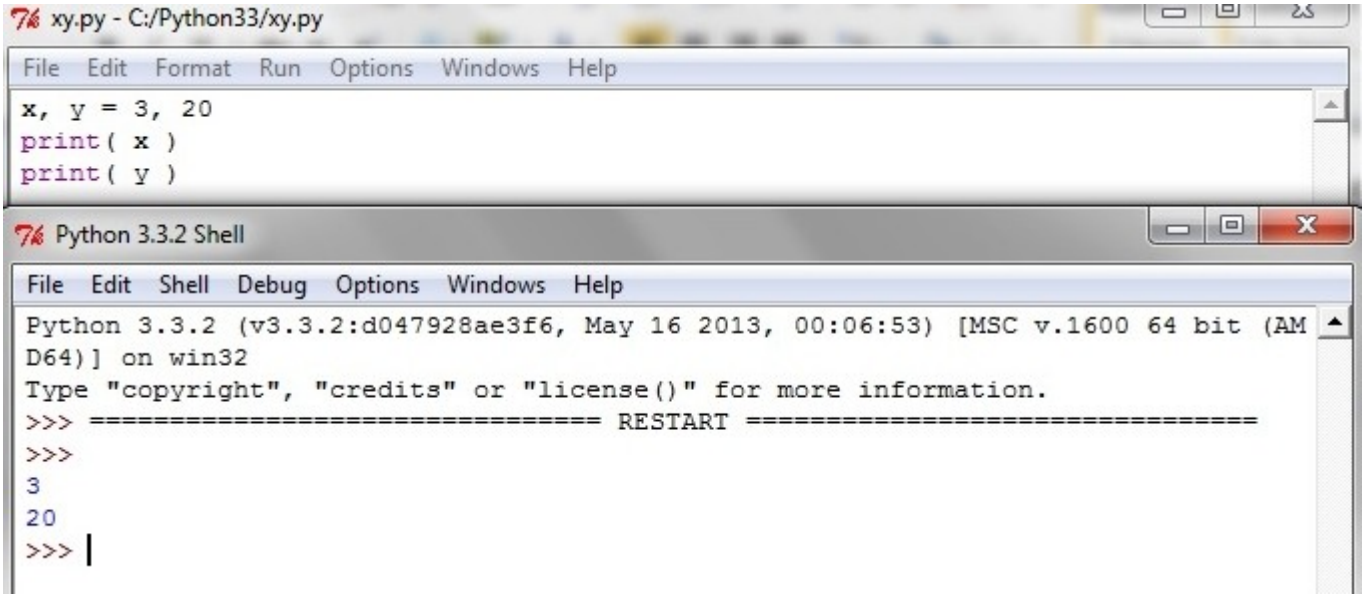


Ανάθεση πολλαπλών τιμών & «Ψυχεδελική ανάθεση» τιμών

Η python μας δίνει τη δυνατότητα να αναθέσουμε την ίδια τιμή σε πολλές μεταβλητές χρησιμοποιώντας μία μόνον δήλωση. Παραθέτω ένα πρόγραμμα που εξηγεί τα παραπάνω :



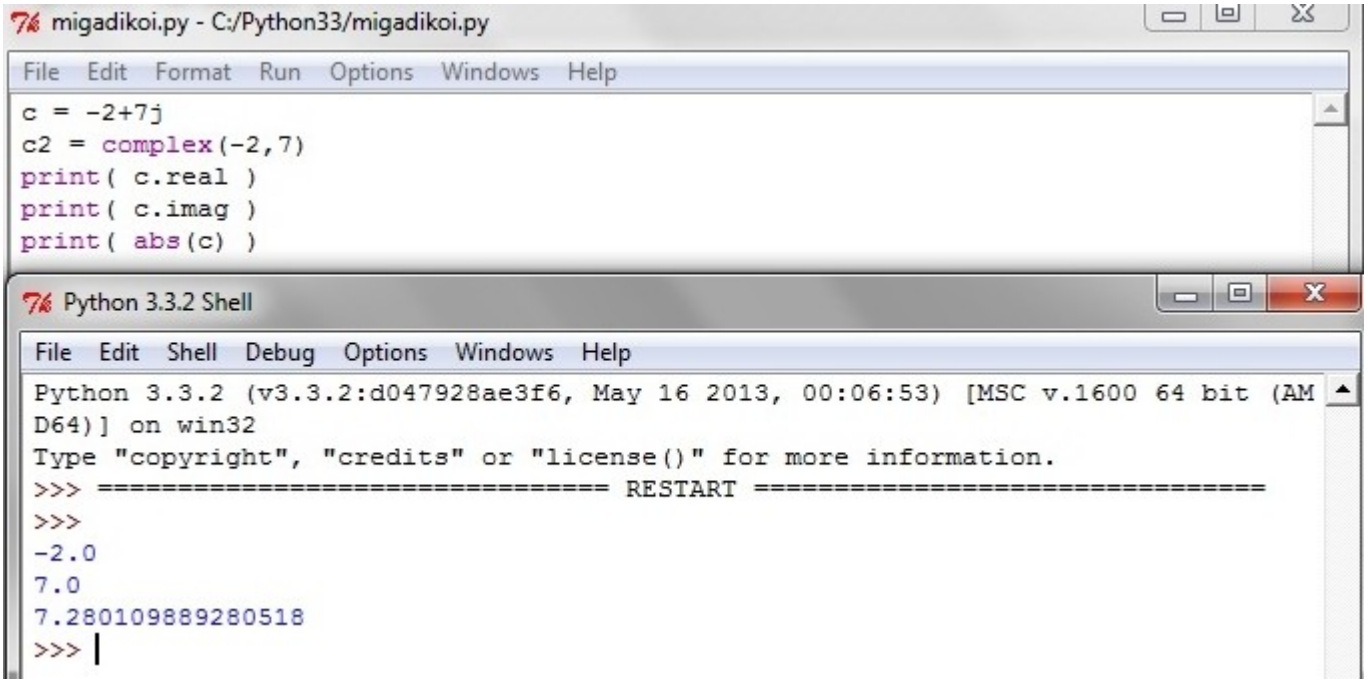
Επίσης μας δίνει τον παρακάτω, άκρως «ψυχεδελικό» τρόπο δήλωσης τιμών.



Όπου οι μεταβλητές παίρνουν κατά σειρά τις τιμές που ακολουθούν. Στο παράδειγμά μας δηλαδή η μεταβλητή x θα πάρει την τιμή 3 και η y την τιμή 20.

Σύνθετοι (μιγαδικοί) αριθμοί

Η υποστήριξη μιγαδικών αριθμών είναι κάτι που δεν βλέπουμε συχνά σε μια γλώσσα προγραμματισμού. Η python μας επιτρέπει να ορίσουμε έναν μιγαδικό με έναν από τους δύο τρόπους που μπορείτε να δείτε στο παράδειγμα που ακολουθεί, να τον αποσυνθέσουμε σε πραγματικό και φανταστικό μέρος και να υπολογίσουμε το μέτρο του**. Είναι επίσης εφικτές πράξεις μεταξύ των μιγαδικών.**



Συνένωση αλφαριθμητικών

Στην python μπορούμε να συνενώσουμε αλφαριθμητικά με 3 τρόπους:

1) Τα αλφαριθμητικά σε παράθεση χωρισμένα με τον χαρακτήρα ‘+’
2) Τα αλφαριθμητικά απλά σε παράθεση
3) Με χρήση της συνάρτησης .strip()

```
string1 = "Are you suggesting"

string2 = "coconuts "

print( string1 + " " +  string2 + "migrate?")

print("Are you suggesting" " coconuts migrate?")

print('Are you suggesting'.strip() + ' coconuts migrate?')
```

Το αλφαριθμητικό ως πίνακας χαρακτήρων

Ένα αλφαριθμητικό στην Python μπορεί κανείς να το διαχειριστεί και ως πίνακα χαρακτήρων (λειτουργικότητα που παρέχεται και από άλλες γλώσσες όπως για παράδειγμα η C).

Για να καταλάβουμε λίγο καλύτερα γιατί ακριβώς μιλάμε αν έχουμε το αλφαριθμητικό “test” σαν πίνακας χαρακτήρων αυτό αναπαρίσταται ως:

0	**1**	**2**	**3**
T	e	S	t

Δηλαδή αν η μεταβλητή μας που περιέχει το αλφαριθμητικό καλείται myString τότε δίνοντας στον διερμηνευτή της Python myString[0] θα πρέπει να μας επιστρέψει τον χαρακτήρα ‘t’.

Ας δούμε λίγο πιο αναλυτικά τι ενέργειες μας επιτρέπει η Python να εκτελέσουμε σε έναν πίνακα χαρακτήρων.

```
text = "Are you suggesting coconuts migrate?"

text[8]

text[0:3]

text[4:7]

text[:19]

text[19:]
```

Δημιουργούμε το αλφαριθμητικό text. Εν συνέχεια, οι γραμμές κώδικα που σας δίνουμε έχουν τα εξής αποτελέσματα (κατά σειρά εμφάνισης):

- ✓ Επιστρέφεται ο χαρακτήρας που βρίσκεται στην θέση 8 (δηλαδή ο 9ος κατά σειρά χαρακτήρας του αλφαριθμητικού αφού η αρίθμηση των θέσεων ξεκινά από το 0)
- ✓ Επιστρέφονται οι χαρακτήρες από τη θέση 0 ως την 3
- ✓ Επιστρέφονται οι χαρακτήρες από τη θέση 4 ως την 7
- ✓ Επιστρέφονται οι χαρακτήρες από τη θέση 19 και πριν'
- ✓ Επιστρέφονται οι χαρακτήρες από τη θέση 19 και μετά

Εκτός από θετικούς δείκτες θέσεων μπορούμε να έχουμε και αρνητικούς.

```
text = "Are you suggesting coconuts migrate?"
```

```
text[-1]
```

```
text[-2]
```

```
text[-3:]
```

```
text[:-10]
```

```
text[-0]
```

Και πάλι κατά σειρά εμφάνισης:

- Επιστρέφεται ο τελευταίος χαρακτήρας
- Επιστρέφεται ο προ-τελευταίος χαρακτήρας
- Επιστρέφονται οι 3 τελευταίοι χαρακτήρες
- Επιστρέφονται οι χαρακτήρες από την δέκατη, από το τέλος, θέση και πριν
- Επιστρέφεται ο πρώτος χαρακτήρας [το -0 θεωρείται ίσο με το 0]

→ Μπορούμε να συνδυάσουμε την συνένωση αλφαριθμητικών με τους πίνακες χαρακτήρων ως εξής:

```
'Do ' + text[4:]
```

Η συγκεκριμένη γραμμή κώδικα συνενώνει το αλφαριθμητικό 'Do' με τους χαρακτήρες του αλφαριθμητικού της μεταβλητής text από την τέταρτη θέση και μετά.

Hacks πινάκων χαρακτήρων

Θεωρούμε το ίδιο αλφαριθμητικό :

```
text = "Are you suggesting coconuts migrate?"
```

```
text[1:2000]
```

```
text[120:]
```

```
text[5:4]
```

```
text[-100:]
```

Κατά σειρά εμφάνισης:

- Παίρνουμε το κομμάτι του πίνακα χαρακτήρων στο οποίο υπάρχει πληροφορία
- Παίρνουμε “ “ (κενό χαρακτήρα)
- Παίρνουμε “ “ (κενό χαρακτήρα)
- Παίρνουμε το κομμάτι του πίνακα χαρακτήρων στο οποίο υπάρχει πληροφορία

Από τα παραπάνω συμπεραίνουμε δηλαδή πως όταν οι θέσεις που ζητούμε δεν υπάρχουν ή δεν έχουν πληροφορία η Python μας επιστρέφει έναν κενό χαρακτήρα.

ΠΡΟΣΟΧΗ: Αν ζητήσουμε τον χαρακτήρα μιας θέσης με αρνητικό δείκτη που βρίσκεται έξω από τα όρια του αλφαριθμητικού μας τότε η Python θα μας επιστρέψει error. Για παράδειγμα, για το αλφαριθμητικό Are you suggesting coconuts migrate? Η εντολή text[-50] θα προκαλέσει σφάλμα.

Εισαγωγή Unicode χαρακτήρα

Η εισαγωγή Unicode χαρακτήρων γίνεται στην Python με τη χρήση του \u ακολουθούμενο από τον κωδικό του χαρακτήρα. Για παράδειγμα:

```
print( "Are\u0020you suggesting..." )
```

Θα εμφανίσει στην κονσόλα μας: Are you suggesting…

Μεταφράζοντας το \u0020 ως τον κενό χαρακτήρα.

Το μήκος ενός αλφαριθμητικού

Το μήκος ενός αλφαριθμητικού μπορούμε να το ανακτήσουμε χρησιμοποιώντας την συνάρτηση len() όπως φαίνεται παρακάτω:

```
len(text)
```

Λίστες

Οι λίστες είναι μια πολύ ενδιαφέρουσα και σημαντική δομή που προσφέρει η Python. Σαν δομή δεδομένου συναντάται σε πολλές γλώσσες προγραμματισμού και αποτελεί, όπως διαισθητικά καταλαβαίνει κανείς μια συλλογή πραγμάτων.

Στην Python δηλώνουμε μια λίστα περικλείοντας τα αντικείμενά της σε τετραγωνικές αγκύλες [… , … , … ,]. Δεν είναι απαραίτητο να δηλώσουμε εξ' αρχής όλα τα αντικείμενα που θέλουμε να περιλαμβάνει η λίστα μας. Επίσης, λόγω του ότι στην Python δεν δηλώνουμε τύπους δεδομένων οι λίστες μας μπορούν να αποτελούνται από αντικείμενα διαφορετικά μεταξύ τους (ακεραίους, πραγματικούς, αλφαριθμητικά, κ.ο.κ.).Ας δούμε μερικά παραδείγματα:Έστω η λίστα mylist:

```
myList = ['monty', 'python', 999, 222]

myList

myList[0]

myList[-1]

myList[1:-1]
```

Κατά σειρά οι παραπάνω εντολές έχουν τα εξής αποτελέσματα:

- Εμφανίζεται η λίστα όπως έχει δηλωθεί
- Εμφανίζεται το πρώτο στοιχείο της λίστας
- Εμφανίζεται το τελευταίο στοιχείο της λίστας
- Εμφανίζονται τα στοιχεία της λίστας μεταξύ του πρώτου και του τελευταίου

Εισαγωγή στοιχείου σε λίστα

```
mylist[1:1] = ['to1', 'to2']
```

Η παραπάνω εντολή εισάγει τα στοιχεία 'to1' και 'to2' στην δεύτερη και τρίτη θέση της λίστας που δηλώσαμε στο προηγούμενο ακριβώς παράδειγμα.

Εν γένει:

mylist[X] = κάποιος τύπος δεδομένου

Θα εισάγει το περιεχόμενο της μεταβλητής Κάποιος τύπος δεδομένου στην θέση X της λίστας myList

Διαγραφή στοιχείου από λίστα

```
mylist[0:2] = []
```

Η παραπάνω εντολή θα διαγράψει τα τρία πρώτα στοιχεία της λίστας mylist

Εν γένει:

mylist[X] = []

Θα διαγράψει το στοιχείο που βρίσκεται στη θέση X της λίστας mylist

Εισαγωγή ενός αντιγράφου της λίστας στην αρχή της

```
mylist[:0] = mylist
```

Η παραπάνω εντολή θα δημιουργήσει ένα αντίγραφο της λίστας mylist στην αρχή της

Καθαρισμός της λίστας

```
mylist[:] = []
```

Η παραπάνω εντολή θα καθαρίσει την λίστα mylist από τα περιεχόμενά της

Μήκος μιας λίστας

Η συνάρτηση len() που χρησιμοποιείται για να μας δώσει το μήκος ενός πίνακα χαρακτήρων χρησιμοποιείται και στις λίστες με τον ίδιο ακριβώς σκοπό.

Επομένως η εντολή:

`Len(mylist)` → Μας επιστρέφει τον αριθμό των στοιχείων που περιέχει η λίστα mylist

Εμφωλευμένες λίστες

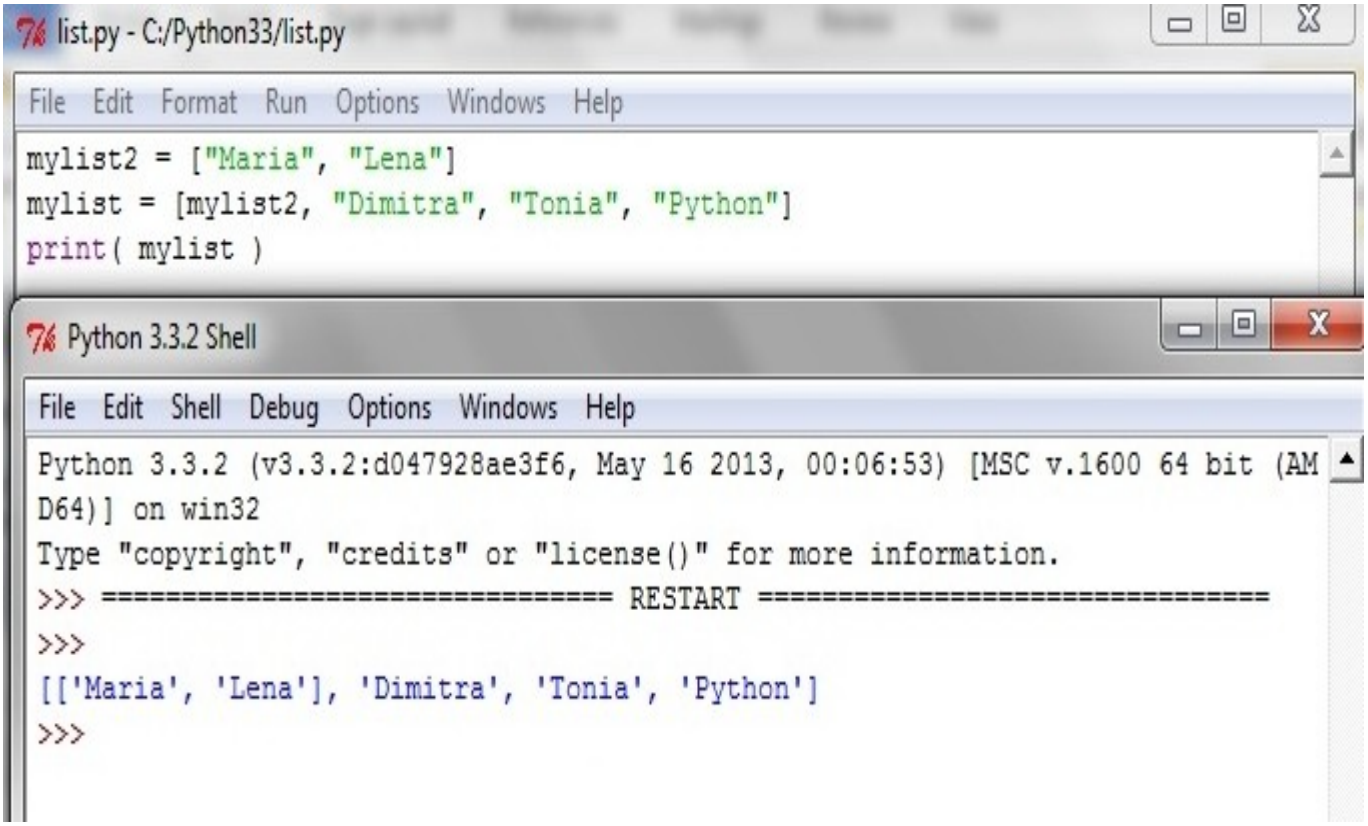
Η Python υποστηρίζει τον όρο της εμφωλευμένης λίστας δηλαδή μια λίστα μέσα σε μια άλλη λίστα.

mylist2 = ["Maria", "Lena"]

mylist = [mylist2, "Dimitra", "Tonia", "Python"]

print(mylist)

Στον παραπάνω κώδικα η λίστα mylist2 αποτελεί στοιχείο της λίστας mylist και επομένως αν εκτελέσουμε το συγκεκριμένο τμήμα κώδικα το αποτέλεσμα που θα πάρουμε θα είναι:



Προσθήκη στοιχείου στο τέλος της λίστας με χρήση της append()

Εκτελώντας

`mylist.append(X)`

το στοιχείο X προστίθεται στο τέλος της τρέχουσας μορφής της λίστας mylist.

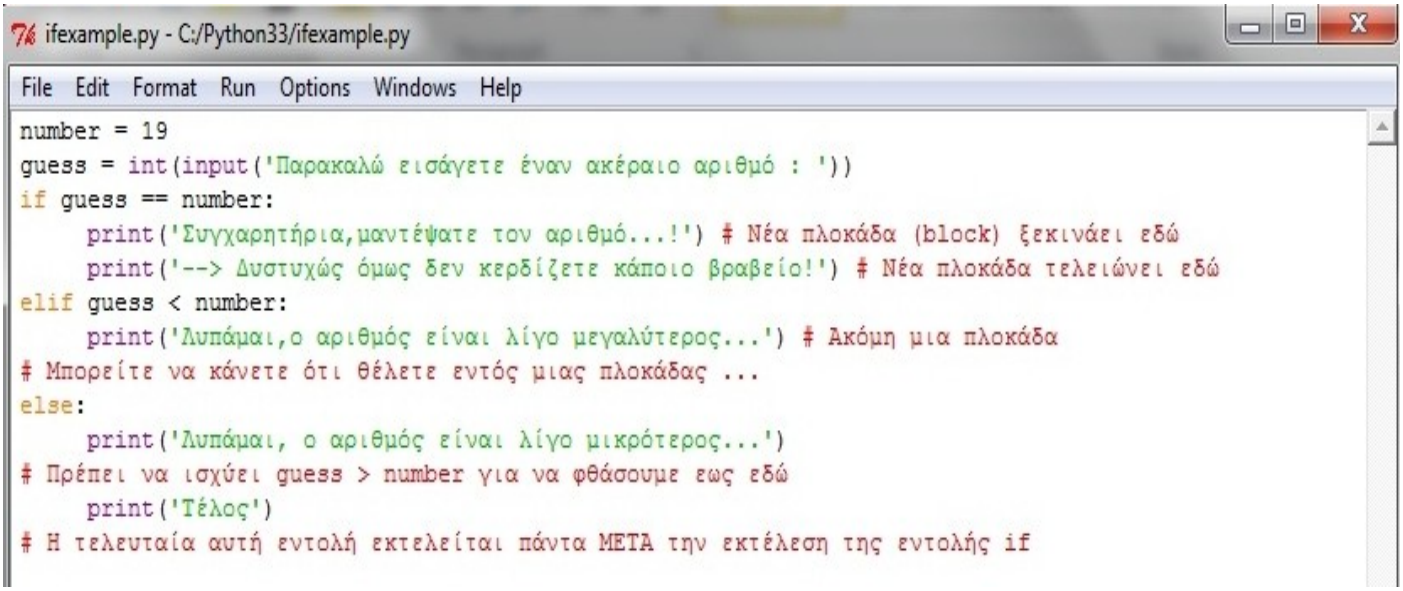
Έλεγχος ροής

Στα προγράμματα που γράψαμε και εκτελέσαμε ως τώρα, υπήρξε πάντα μια σειρά εντολών, τις οποίες εκτελούσε πιστά η Python με την ίδια σειρά. Τι γίνεται όμως αν θέλουμε να αλλάξουμε την ροή εκτέλεσης; Αν για παράδειγμα, θέλουμε το πρόγραμμα να πάρει μερικές αποφάσεις και να κάνει διαφορετικά πράγματα υπό διαφορετικές προϋποθέσεις, όπως π.χ. να εκτυπώσει “καλημέρα” ή “καλησπέρα”, ανάλογα με την ώρα; Αυτό επιτυγχάνεται χρησιμοποιώντας εντολές ελέγχου ροής. Υπάρχουν τρεις εντολές ελέγχου ροής στην Python **- if, for και while.**

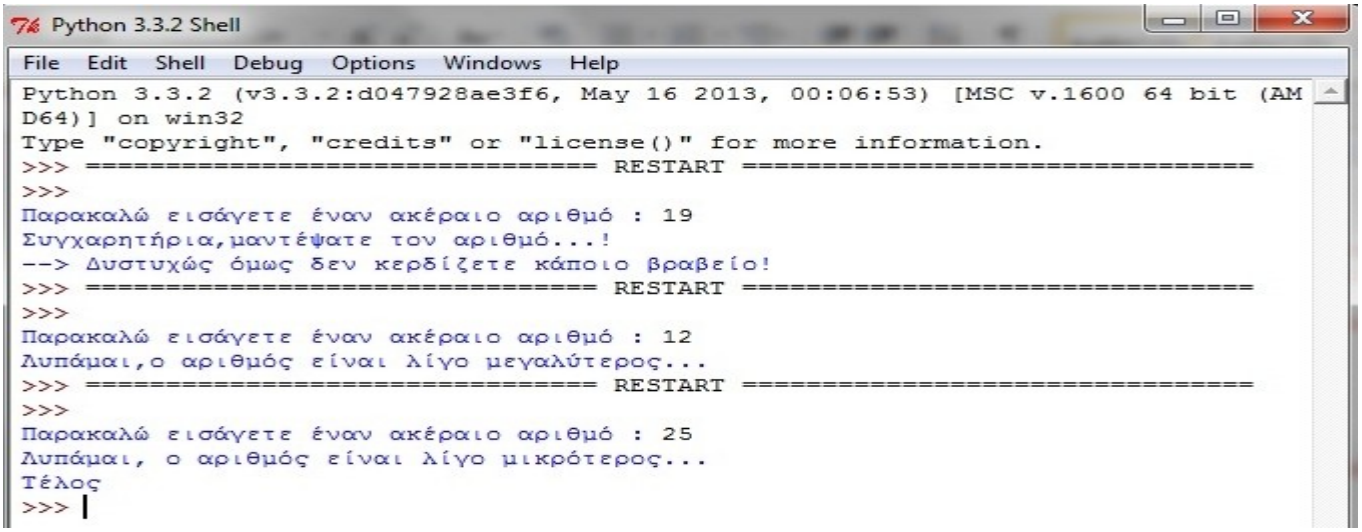
Η εντολή if

Η εντολή if χρησιμοποιείται για να ελεγχθεί μια συνθήκη και εάν (if) η συνθήκη αυτή είναι αληθής, τότε εκτελείται ένα σύνολο ή πλοκάδα εντολών (που ονομάζεται if-block), διαφορετικά (else) γίνεται επεξεργασία ενός άλλου συνόλου εντολών (που ονομάζεται else-block). Η χρήση του όρου else είναι προαιρετική.

Ακολουθεί ένα απλό παράδειγμα με την χρήση της εντολής if.



Αποθηκεύω ως ifexample.py και στη συνέχεια Run→Run Module (F5) και δίνοντας διάφορες τιμές προκύπτει :



Πώς λειτουργεί το παραπάνω πρόγραμμα:

Στο πρόγραμμα αυτό μαντεύει ο χρήστης αριθμούς και ελέγχουμε αν αντιστοιχούν στον αριθμό που έχουμε ορίσει. Ορίζουμε την μεταβλητή number ως οποιονδήποτε ακέραιο αριθμό επιθυμούμε, π.χ. 19. Ύστερα παίρνουμε την πρόγνωση του χρήστη χρησιμοποιώντας την συνάρτηση input(). Οι συναρτήσεις είναι απλά

επαναχρησιμοποιήσιμα κομμάτια προγραμμάτων. Θα αναφερθούμε εκτενέστερα σε αυτές στη συνέχεια γι' αυτές.

Παρέχουμε μια συμβολοσειρά (string) στην ενσωματωμένη συνάρτηση input, η οποία εκτυπώνει στην οθόνη αυτή την συμβολοσειρά και αναμένει την εισαγωγή δεδομένων από τον χρήστη. Μόλις εισάγουμε κάτι και πατήσουμε το πλήκτρο enter, η συνάρτηση input() επιστρέφει αυτό που εισάγαμε ως συμβολοσειρά. Στην

συνέχεια, αυτή η συμβολοσειρά μετατρέπεται σε ακέραιο αριθμό με την χρήση της int κι αποθηκεύεται στην μεταβλητή guess. Στην πραγματικότητα, η int είναι μια κλάση, άλλα το μόνο που χρειάζεται να ξέρετε προς το παρόν είναι ότι μπορείτε να την χρησιμοποιήσετε για να μετατρέπετε μια συμβολοσειρά σ' έναν ακέραιο αριθμό (υπό την προϋπόθεση ότι η συμβολοσειρά περιέχει έναν έγκυρο ακέραιο αριθμό στο κείμενο).

Στο επόμενο βήμα συγκρίνουμε το προγνωστικό του χρήστη με τον αριθμό που επιλέξαμε. Εάν είναι ίδιοι, εκτυπώνεται ένα μήνυμα επιτυχίας. Σημειώστε ότι χρησιμοποιούμε επίπεδα εσοχών για να δηλώσουμε στην Python ποιες εντολές ανήκουν σε ποια πλοκάδα. Γι' αυτό το λόγο, **οι εσοχές είναι πολύ σημαντικές στην Python.**

Παρατηρήση → εντολή if περιλαμβάνει άνω και κάτω τελεία στο τέλος – έτσι δηλώνουμε στην Python ότι ακολουθεί μια πλοκάδα εντολών.

Στην συνέχεια ελέγχουμε αν το προγνωστικό είναι μικρότερο από τον αριθμό μας, κι αν ναι, πληροφορούμε τον χρήστη ότι η πρόγνωσή του πρέπει να είναι λίγο μεγαλύτερη απ' αυτήν. Αυτό που χρησιμοποιήσαμε εδώ είναι ο όρος elif ο οποίος στην πραγματικότητα συνδυάζει δύο συσχετιζόμενες εντολές if else-if else σε μία εντολή if-elif-else. Αυτό κάνει το πρόγραμμα ευκολότερο και μειώνει τον αριθμό των εσοχών που απαιτούνται.

Οι εντολές elif και else πρέπει επίσης να έχουν άνω και κάτω τελεία στο τέλος της λογικής γραμμής, ακολουθούμενες από τις αντίστοιχες πλοκάδες εντολών (με κατάλληλες εσοχές, βεβαίως). Μπορείτε να έχετε άλλη μία εντολή if εντός της πλοκάδας if (if-block) μιας εντολής if κ.οκ. Αυτό αποκαλείται **εμφωλευμένη εντολή if.**

Είναι σημαντικό να θυμόμαστε ότι τα τμήματα elif και else είναι προαιρετικά. Μια ελάχιστη έγκυρη εντολή if είναι:

```
if True:
print('Ναι, είναι αληθές!')
```

Αφού η Python τελειώσει την εκτέλεση ολόκληρης της εντολής if συμπεριλαμβανομένων και των συσχετιζόμενων όρων elif και else, προχωράει στην επόμενη εντολή στην πλοκάδα που περιέχει την εντολή if. Στην περίπτωσή μας πρόκειται για την κύρια πλοκάδα με την οποία ξεκινάει η εκτέλεση του προγράμματος και η επόμενη εντολή είναι print('Τέλος'). Μετά απ' αυτό η Python βλέπει το τέλος του προγράμματος και απλά τελειώνει εκεί.

Δεν υπάρχει εντολή switch στην Python. Μπορείτε να χρησιμοποιήσετε μια εντολή if..elif..else για να κάνετε το ίδιο πράγμα (και σε μερικές περιπτώσεις να χρησιμοποιήσετε ένα λεξικό για να το κάνετε γρήγορα).

Η εντολή while

Η εντολή while σας επιτρέπει να εκτελείτε επανειλημμένα μια πλοκάδα εντολών, όσο μια προϋπόθεση παραμένει αληθές. Η εντολή while είναι ένα παράδειγμα αυτού που αποκαλείται εντολή βρόχου (looping statement). Μια εντολή while μπορεί να έχει έναν προαιρετικό όρο else.

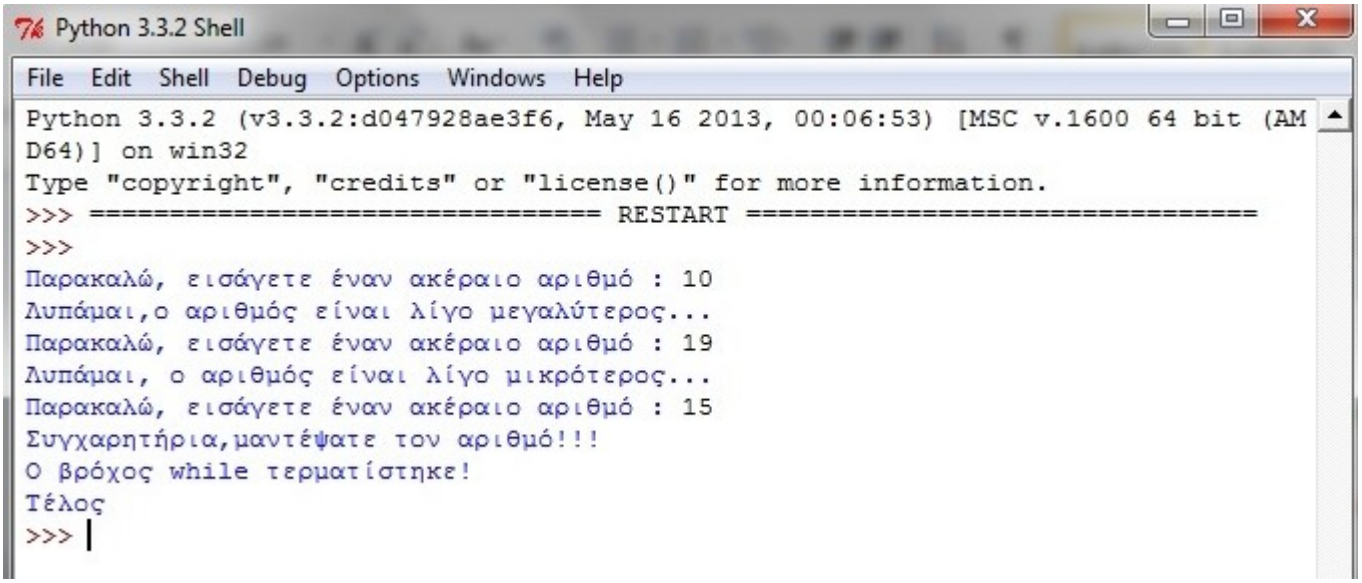
Στη συνέχεια παραθέτω ένα απλό παράδειγμα χρήσης της εντολής while.

whileexample.py - C:/Python33/whileexample.py

File Edit Format Run Options Windows Help

```
number = 15
running = True
while running:
    guess = int(input('Παρακαλώ, εισάγετε έναν ακέραιο αριθμό : '))
    if guess == number:
        print('Συγχαρητήρια,μαντέψατε τον αριθμό!!!')
        running = False # αυτό κάνει τον βρόχο while να σταματήσει εδώ
    elif guess < number:
        print('Λυπάμαι,ο αριθμός είναι λίγο μεγαλύτερος...')
    else:
        print('Λυπάμαι, ο αριθμός είναι λίγο μικρότερος...')
else:
        print('Ο βρόχος while τερματίστηκε!')
# Μπορείτε να προσθέσετε ό,τι επιπλέον θέλετε εδώ
print('Τέλος')
```

Αποθηκεύω το πρόγραμμα ως whileexample.py , εκτελώ Run→Run Module (F5) και προκύπτει :



Πώς λειτουργεί το παραπάνω πρόγραμμα :

Σ' αυτό το πρόγραμμα παίζουμε ακόμα το παιχνίδι πρόγνωσης, αλλά το πλεονέκτημα είναι ότι επιτρέπει στον χρήστη να συνεχίσει να μαντεύει μέχρι να βρει τον σωστό αριθμό -δεν χρειάζεται δηλ. να εκτελεί επανειλημμένα το πρόγραμμα για κάθε πρόγνωση, όπως γινόταν στο προηγούμενο παράδειγμα. Αυτό είναι ένα πολύ καλό παράδειγμα της χρήσης της εντολής while.

Μετακινούμε τις εντολές input και if εντός του βρόχου while και ορίζουμε την μεταβλητή running σε True πριν τον βρόχο while. Πρώτα ελέγχουμε αν η μεταβλητή running έχει την τιμή True και στην συνέχεια προχωράμε στην εκτέλεση της αντίστοιχης πλοκάδας while (while-block). Αφού εκτελεστεί αυτή η πλοκάδα, η προϋπόθεση ελέγχεται και πάλι, δηλ. στην περίπτωσή μας η μεταβλητή running. Εάν η τιμή της συνεχίζει να είναι true (αληθές), εκτελείται και πάλι ολόκληρη η πλοκάδα while (while-block), διαφορετικά προχωράμε στην εκτέλεση της προαιρετικής πλοκάδας else (else-block) και μετά προχωράμε στην επόμενη εντολή.

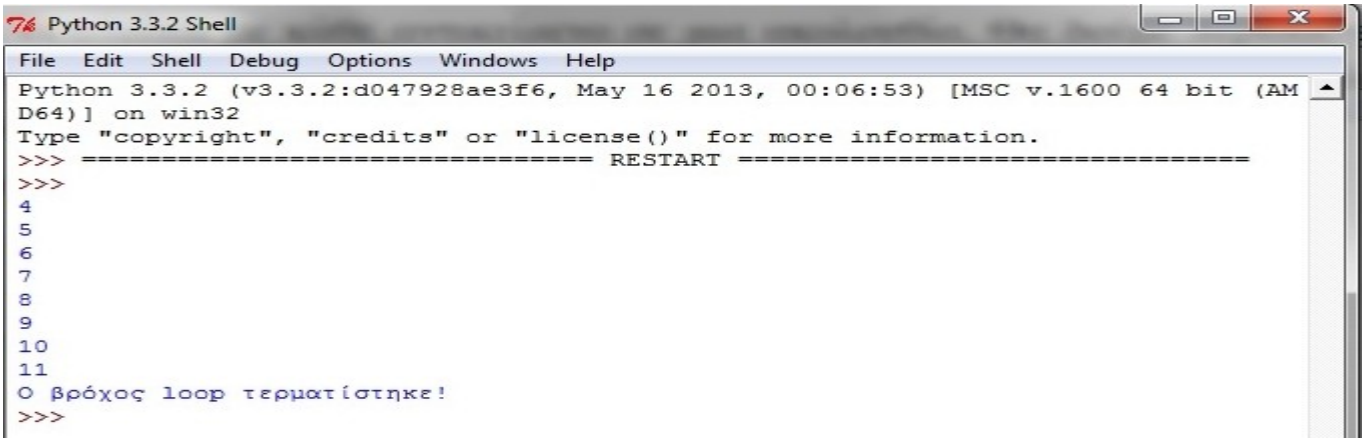
Η πλοκάδα else εκτελείται όταν η προϋπόθεση του βρόχου while αποκτά την τιμή False (ψευδές) –αυτό μπορεί να συμβεί ακόμα και κατά την πρώτη φορά που ελέγχεται η προϋπόθεση. Εάν υπάρχει ένας όρος else για ένα βρόχο while, εκτελείται πάντοτε, εκτός κι αν διακόψετε τον βρόχο με την εντολή break. Οι τιμές True (αληθές) και False (ψευδές) ονομάζονται Μπούλειοι τύποι (Boolean types) και μπορείτε να θεωρήσετε ότι αντιστοιχούν στις τιμές 1 και 0 αντίστοιχα.

Ο βρόχος for

Η εντολή ***for..in*** είναι άλλη μία εντολή βρόχου, η οποία ***επαναλαμβάνεται*** σε μια ακολουθία αντικειμένων, δηλ. εκτελείται σε κάθε αντικείμενο σε μια ακολουθία. Θα δούμε περισσότερες λεπτομέρειες σχετικά με τις ακολουθίες στα επόμενα κεφάλαια. Αυτό που πρέπει να γνωρίζετε προς το παρόν είναι ότι μια ακολουθία είναι απλά μια ταξινομημένη συλλογή αντικειμένων.

Ακολουθεί παράδειγμα χρήσης της εντολής for...in :

Και τρέχοντάς το προκύπτει :



Πώς λειτουργεί το παραπάνω πρόγραμμα:

Σ' αυτό το πρόγραμμα εκτυπώνουμε μια ακολουθία αριθμών. Παράγουμε αυτή την ακολουθία αριθμών με την ενσωματωμένη συνάρτηση range. Αυτό που κάνουμε εδώ είναι ότι παρέχουμε δύο αριθμούς και η range επιστρέφει μια ακολουθία αριθμών, ξεκινώντας από τον πρώτο αριθμό έως το δεύτερο αριθμό. Για παράδειγμα, η range(4,12) δίνει την ακολουθία [4, 5, 6, 7,8,9,10,11]. Εξ ορισμού, η range έχει ως αριθμό κλιμάκωσης (step count) το 1. Εάν παρέχουμε έναν τρίτο αριθμό στην range, τότε αυτός γίνεται ο αριθμός κλιμάκωσης (step count). Για παράδειγμα, range(1,5,2) δίνει ως αποτέλεσμα [1,3]. Θυμηθείτε το εύρος των αριθμών εκτείνεται έως τον δεύτερο αριθμό, δηλ. δεν συμπεριλαμβάνει τον δεύτερο αριθμό.

Στην συνέχεια, ο βρόχος for επαναλαμβάνεται σ' αυτό το εύρος - for i in range(4,12) είναι αντίστοιχο του for i in [4, 5, 6, 7,8,9,10,11] που είναι σαν να αντιστοιχούμε

κάθε αριθμό (ή αντικείμενο) της ακολουθίας στο i ξεχωριστά και στην συνέχεια να εκτελούμε την πλοκάδα των εντολών για κάθε τιμή της i.

Στην περίπτωσή μας, απλά εκτυπώνουμε την τιμή στην πλοκάδα των εντολών. Θυμηθείτε ότι το τμήμα else είναι προαιρετικό. Όταν συμπεριλαμβάνεται, εκτελείται πάντα μόλις τερματιστεί ο βρόχος for, εκτός κι αν συναντηθεί η εντολή break ενδιάμεσα.

Επίσης θυμηθείτε ότι ο βρόχος for..in δουλεύει για οποιαδήποτε ακολουθία. Εδώ έχουμε μια λίστα αριθμών που παράγονται από την ενσωματωμένη συνάρτηση range, αλλά γενικά μπορούμε να χρησιμοποιήσουμε οποιοδήποτε είδος ακολουθίας οποιουδήποτε είδους αντικειμένων!

Στην Python, ο βρόχος for είναι ριζικά διαφορετικός από τον βρόχο for της C/C++. Προγραμματιστές της C# θα παρατηρήσουν ότι ο βρόχος for στην Python είναι παρόμοιος με τον βρόχο foreach στην C#. Προγραμματιστές της Java θα παρατηρήσουν ότι αυτός είναι παρόμοιος με το for (int i :IntArray) στην Java 1.5 .

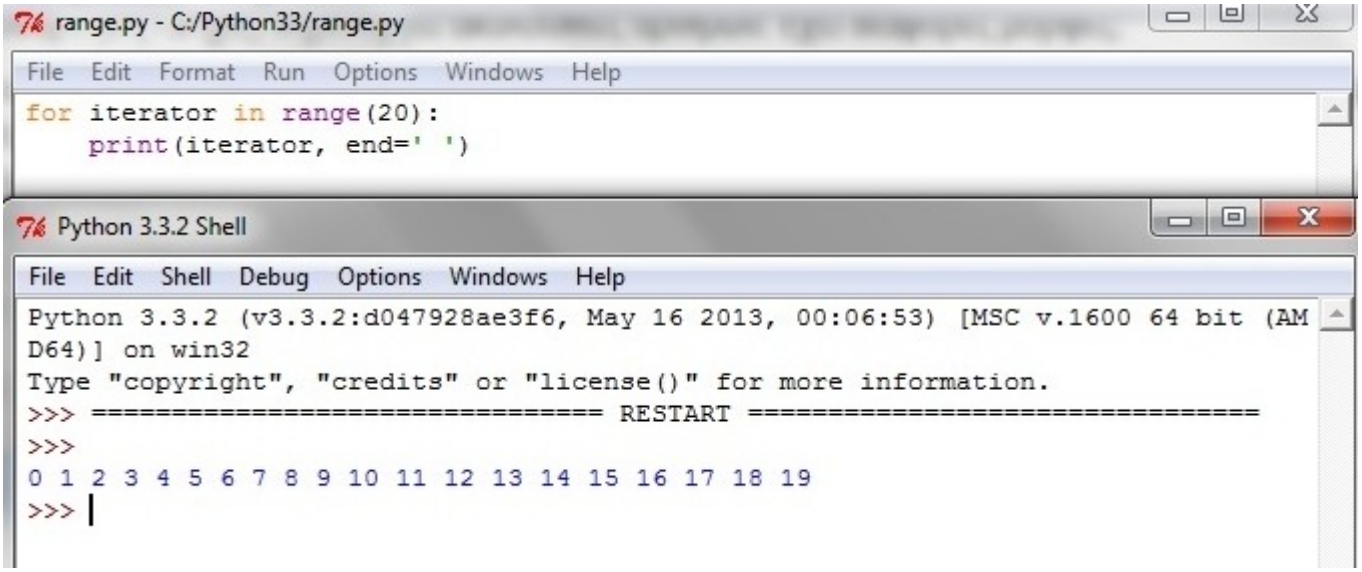
Στη C/C++, αν θέλετε να γράψετε π.χ. for (int i = 0; i < 5; i++), τότε στην Python απλά γράφετε for i in range(0,5). Όπως βλέπετε, ο βρόχος for είναι πιο απλός, πιο εκφραστικός και λιγότερο ευπαθής σε σφάλματα στην Python.

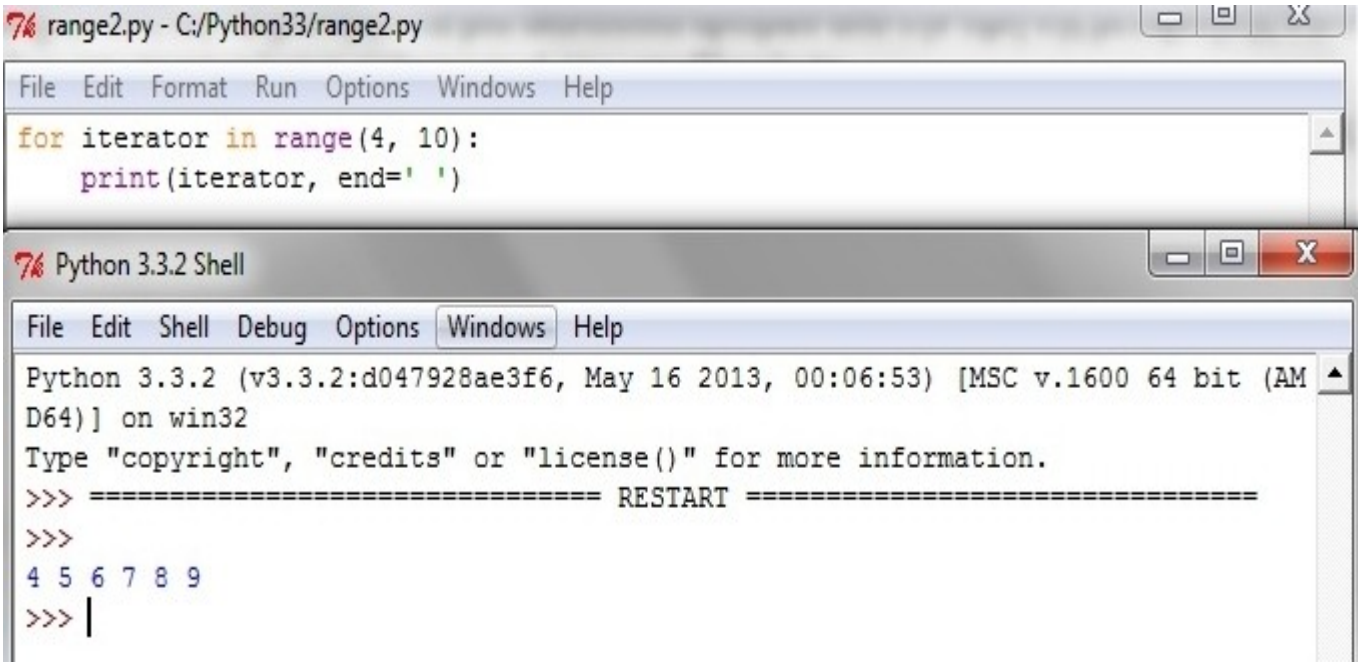
Η συνάρτηση range()

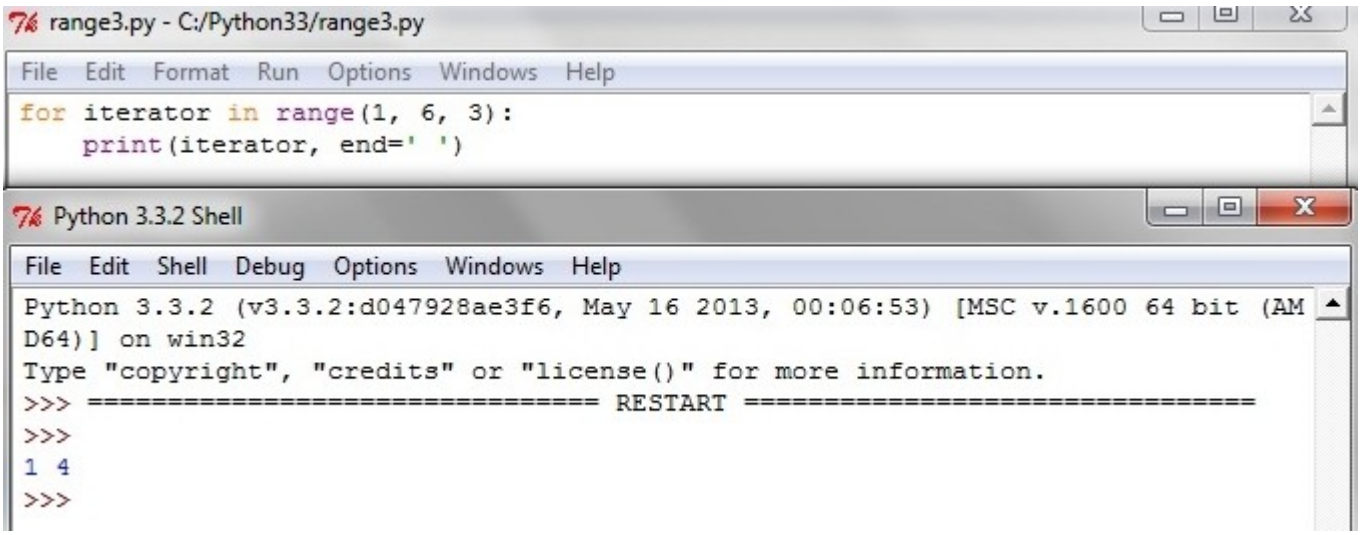
Η συνάρτηση range() δημιουργεί ακολουθίες αριθμών. Έχει διάφορες μορφές:

- ✓ **range(number)** : Δημιουργείται μια ακολουθία αριθμών από το 0 ως την προηγούμενη ακέραια τιμή της μεταβλητής number
- ✓ **range(from, to)** : Δημιουργείται μια ακολουθία αριθμών από την τιμή της μεταβλητής from μέχρι την προηγούμενη ακέραια τιμή της μεταβλητής to
- ✓ **range(from, to, step)** : Δημιουργείται μια ακολουθία αριθμών από την τιμή της μεταβλητής from μέχρι την προηγούμενη ακέραια τιμή της μεταβλητής to με βήμα step.

Ας δούμε κάποια παραδείγματα :





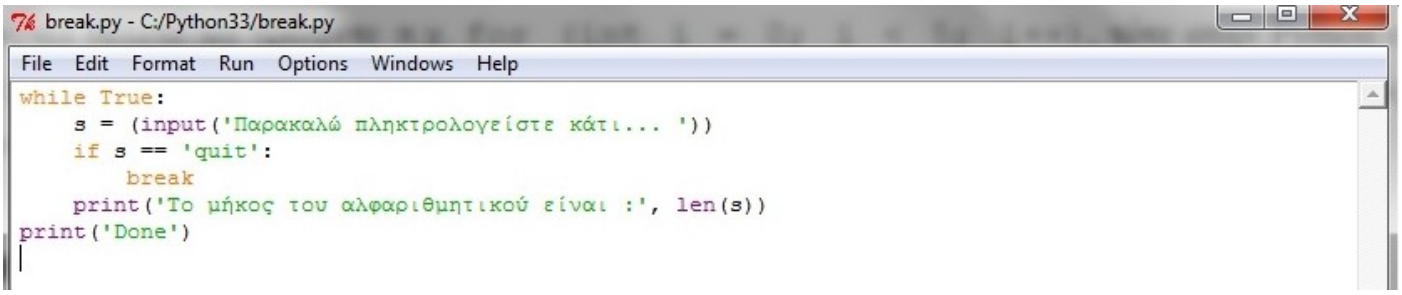


Η εντολή break

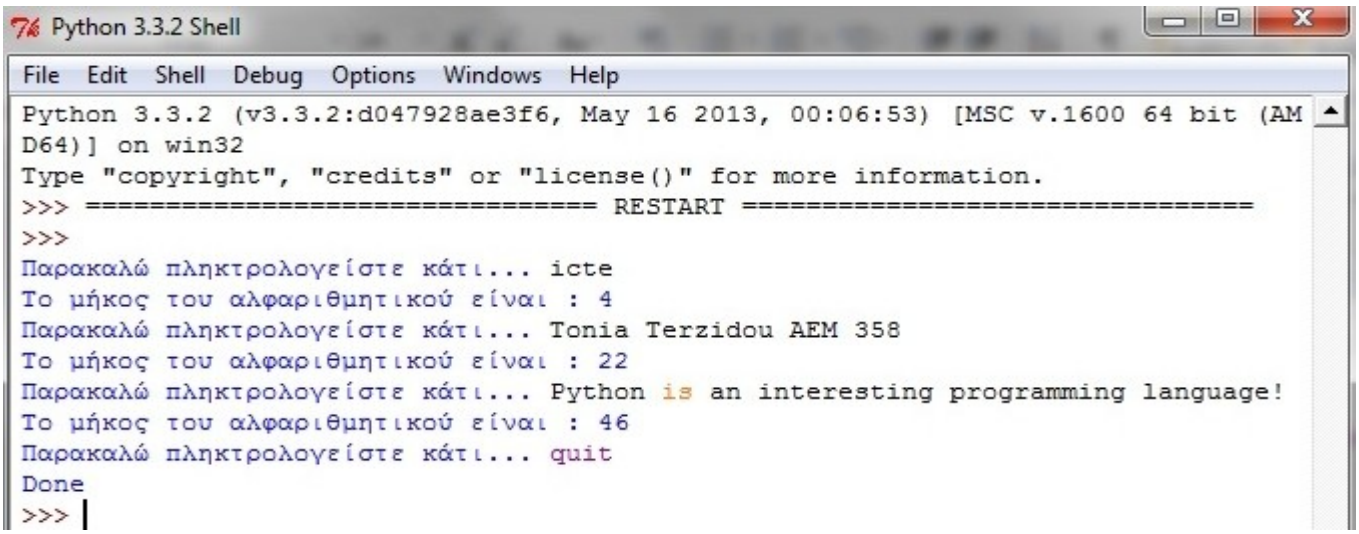
Η εντολή break χρησιμοποιείται για τη διακοπή μιας εντολής βρόχου, δηλ. τη διακοπή της εκτέλεσης της εντολής βρόχου ακόμη κι αν η προϋπόθεση του βρόχου δεν έχει γίνει False (ψευδής) ή δεν έχει επαναληφθεί σ' όλη την ακολουθία των αντικειμένων.

- ✓ Είναι σημαντικό να σημειώσουμε ότι εάν διακόψετε έναν βρόχο for ή while κατ' αυτό τον τρόπο, οποιαδήποτε αντίστοιχη πλοκάδα βρόχου else δεν εκτελείται.

Ακολουθεί ένα παράδειγμα χρήσης της εντολής break :



Από το οποίο προκύπτει έξοδος :



Πώς λειτουργεί το παραπάνω πρόγραμμα:

Σ' αυτό το πρόγραμμα παίρνουμε επανειλημμένα τα εισαχθέντα δεδομένα του χρήστη και εκτυπώνουμε το μήκος κάθε εισαγωγής (δηλ. το σύνολο χαρακτήρων). Έχουμε ορίσει έναν ειδικό όρο για την διακοπή του προγράμματος, ελέγχοντας αν τα εισαχθέντα δεδομένα του χρήστη είναι 'quit'. Σταματάμε το πρόγραμμα διακόπτοντας τον βρόχο και φθάνουμε στο τέλος του προγράμματος. Το μήκος της εισαχθείσας συμβολοσειράς μπορεί να βρεθεί χρησιμοποιώντας την ενσωματωμένη συνάρτηση len .

Καλό θα ήταν να αποφεύγεται η χρήση της δήλωσης break, καθώς κάνει πιο δυσανάγνωστο τον κώδικα. Συχνά μπορεί να αφαιρεθεί με την αλλαγή της συνθήκης

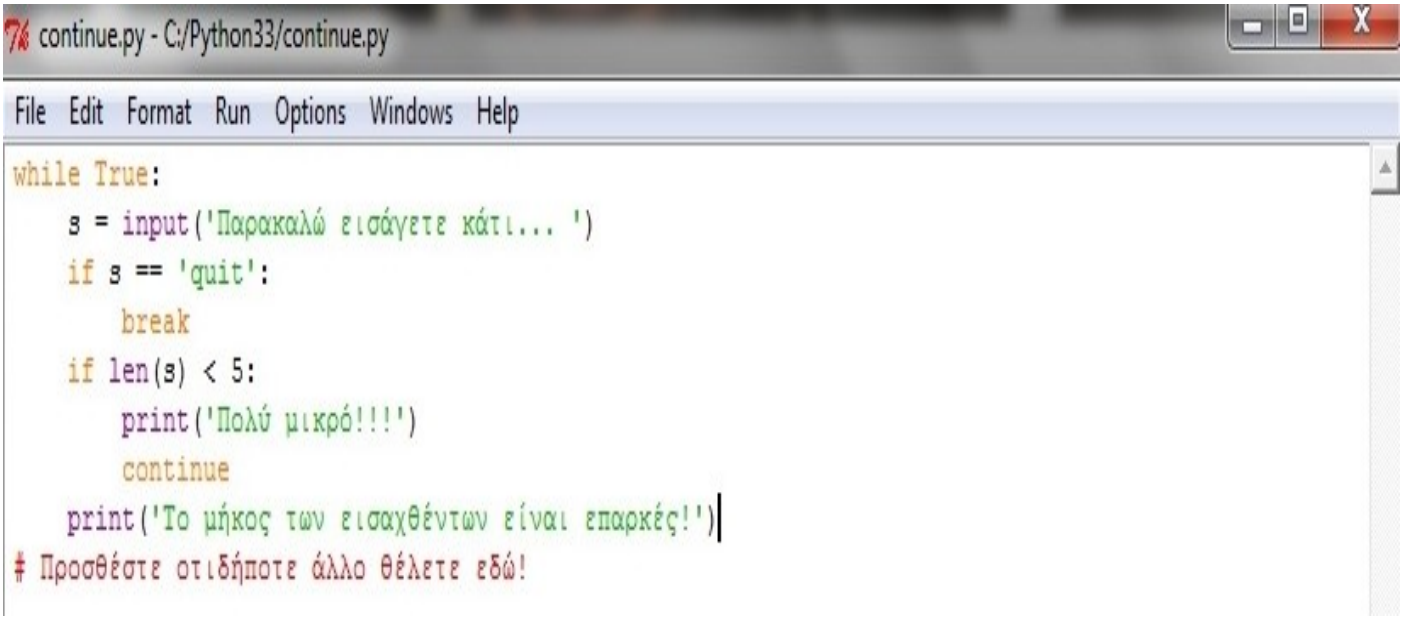
στο βρόγχο. Κάποιες άλλες φορές, βοηθάει στο να κατανοήσουμε πως τερματίζεται ο βρόγχος λόγω κάποιας ειδικής περίπτωσης.

Θυμηθείτε ότι η εντολή break μπορεί να χρησιμοποιηθεί επίσης και με το βρόχο for.

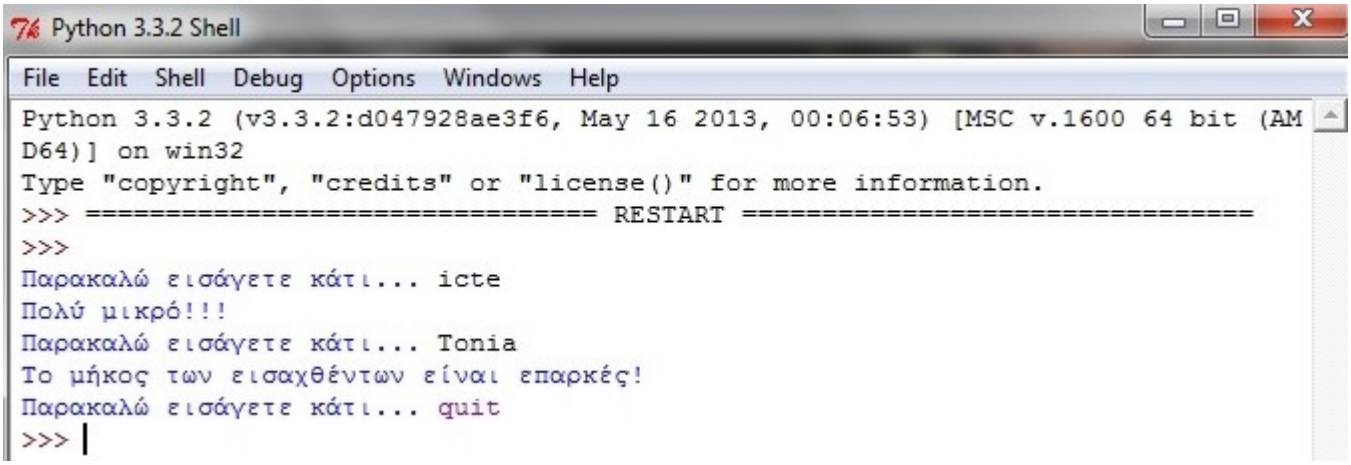
Η εντολή continue

Η εντολή continue χρησιμοποιείται για να υποδείξουμε στην Python να παραλείψει τις υπόλοιπες εντολές στην τρέχουσα πλοκάδα βρόχου και να συνεχίσει με την επόμενη επανάληψη του βρόχου.

Στη συνέχεια παραθέτω ένα απλό πρόγραμμα εφαρμογής της εντολής continue.



Και προκύπτει…



Πώς λειτουργεί το παραπάνω πρόγραμμα:

Σ' αυτό το πρόγραμμα, αποδεχόμαστε δεδομένα εισαγωγής από τον χρήστη, αλλά τα επεξεργαζόμαστε μόνο εάν έχουν τουλάχιστον μήκος 5 χαρακτήρων. Έτσι, χρησιμοποιούμε την ενσωματωμένη συνάρτηση len για να βρούμε το μήκος κι αν αυτό είναι μικρότερο από 3, τότε παραλείπουμε τις υπόλοιπες εντολές στην πλοκάδα χρησιμοποιώντας την εντολή continue. Διαφορετικά, οι υπόλοιπες εντολές στην πλοκάδα εκτελούνται και μπορούμε να κάνουμε οποιοδήποτε είδος επεξεργασίας θέλουμε σ' αυτό το σημείο.

- ⇨ Η εντολή continue δουλεύει επίσης και με το βρόχο for.

Η δήλωση pass

Η δήλωση pass, είναι μια εξαιρετικά ενδιαφέρουσα δήλωση, ίσως η πιο χρήσιμη για έναν προγραμματιστή. Όταν ο Διερμηνευτής τη συναντήσει, δεν κάνει απολύτως τίποτε.

```
while True:
    pass
```

Ο συγκεκριμένος κώδικας θα προκαλέσει έναν ατέρμονα βρόγχο (δηλ έναν βρόγχο που θα απασχολεί αιώνια τον διερμηνευτή σας). Ανά πάσα στιγμή τον διακόπτετε πατώντας την ακολουθία πλήκτρων ctrl + c.

Γιατί όμως είναι κάτι τέτοιο χρήσιμο; Είναι μια εύλογη απορία!

Σκεφτείτε ότι έχετε να κατασκευάσετε ένα μεγάλο πρόγραμμα κάνοντας χρήση αντικειμενοστρεφούς σχεδίασης. Ο αντικειμενοστρέφής προγραμματισμός κάνει χρήση αντικειμένων.Ας υποθέσουμε λοιπόν ότι σχεδιάζεται ένα σκελετό του προγράμματός σας που αποτελείται από τα αντικείμενα uowm, duth, apth.

```
class uowm:
    pass
class duth:
    pass
class apth:
    pass
```

Κάνοντάς το αυτό,μπορείτε να έχετε τις κλάσεις που αποφασίσατε να δημιουργήσετε, κενές, ώστε σιγά – σιγά να υλοποιείτε το σύστημά σας, γνωρίζοντας πάντα όλα όσα έχετε να κάνετε.

Στη συνέχεια παρουσιάζω κάποια προγράμματα για να γίνουν πιο κατανοητές οι έννοιες που αναφέρθηκαν παραπάνω :

1° πρόγραμμα

program1.py - C:/Python33/program1.py

File Edit Format Run Options Windows Help

```
answer = int ( input ( 'Ποιό ειναι το γινόμενο 2 * 5: ' ) )
if ( answer == 10) :
      print ( 'Σωστά!!! ' )
print ( 'Έξοδος από το πρόγραμμα!' )
```

Αν δοκιμάσουμε να εκτελέσουμε τον παραπάνω κώδικα ,τότε για οποιονδήποτε αριθμό επιτελεί την λειτουργία για την οποία προορίζεται, να πληκτρολογήσεις το σωστό γινόμενο του 2*5 και να σου εμφανιστεί το μήνυμα : «Σωστά!». Αν δεν πληκτρολογήσεις 10 και πληκτρολογήσεις οποιδήποτε άλλον αριθμό θα εμφανιστεί το μήνυμα : «Έξοδος από το πρόγραμμα».

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Ποιό ειναι το γινόμενο 2 * 5: 10
Σωστά!!!
Έξοδος από το πρόγραμμα!
>>> ========== RESTART ==========
>>>
Ποιό ειναι το γινόμενο 2 * 5: 15
Έξοδος από το πρόγραμμα!
>>>
```

Αν όμως αντί για αριθμό δοκιμάσουμε να δώσουμε ως είσοδο ένα οποιοδήποτε γράμμα, τότε παίρνουμε το ακόλουθο μήνυμα :

Python 3.3.2 Shell

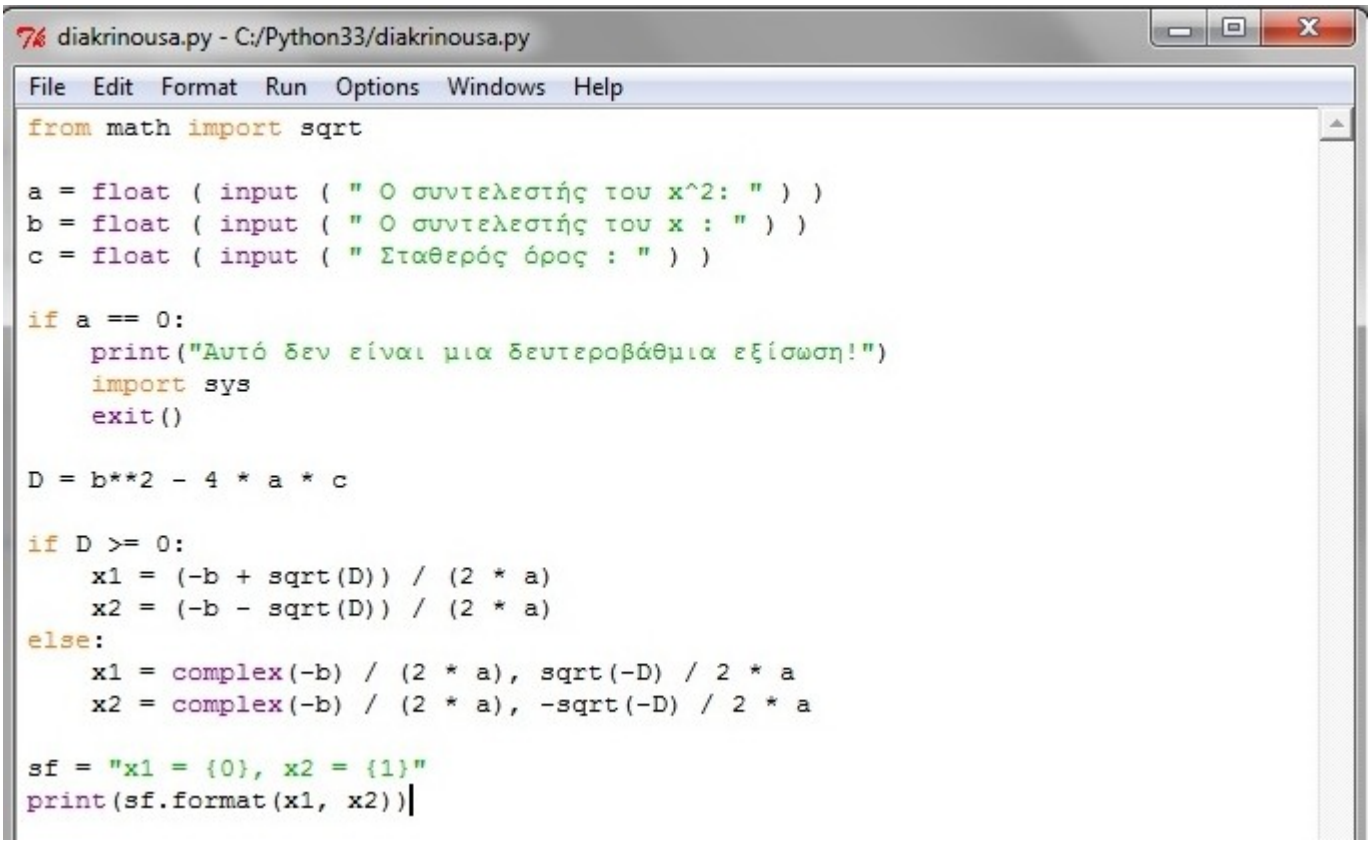
File Edit Shell Debug Options Windows Help

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Ποιό ειναι το γινόμενο 2 * 5: α
Traceback (most recent call last):
  File "C:/Python33/program1.py", line 1, in <module>
    answer = int ( input ( 'Ποιό ειναι το γινόμενο 2 * 5: ' ) )
ValueError: invalid literal for int() with base 10: 'α'
>>>
```

Αυτό συμβαίνει γιατί προσπαθούμε να μετατρέψουμε το γράμμα ‘a’ σε ακέραιο μέσω της συνάρτησης int(),πράγμα που δεν γίνεται, και για αυτό εγείρεται μια εξαίρεση τύπου ValueError.

2ο πρόγραμμα

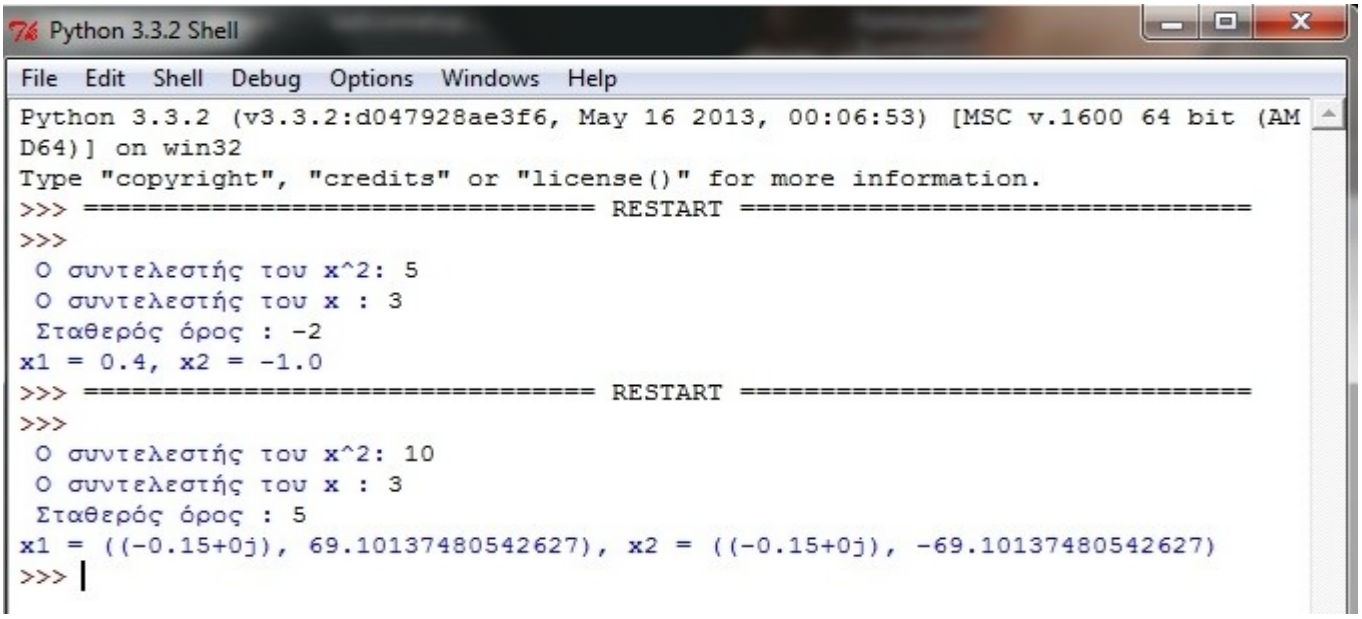
Στο πρόγραμμα που ακολουθεί υπολογίζονται οι λύσεις μιας δευτεροβάθμιας εξίσωσης.

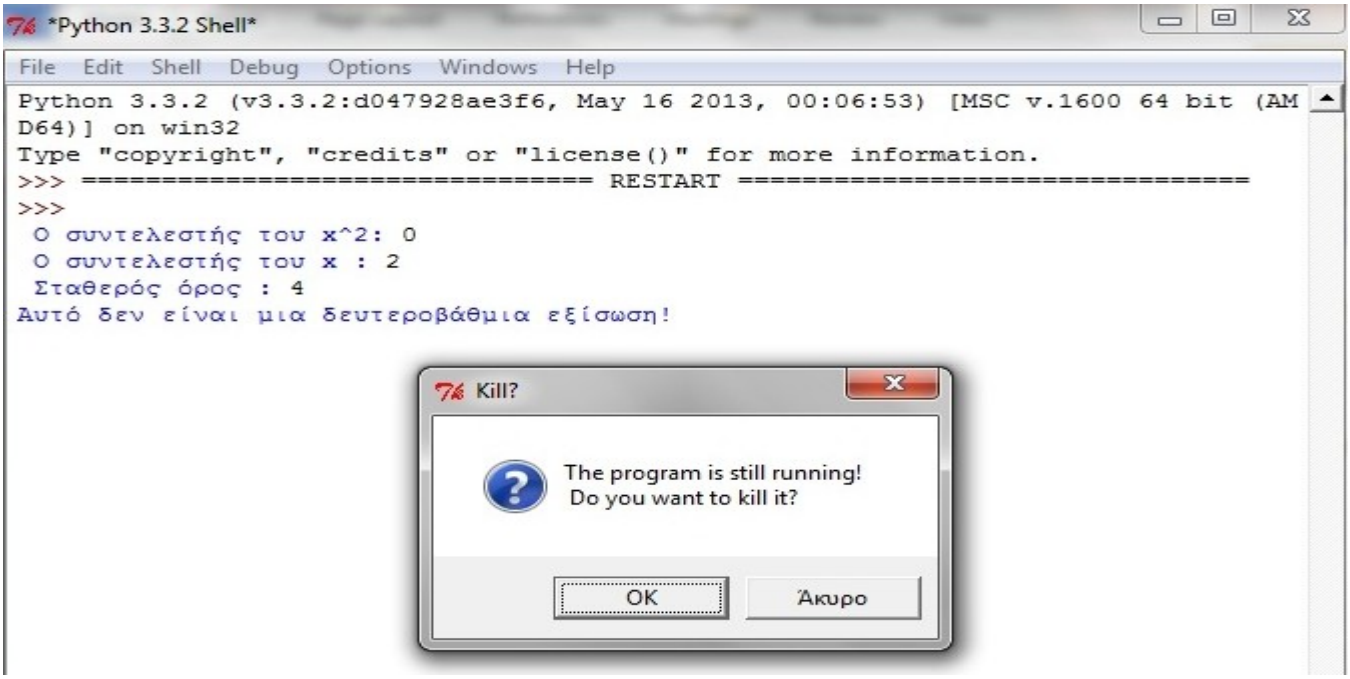


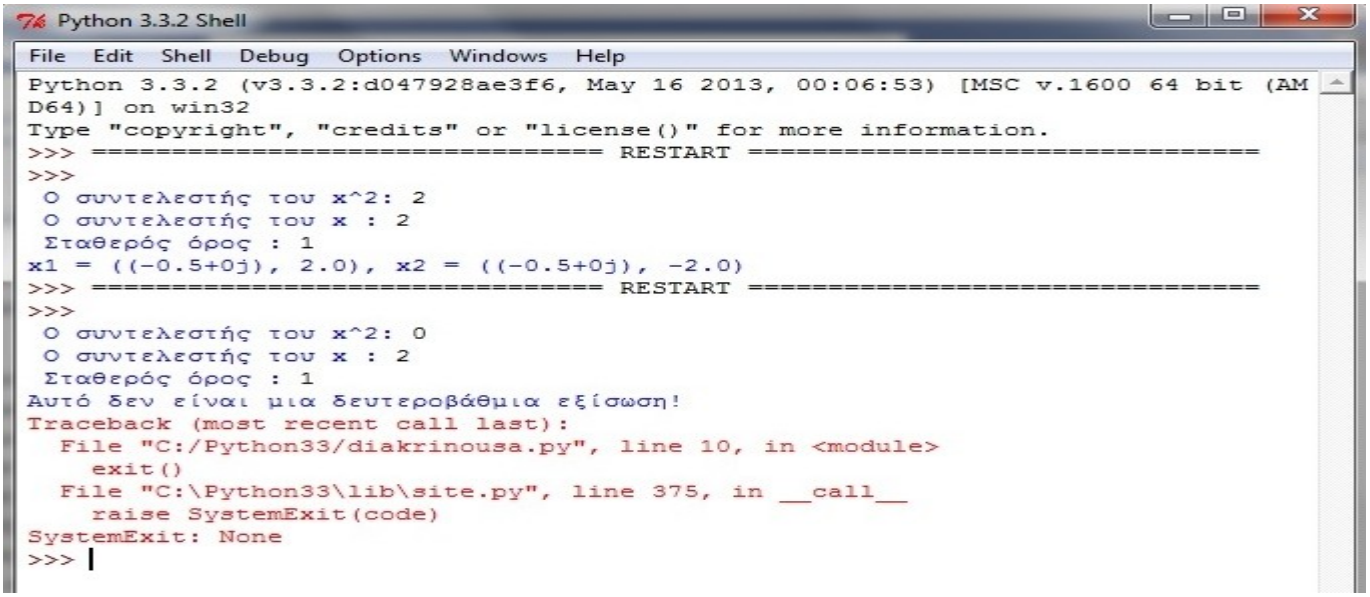
Ανάλογα με το αν η διακρίνουσα είναι θετική ή αρνητική χρησιμοποιούμε μιγαδικούς αριθμούς και στην συνέχεια τυπώνουμε το αποτέλεσμα. Υπενθιμίζεται ότι ο τύπος επίλυσης της δευτεροβάθμιας εξίσωσης είναι :

$$x_{1,2} = \frac{-\beta + \sqrt{\Delta}}{2\alpha}, \text{ όπου } \Delta = \beta^2 - 4\alpha\gamma$$

Αν η διακρίνουσα Δ είναι θετική τότε έχουμε πραγματικές ρίζες (περίπτωση μέσα στο πρώτο μπλοκ κώδικα μετά το if ενώ αλλιώς έχουμε μιγαδικές (περίπτωση μέσα στο μπλοκ else). Στην περίπτωση που η διακρίνουσα είναι 0 έχουμε μια διπλή πραγματική ρίζα, πράγμα που ισχύει και στο αποτέλεσμα του προγάμματός μας αφού δημιουργούνται δύο ίδιες ρίζες με μηδενικό φανταστικό μέρος.

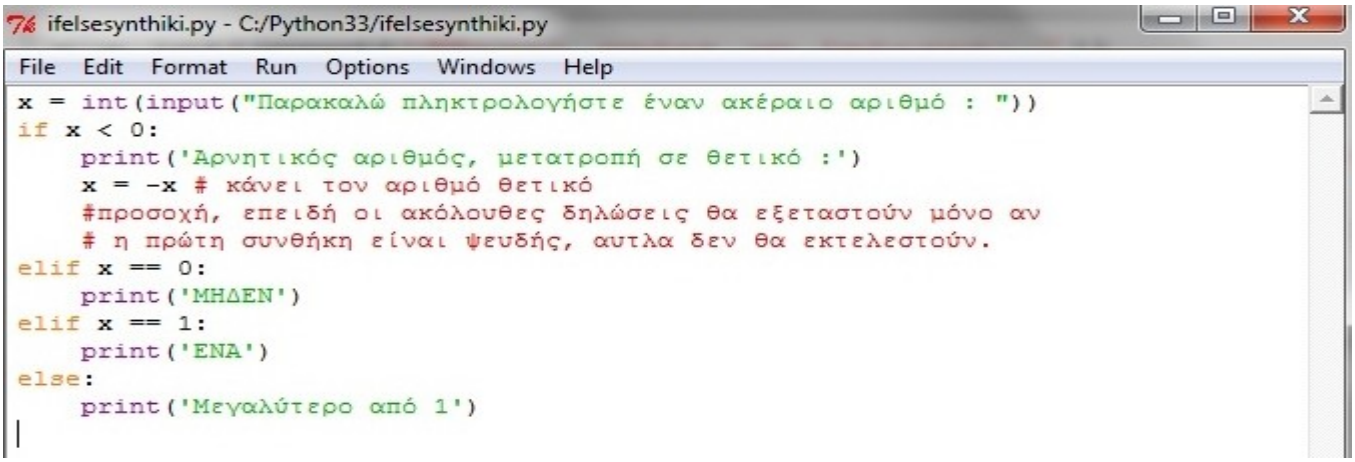

Python 3.3.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Ο συντελεστής του x^2: 5
Ο συντελεστής του x : 3
Σταθερός όρος : -2
x1 = 0.4, x2 = -1.0
>>> ========== RESTART ==========
>>>
Ο συντελεστής του x^2: 10
Ο συντελεστής του x : 3
Σταθερός όρος : 5
x1 = ((-0.15+0j), 69.10137480542627), x2 = ((-0.15+0j), -69.10137480542627)
>>>


Python 3.3.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Ο συντελεστής του x^2: 0
Ο συντελεστής του x : 2
Σταθερός όρος : 4
Αυτό δεν είναι μια δευτεροβάθμια εξίσωση!
Kill?
The program is still running!
Do you want to kill it?
OK
Άκυρο

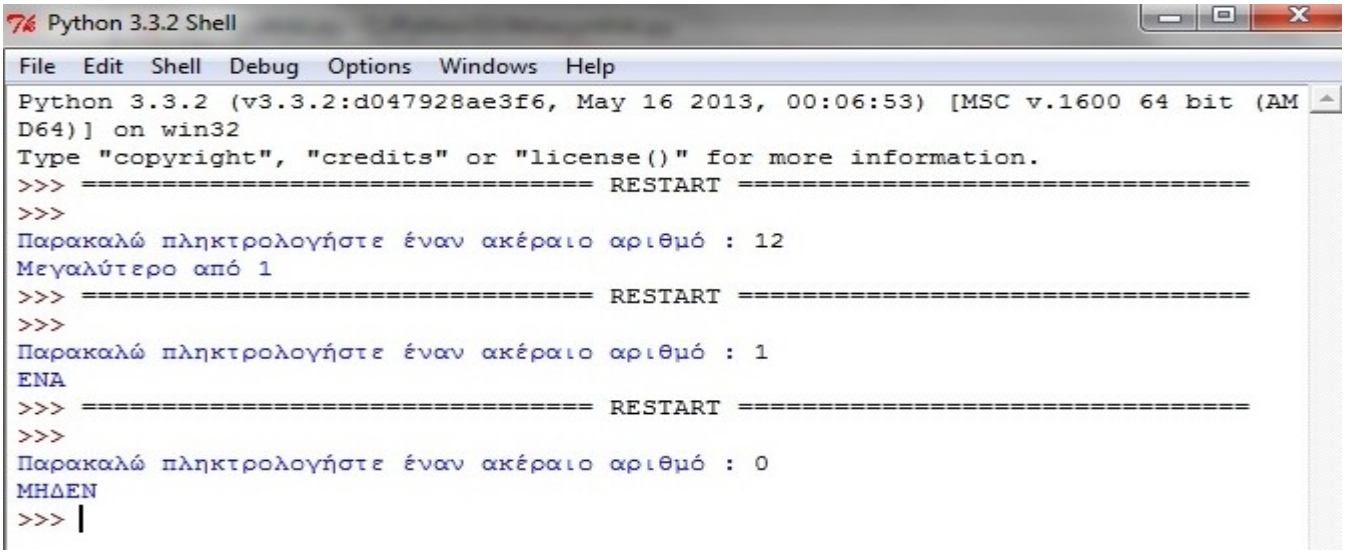

Python 3.3.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Ο συντελεστής του x^2: 2
Ο συντελεστής του x : 2
Σταθερός όρος : 1
x1 = ((-0.5+0j), 2.0), x2 = ((-0.5+0j), -2.0)
>>> ========== RESTART ==========
>>>
Ο συντελεστής του x^2: 0
Ο συντελεστής του x : 2
Σταθερός όρος : 1
Αυτό δεν είναι μια δευτεροβάθμια εξίσωση!
Traceback (most recent call last):
File "C:/Python33/diakrinousa.py", line 10, in <module>
exit()
File "C:\Python33\lib\site.py", line 375, in __call__
raise SystemExit(code)
SystemExit: None
>>>

3ο πρόγραμμα

Στην Python , δεν υπάρχει η δήλωση switch όπου αναλόγως με μια παράμετρο εκτελείται μια λειτουργία από k διαφορετικές, όπου k είναι το πλήθος των διαφορετικών περιπτώσεων που έχουμε συμπεριλάβει στην switch. Η δήλωση switch είναι γνωστή από γλώσσες προγραμματισμού όπως C/C++ ή Java. Στην Python, ο προτεινόμενος τρόπος είναι η χρήση της δομής if…elif…else…Ένα απλό παράδειγμα αυτής της χρήσης είναι το παρακάτω :



και εκτελώντας το παραπάνω πρόγραμμα προκύπτει :

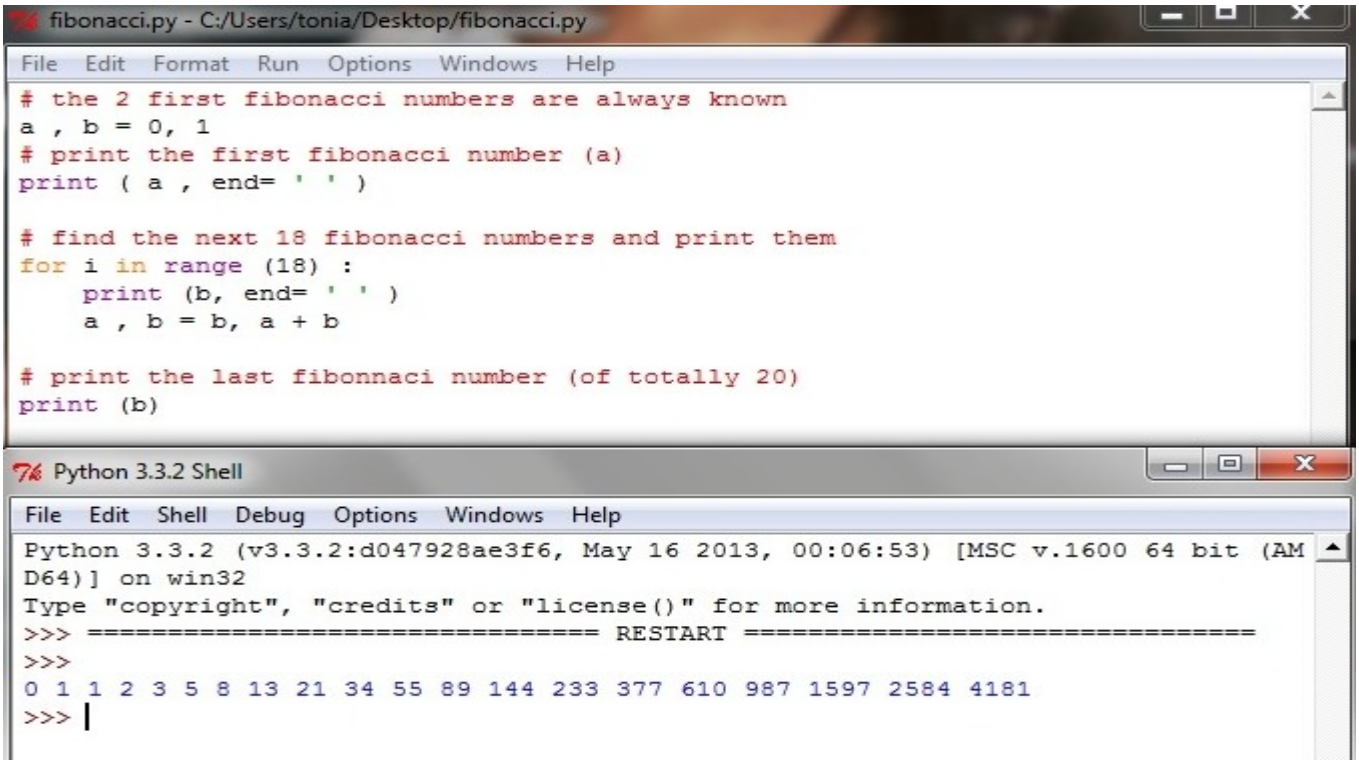


Αυτός ο τρόπος προτείνεται όταν τα elif που χρησιμοποιούνται είναι σχετικά λίγα, κρατώντας έτσι τον κώδικα ευανάγνωστο.

4ο παράδειγμα

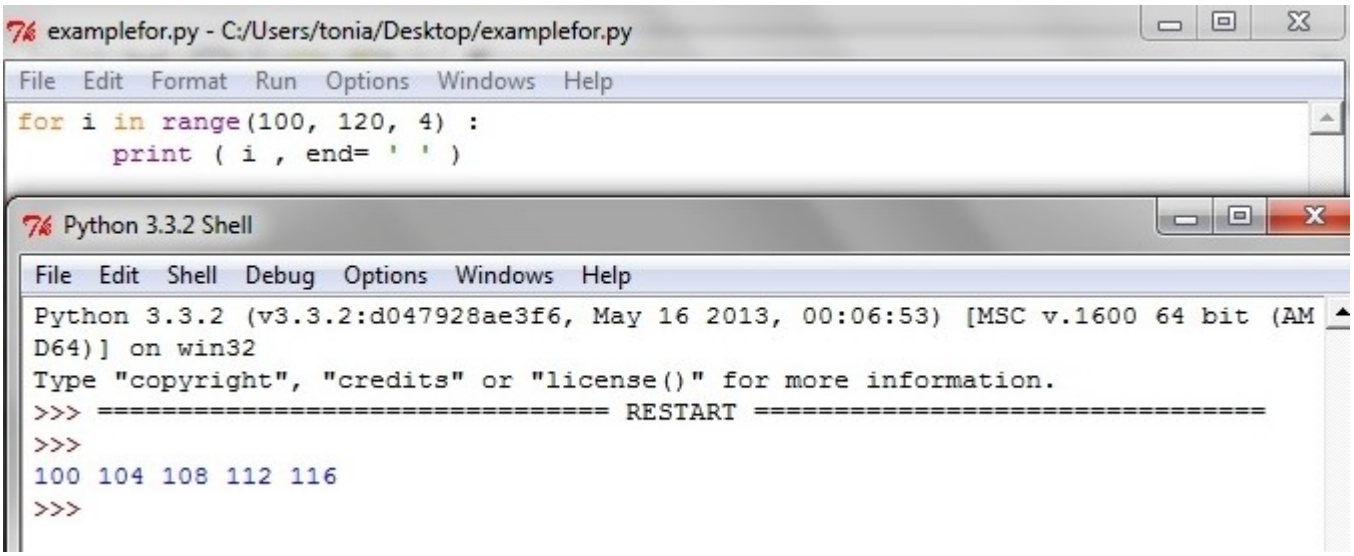
Οι βρόγχοι for εκτελούνται για συγκεκριμένο πλήθος φορών. Για την δημιουργία τους χρησιμοποιείται η συνάρτηση range() που αναφέραμε παραπάνω. Έτσι, όσο η συνάρτηση range() επιστρέφει ένα καινούργιο αντικείμενο, οι εντολές στο σώμα του βρόγχου συνεχίζουν να εκτελούνται.

Ως παράδειγμα θα παραθέσω τον υπολογισμό των 20 πρώτων αριθμών Fibonacci. Υπενθιμίζεται πως ο τύπος για τον υπολογισμό του n-οστού αριθμού Fibonacci είναι: $F_n = F_{n-1} + F_{n-2}$ όπου $F_0 = 0$ και $F_1 = 1$.



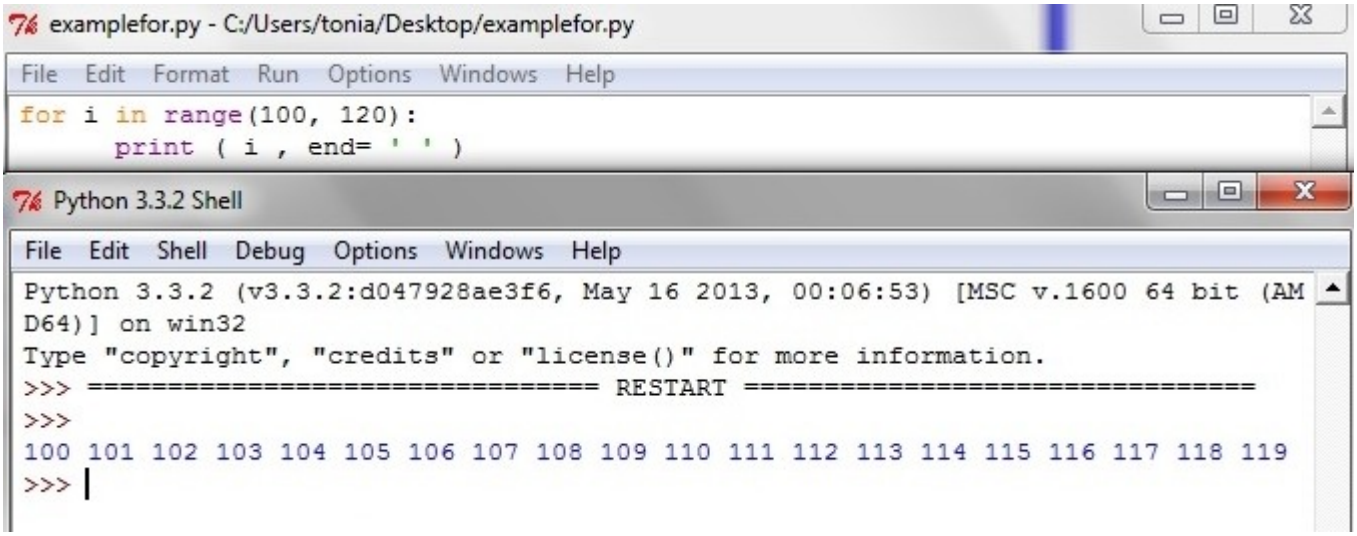
Όπου i θα πάρει διαδοχικά τις τιμές από 0 εως και 17, με αποτέλεσμα να εκτελεστούν οι εντολές στο σώμα βρόγχου 18 φορές. Η συνάρτηση range() δουλεύει μόνο για ακεραίους. Θα μπορούσαμε να παράγουμε με την βοήθεια της range() τους αριθμούς μέσα σε ένα συγκεκριμένο εύρος, ή με ένα συγκεκριμένο βήμα όπως φαίνεται στις περιπτώσεις που παραθέτω αμέσως κάτω.

1η περίπτωση



Σε αυτή την περίπτωση, τυπώνουμε τους αριθμούς στο διάστημα [100,120) με βήμα ανά 4.

2η περίπτωση



Σε αυτή την περίπτωση, τυπώνουμε όλους τους αριθμούς στο διάστημα [100,120).

Παρατήρηση → Αν είχαμε `for i in range(100,120,2)` θα τυπώναμε τους αριθμούς στο διάστημα [100,120) , αλλά με βήμα ανα 2, συνεπώς θα τυπόνονταν **μόνο οι άρτιοι αριθμοί** μέσα σε αυτό το διάστημα.

Χρήση λεξικού

Ένας άλλος τρόπος να εξομοιώσουμε την λειτουργία μιας δήλωσης switch, είναι με την χρήση λεξικών. Τα λεξικά μας επιτρέπουν να αντιστοιχήσουμε σε ένα κλειδί τους (key) μια μεταβλητή. Χρησιμοποιώντας λοιπόν ως κλειδιά τις επιλογές που θέλουμε να έχουμε, και αντιστοιχίζοντάς τα με τις συναρτήσεις που θέλουμε να χρησιμοποιήσουμε, παίρνουμε λειτουργικότητα εφάμιλλη με την επιθυμητή.

Ένα λεξικό είναι κάτι σαν ένα βιβλίο επαφών στο οποίο βρίσκετε την διεύθυνση ή τις πληροφορίες επικοινωνίας κάποιου, γνωρίζονται μόνο το όνομά του. Με άλλα λόγια, υπάρχει μια συσχέτιση μεταξύ του κλειδιού (του ονόματος) και της τιμής (των λεπτομερειών). **Σημειώστε ότι το κλειδί πρέπει να είναι μοναδικό, αλλιώς δεν θα μπορούσατε να βρείτε τις σωστές πληροφορίες επικοινωνίας κάποιου, αν είχατε δύο άτομα με το ίδιο ακριβώς όνομα.**

Σημειώστε ότι μπορείτε να χρησιμοποιείτε μόνο αμετάβλητα αντικείμενα (για παράδειγμα συμβολοσειρές) ως κλειδιά ενός λεξικού, αλλά μπορείτε να χρησιμοποιείτε είτε αμετάβλητα είτε μεταβλητά αντικείμενα για τις τιμές του λεξικού.

Αυτό με απλά λόγια σημαίνει ότι θα πρέπει να χρησιμοποιείτε μόνο απλά αντικείμενα ως κλειδιά

Τα ζεύγη κλειδιών και τιμών σε ένα λεξικό ορίζονται με τη σύνταξη :

```
d = {key1 : value1, key2 : value2 }
```

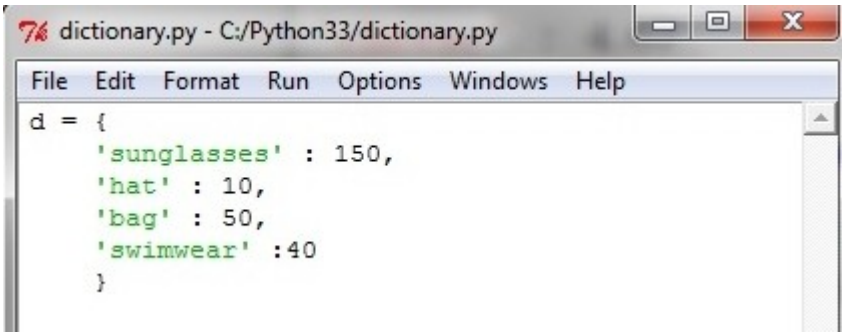
Σημειώστε ότι τα ζεύγη κλειδιών/τιμών χωρίζονται με κόμμα, και όλα μαζί περικλείονται σε ένα ζεύγος αγκίστρων (δηλάδη { και }).

Θυμηθείτε ότι τα ζεύγη κλειδιών/τιμών σε ένα λεξικό δεν μπαίνουν σε κανενός είδους σειρά. Αν θέλετε να είναι σε συγκεκριμένη σειρά, θα πρέπει να τα ταξινομήσετε προτού τα χρησιμοποιήσετε.

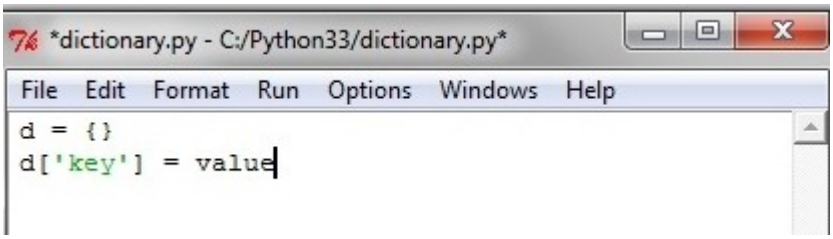
Τα λεξικά που θα χρησιμοποιήσετε είναι περιστάσεις/αντικείμενα της κλάσης dict.

Δημιουργία λεξικού

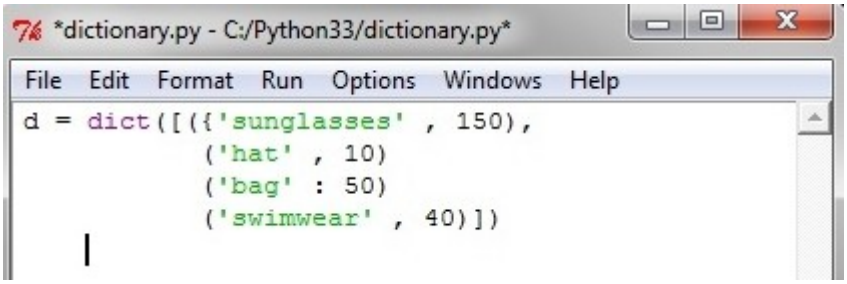
Μπορούμε να αναθέσουμε κάποιες τιμές στο λεξικό με την δημιουργία του. Για παράδειγμα :



Ακόμα κι αν έχουμε ήδη δημιουργήσει ένα λεξικό μπορούμε εύκολα να αναθέτουμε τιμές σε αυτό. Στο παράδειγμα που ακολουθεί δημιουργούμε ένα άδειο λεξικό και στην συνέχεια προσθέτουμε μια τιμή.



Μπορούμε επίσης να δμιουργήσουμε ένα λεξικό μέσω της μεθόδου dict().

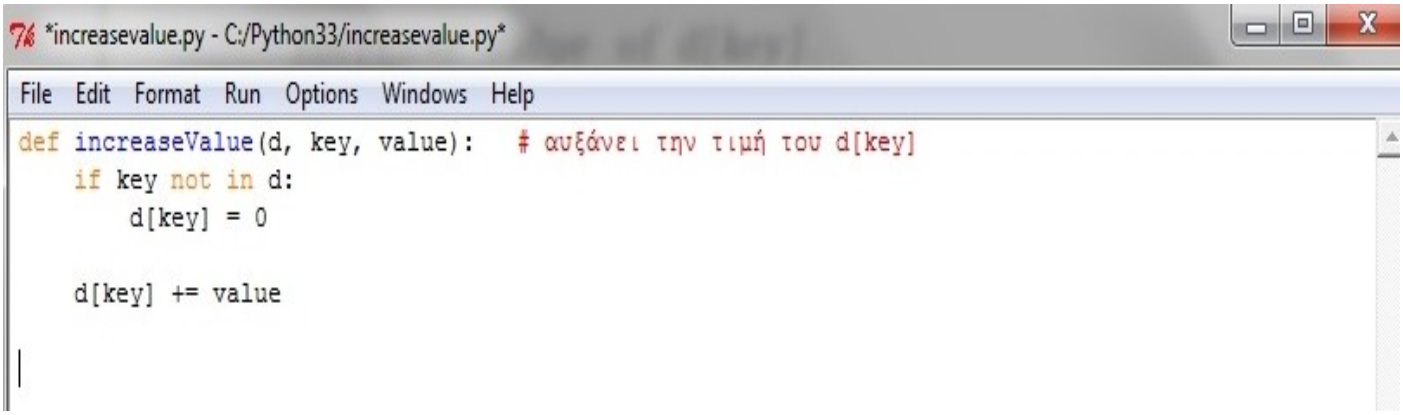


Λειτουργίες σε λεξικό

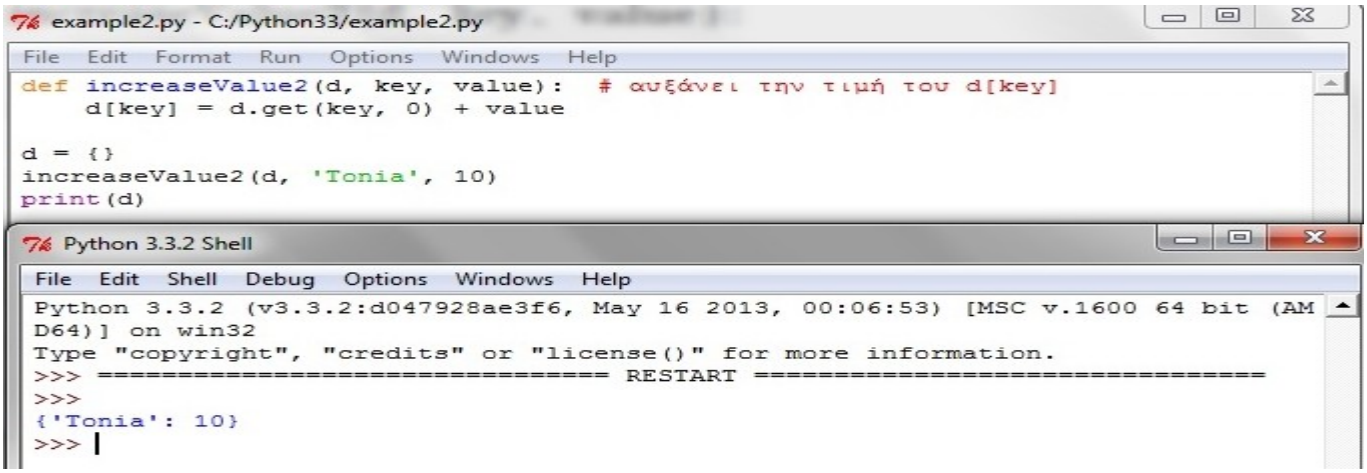
- ✓ Διαγραφή : Χρησιμοποιούμε το del. Για παράδειγμα η εντολή `del d['swimwear']` διαγράφει την αντίστοιχη εγγραφή από το παραπάνω λεξικό.
- ✓ Μέγεθος : Η ανάκτηση του μεγέθους ενός λεξικού (αριθμός ζευγαριών κλειδί/τιμή) ανακτάται μέσω της μεθόδου `len(d)`.
- ✓ Κλειδιά : Για να βρούμε τα κλειδιά ενός λεξικού(χρήσιμο σε βρόγχους) χρησιμοποιούμε την μέθοδο `d.keys()`.
- ✓ Τιμές : Αντίστοιχα για τις τιμές της μεθόδου `d.values()`.
- ✓ Κλειδί/Τιμή : Για να ανακτήσουμε όλα τα ζευγάρια κλειδί/τιμή από ένα συγκεκριμένο λεξικό `d.times()` η οποία και μας επιστρέφει πλειάδες από δυο στοιχεία , όπου το πρώτο είναι το κλειδί ακολουθούμενο από την αντίστοιχη τιμή.

Μια λειτουργία που συναντιέται αρκετά συχνά στην πράξη, είναι να θέλουμε να προσθέσουμε μια τιμή σε μια συγκεκριμένη θέση του λεξικού. Πολλές φορές θέλουμε να δούμε αν το κλειδί του λεξικού υπάρχει ήδη, να ανακτήσουμε την ήδη υπάρχουσα τιμή και να την ενημερώσουμε με βάση μια συνάρτηση, ενώ αν δεν υπάρχει το συγκεκριμένο κλειδί, να το δημιουργήσουμε και να τοποθετήσουμε την τιμή που θέλουμε. Παρακάτω παραθέτω δύο τρόπου για την υλοποίηση αυτής της λειτουργίας. Τονίζεται ότι και οι δύο τρόποι έχουν ακριβώς το ίδιο αποτέλεσμα.

1η συνάρτηση :



2η συνάρτηση :

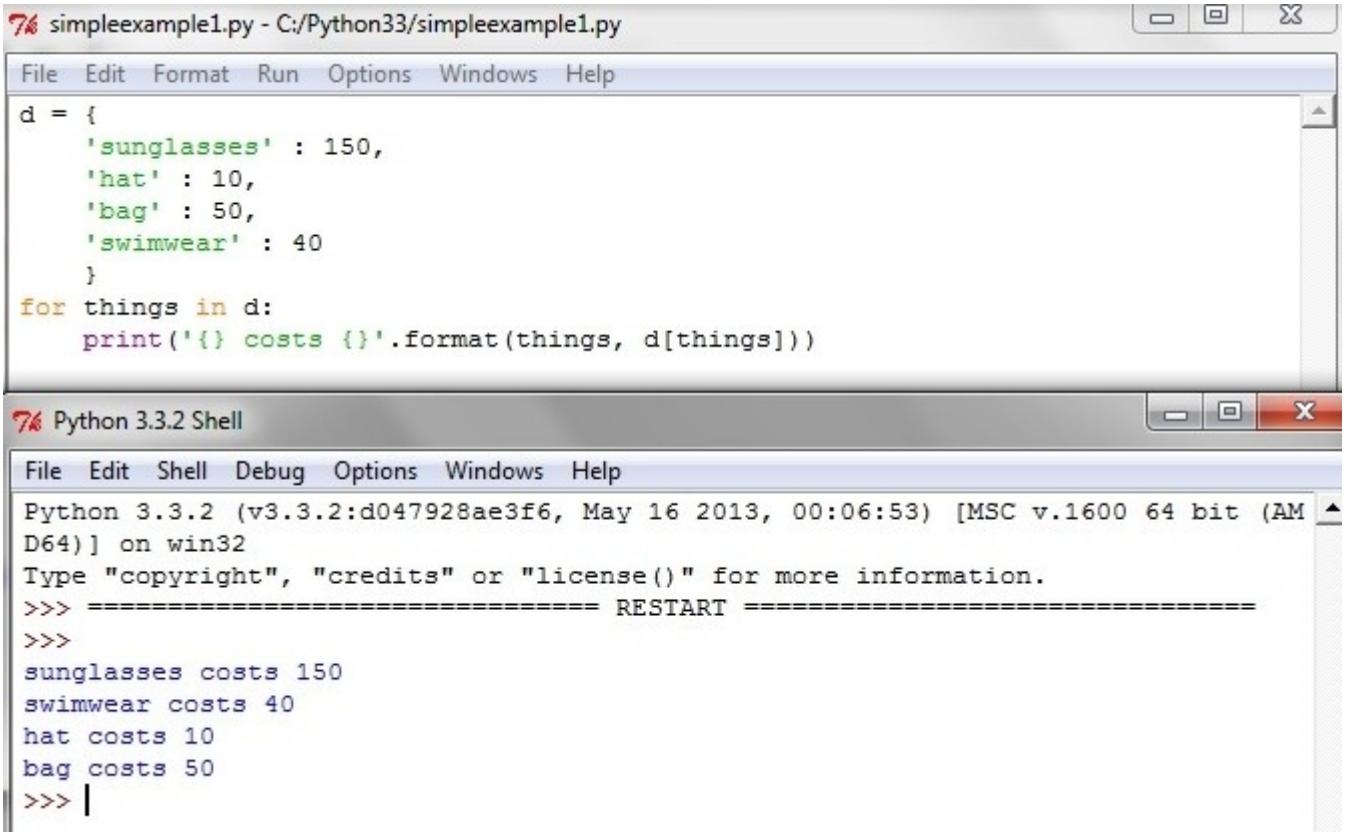


- Η πρώτη συνάρτηση κοιτάζει αν υπάρχει το κλειδί μέσα στο λεξικό. Αν δεν υπάρχει, το δημιουργεί και του δίνει μια αρχική τιμή (μηδέν στην περίπτωσή μας). Στη συνέχεια προσθέτει στην τιμή του λεξικού που ανακτά την τιμή που έχουμε περάσει στο όρισμα της συνάρτησης (value).
- Η δεύτερη συνάρτηση επιτυγχάνει το ίδιο αποτέλεσμα με έναν πιο κομψό τρόπο. Χρησιμοποιούμε την μέθοδο get που εφαρμόζεται σε ένα λεξικό. Αυτή μας επιστρέφει την τιμή του d[key] ή την τιμή που προσδιορίζουμε με τον δεύτερο ορισμό εφόσον δεν υπάρχει αυτό το κλειδί στο λεξικό Στην συνέχεια προσθέτουμε την τιμή της value στην ανακτηθείσα τιμή και το αποτέλεσμα το εκχωρούμε στο d[key] έχοντας κάνει την ενημέρωση που επιθυμούσαμε.

Διάτρεξη τιμών

Η διάτρεξη των τιμών σε ένα λεξικό γίνεται όπως στις λίστες. Μόνο που εδώ πρέπει να προσέξουμε ότι δεν είναι εγγυημένη η σειρά με την οποία θα διατρεχθούν τα στοιχεία του λεξικού, και για αυτό τον λόγο δεν πρέπει να βασιζόμαστε σε αυτή.

Ακολουθεί ένα σχετικό παράδειγμα :



Αναφορά και Τροποποίηση

Τα λεξικα είναι τροποποιήσιμα αντικείμενα (mutable). Για αυτό τον λόγο απαιτείται προσοχή όταν τροποποιούμε ένα λεξικό καθώς όλες οι αναφορές σε αυτό το λεξικό τροποποιούνται επίσης. Αν επιθυμούμε την αποφυγή μίας τέτοιας συμπεριφοράς, τότε πρεπει να χρησιμοποιούμε την μέθοδο copy().

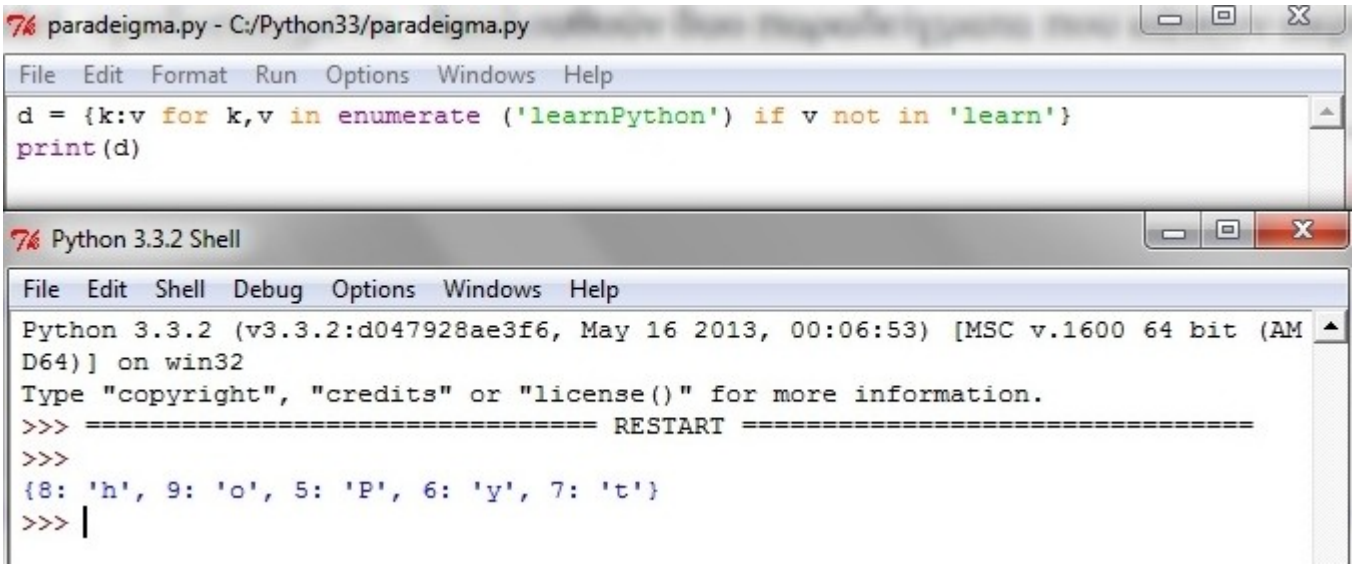
```
d['a'] = 123

c = d

c['a'] = 1821

print(d['a'])
```

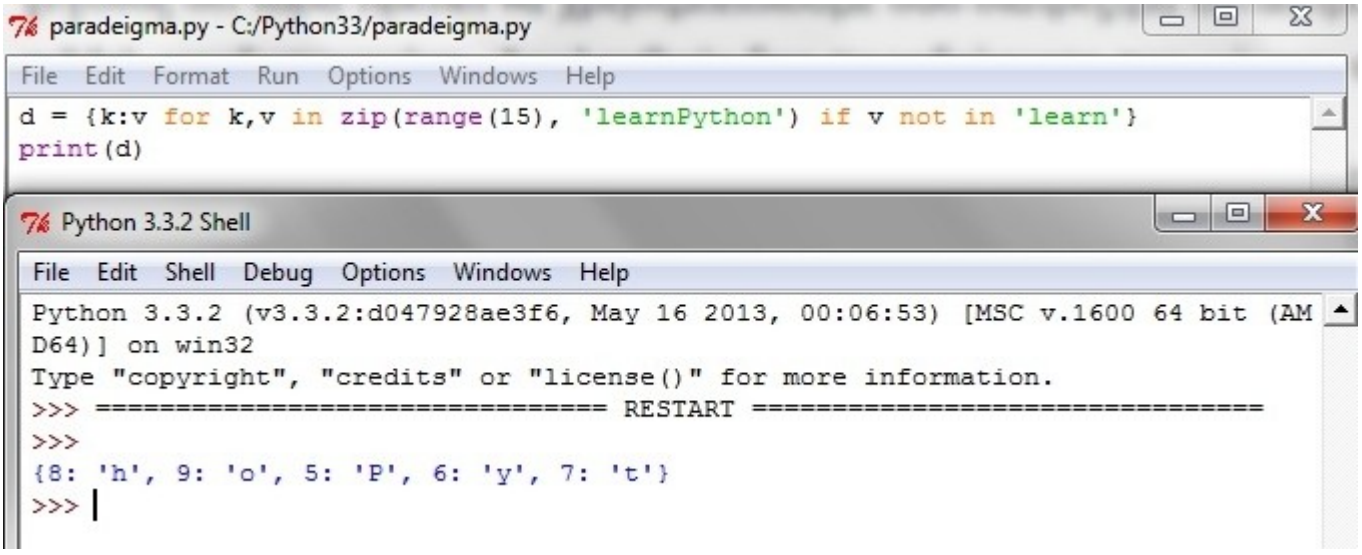
Σε αντιδιαστολή με :

```
d = {}

d['a'] = 123

c = d.copy

c['a'] = 1821

print(d['a'])
```

Κατανοήσεις λεξικού (Dict comprehension)

Αντίστοιχα με τις κατανοήσεις λίστας, υπάρχουν και κατανοήσεις λεξικών. Η διαφορά, πέρα από την προφανή συντακτική διαφορά με τις {} έγκειται στο γεγονός ότι τώρα πρέπει να χρησιμοποιούμε δυο διατρέξιμα αντικείμενα (iterable) , ομοδοποιημένα. Ακολουθούν δυο παραδείγματα που κάνουν ακρβώς το ίδιο πράγμα.





Αν προσέξετε στο αποτέλεσμα που παρουσιάζεται, δεν τυπώνονται σε σειρά τα γράμματα της λέξης Python. Αυτό συμβαίνει γιατί η δομή του λεξικού δεν εγγυάται κάποια σειρά. Επίσης , πρέπει να προσέξουμε πως η διάτρεξη σταματάει όταν τελειώσουν τα στοιχεία ενός από όλα τα αντικείμενα που διατρέχουμε.

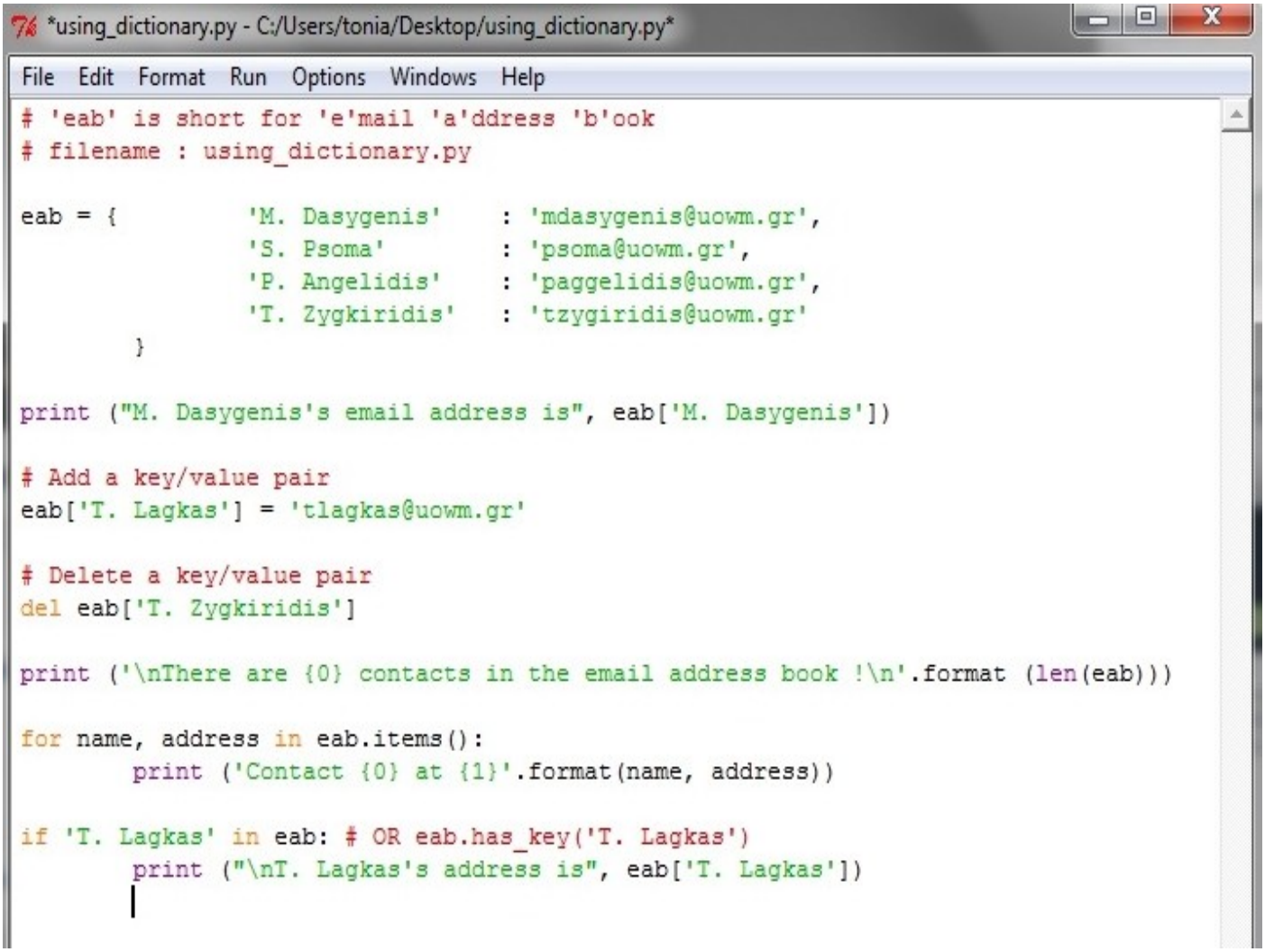
Ταξινομημένο Λεξικό

Η ευρέως χρησιμοποιούμενη δομή ελέγχου του λεξικού, δεν εξασφαλίζει κάποια σειρά στα ζευγάρια κλειδί/τιμή που αποθηκεύονται. Έτσι, δυσχεραίνεται η χρήση λεξικών σαν μέσα αποθήκευσης σε ορισμένες περιπτώσεις. Ορισμένες δυναμικές γλώσσες μπορούν και εγγυώνται μια συγκεκριμένη σειρά προσπέλασης των στοιχείων στις αντίστοιχες δομές. Η σειρά καθορίζεται από τον χρόνο όπου εισήχθηκε κάθε κλειδί . Καινούργια κλειδιά τοποθετούνται στο τέλος του λεξικού. Ωστόσο, κλειδιά στα οποία ανανεώθηκε η τιμή τους, δεν αλλάζουν θέση.

Η δομή του ταξινομημένου λεξικού είναι ουσιαστικά ένα λεξικό (με την έννοια ότι παρέχει την ίδια λειτουργικότητα) , μόνο που έχει και την ιδιότητα που περιγράφτηκε παραπάνω.

➔ Στη συνέχεια, παραθέτω ένα παράδειγμα προγράμματος με τη χρήση λεξικού:

Πρώτα δημιουργείται το λεξικό eab (email address book) το οποίο περιλαμβάνει τις ηλεκτρονικές διευθύνσεις καθηγητών του τμήματός μας. Στη συνέχεια, η πρόσβαση στα ζεύγη κλειδιών/τιμών γίνεται με τη χρήση του τελεστή ευρετηρίασης.



Δημιουργούμε το λεξικό eab (email address book). Εισάγουμε ζευγάρια κλειδί-τιμή καθορίζοντας το κλειδί, χρησιμοποιώντας τον τελεστή ευρετηρίασης (indexing operator) .

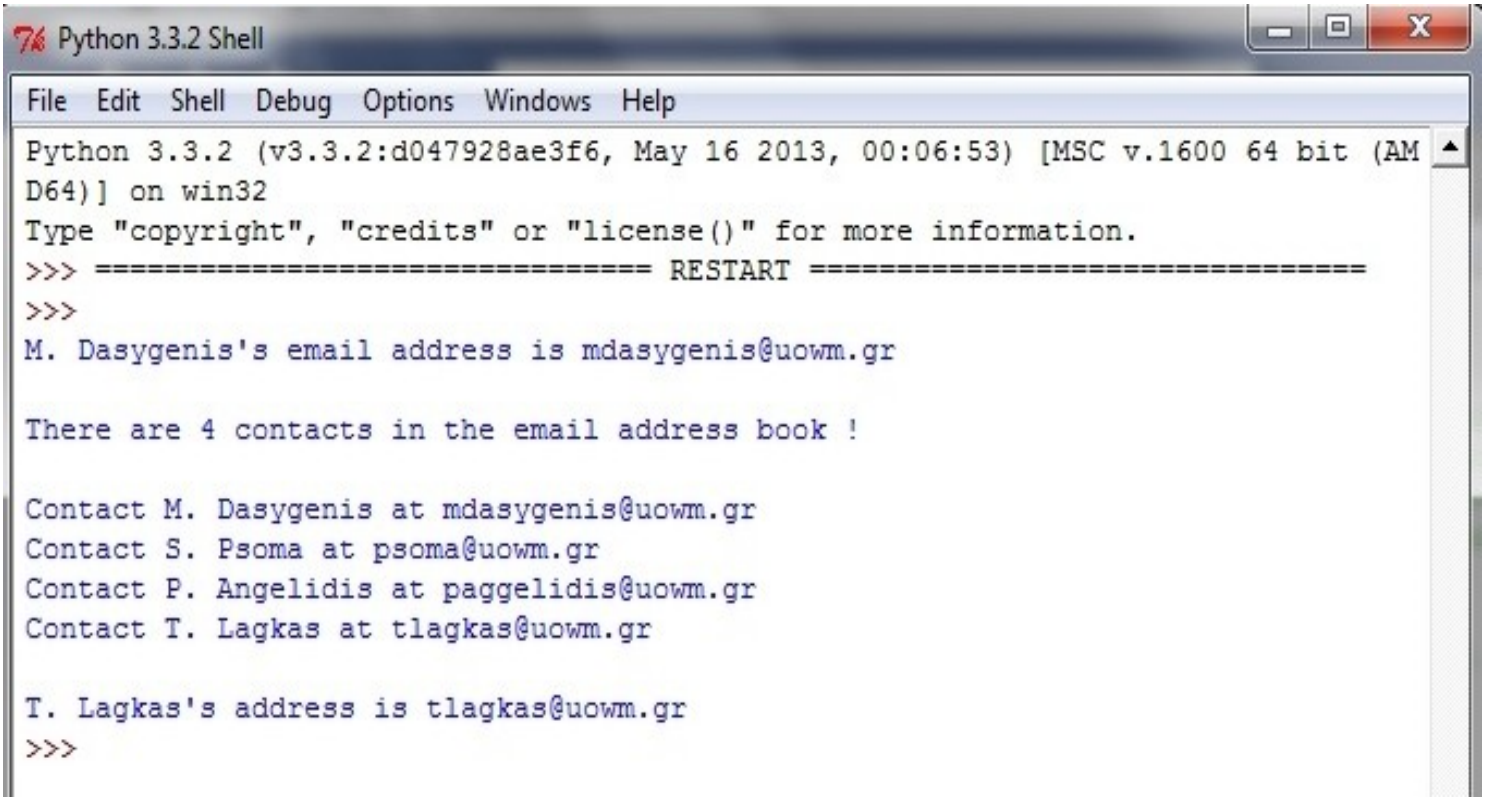
Η διαγραφή ζευγών κλειδιών/τιμών γίνεται με την εντολή del, προσδιορίζοντας το λεξικό και τον τελεστή ευρετηρίασης για το κλειδί που θα αφαιρεθεί. Δεν χρειάζεται να οριστεί η τιμή που αντιστοιχεί στο κλειδί που θα οριστεί.

Έπειτα, εισάγουμε κάθε ζευγάρι κλειδί-τιμή του λεξικού χρησιμοποιώντας τη μέθοδο items του λεξικού, η οποία επιστρέφει μια λίστα πλειάδων, όπου κάθε πλειάδα περιέχει ένα ζευγάρι στοιχείων -το κλειδί ακολουθούμενο από την τιμή. Ανακτούμε αυτό το ζευγάρι και το εκχωρούμε στις μεταβλητές name (ονομασία) και address (διεύθυνση) αντιστοίχως για κάθε ζευγάρι, χρησιμοποιώντας το βρόχο for..in και μετά τυπώνει αυτές τις τιμές στην πλοκάδα for.

Μπορούμε να προσθέσουμε νέα ζευγάρια κλειδί-τιμή, απλά χρησιμοποιώντας τον τελεστή ευρετηρίασης για να εισάγουμε ένα κλειδί και να εκχωρήσουμε σ' αυτό μια τιμή, όπως έχουμε κάνει για το T. Lagkas στην ανωτέρω περίπτωση.

Μπορούμε να ελέγξουμε εάν ένα ζευγάρι κλειδί-τιμή υπάρχει, χρησιμοποιώντας τον τελεστή in, ή ακόμα και τη μέθοδο has_key της κλάσης dict. Μπορείτε να δείτε την τεκμηρίωση για ολόκληρη τη λίστα των μεθόδων της κλάσης dict χρησιμοποιώντας την εντολή **help(dict).**

Προκύπτει με εκτέλεση του προγράμματος :



Συναρτήσεις

Ας ξεκινήσουμε με κάποιους βασικούς ορισμούς που θα μας βοηθήσουν στην καλύτερη κατανόηση όσων ακολουθήσουν.

Συνάρτηση είναι μια ονομαζόμενη ακολουθία δηλώσεων η οποία πραγματοποιεί ένα συγκεκριμένο υπολογισμό. Αποτελούν μια τεχνική αφαίρεσης μέσω της οποίας μπορεί να δοθεί όνομα σε μια σύνθετη λειτουργία και στην συνέχεια να παρέχεται η δυνατότητα να αναφερόμαστε σε αυτή σύνθετη λειτουργία ως μονάδα.

Αν μια συνάρτηση εφαρμόζεται σε αντικείμενο ονομάζεται **μέθοδος**.

Όρισμα είναι μια τιμή η οποία παίρνεται από το πρόγραμμα στην κλήση της συνάρτησης. Μπορεί να χαρακτηριστεί και ως είσοδος της συνάρτησης από το πρόγραμμα από το οποίο καλείται. Μια συνάρτηση μπορεί αν επιδέχεται πολλά ορίσματα.

Επιστρεφόμενη τιμή είναι η τιμή η οποία επιστρέφεται από την συνάρτηση στο περιβάλλον που την κάλεσε μέσω μιας δήλωσης return. Όταν συναντάται η δήλωση return ,τερματίζεται η εκτέλεση της συνάρτησης και η ποή εκτέλεσης επιστρέφει στο περιβάλλον από όπου καλέστηκε!

Οι συναρτήσεις μας επιτρέπουν την ομαδοποίηση κώδικα που επιτελεί μια συγκεκριμένη λειτουργία και, κατά συνέπεια, την ευκολότερη επαναχρησιμοποίησή του. Επιπλέον, επιτρέπουν τον σχεδιασμό του τελικού συστήματος λογισμικού σε υψηλότερο επίπεδο αφαίρεσης, όπου η λειτουργικότητα παρέχεται μέσω συναρτήσεων. Έτσι, στην συνέχεια απομένει η υλοποίηση αυτών, μέχρι του σημείου όπου οι μικρότερες δομικές μονάδες που χρησιμοποιούνται, παρέχονται ήδη από το σύστημα.

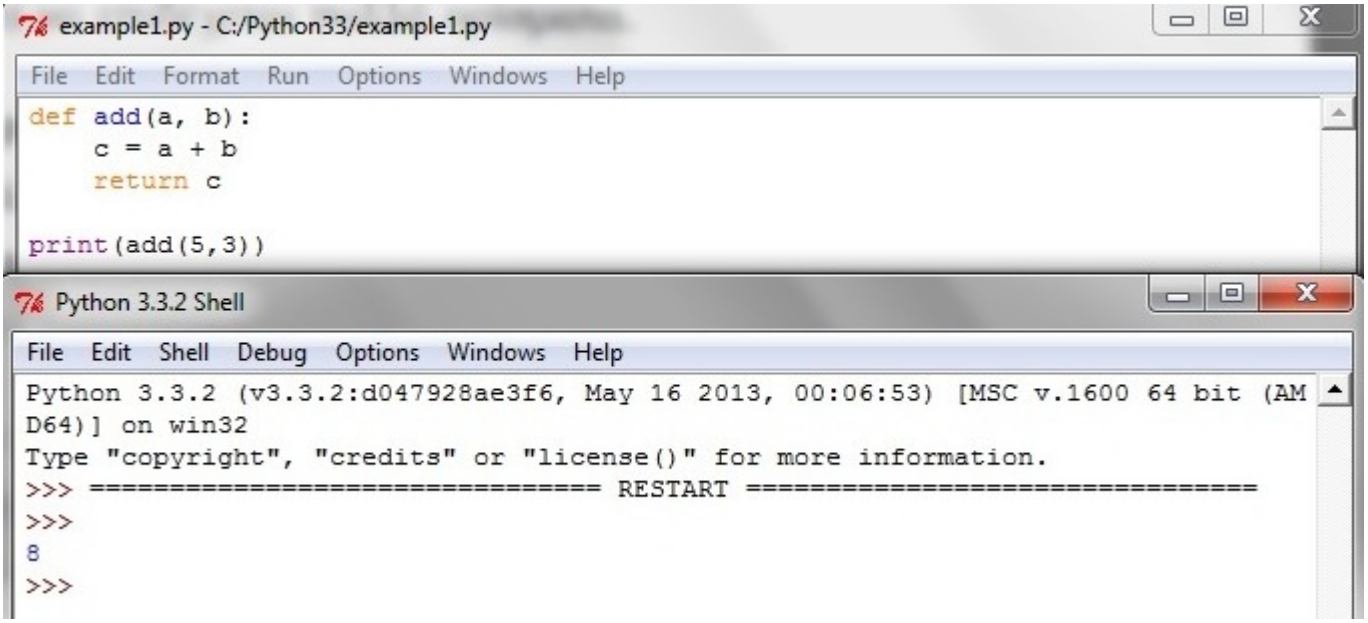
Οι συναρτήσεις είναι επαναχρησιμοποιήσιμα μέρη προγραμμάτων. Σας επιτρέπουν να δίνετε ένα όνομα σε ένα σύνολο εντολών και να τρέχετε εκείνο το σύνολο εντολών χρησιμοιώντας το όνομά τους, οπουδήποτε στο προγραμμά σας και όσες φορές θέλετε. Αυτό είναι γνωστό σαν **κλήση (calling) της συνάρτησης**. Έχουμε ήδη χρησιμοποιήσει πολλές ενσωματωμένες συναρτήσεις όπως τη len και τη range.

Η έννοια των συναρτήσεων είναι πιθανόν το πιο σπουδαίο δομικό στοιχείο οποιουδήποτε μη στοιχειώδους προγράμματος (σε όλες τις γλώσσες προγραμματισμού), γι' αυτό θα διερευνήσουμε διάφορες πτυχές των συναρτήσεων.

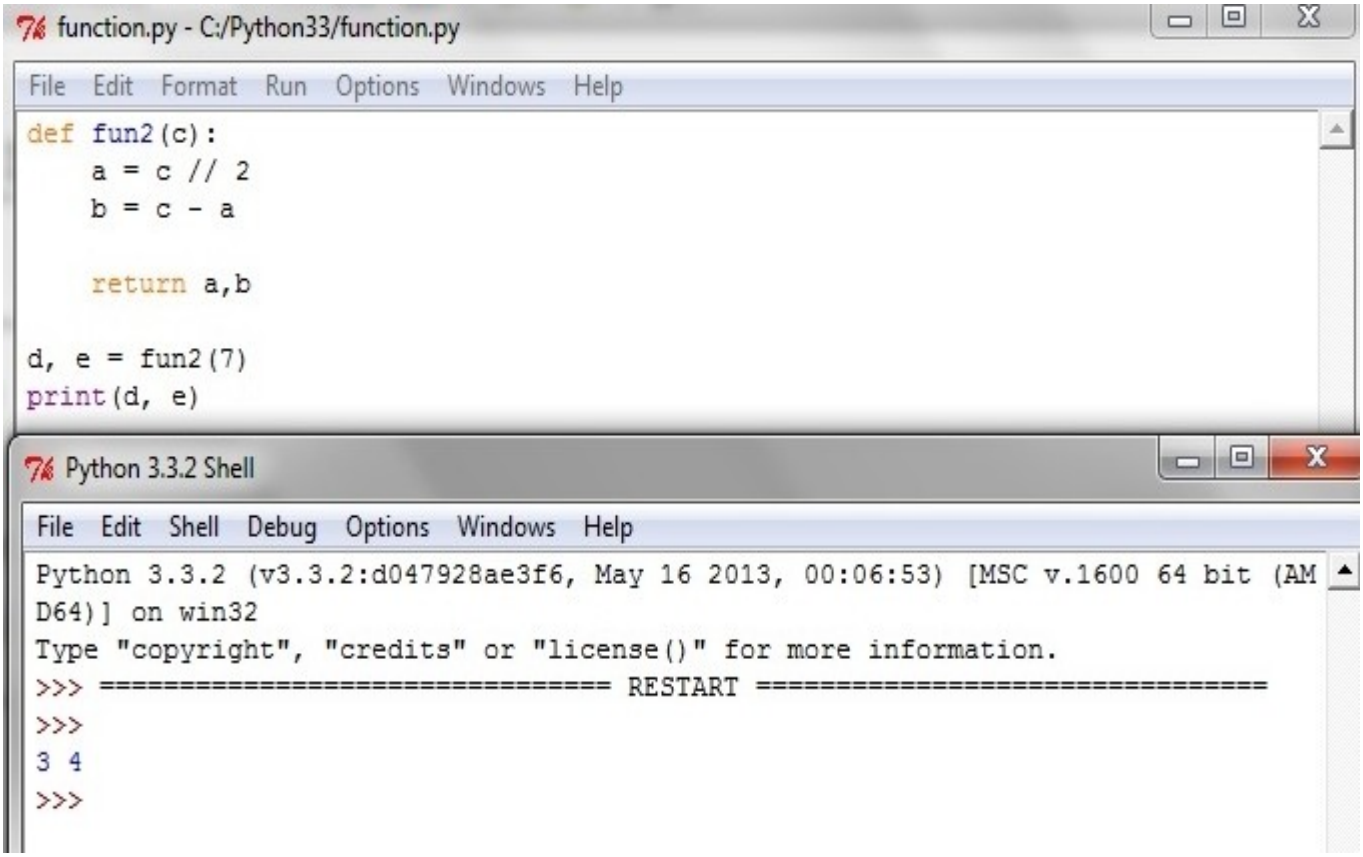
Οι συναρτήσεις ορίζονται χρησιμοποιώντας τη λέξη κλειδί def, μετά την οποία ακολουθεί ένα όνομα που ***ταυτοποιεί*** την εκάστοτε συνάρτηση και κατόπιν

ακολουθεί ένα ζευγάρι παρενθέσεων που μπορούν να περικλείουν μερικά ονόματα μεταβλητών, και η γραμμή τελειώνει με διπλή τελεία (:).

- Στο ακόλουθο παράδειγμα βλέπουμε την δημιουργία μιας συνάρτησης με δύο ορίσματα και την επιστροφή του τελικού της αποτελέσματος. Στο τέλος, εκτυπώνουμε την τιμή που επιστρέφεται καλώντας την συνάρτηση.

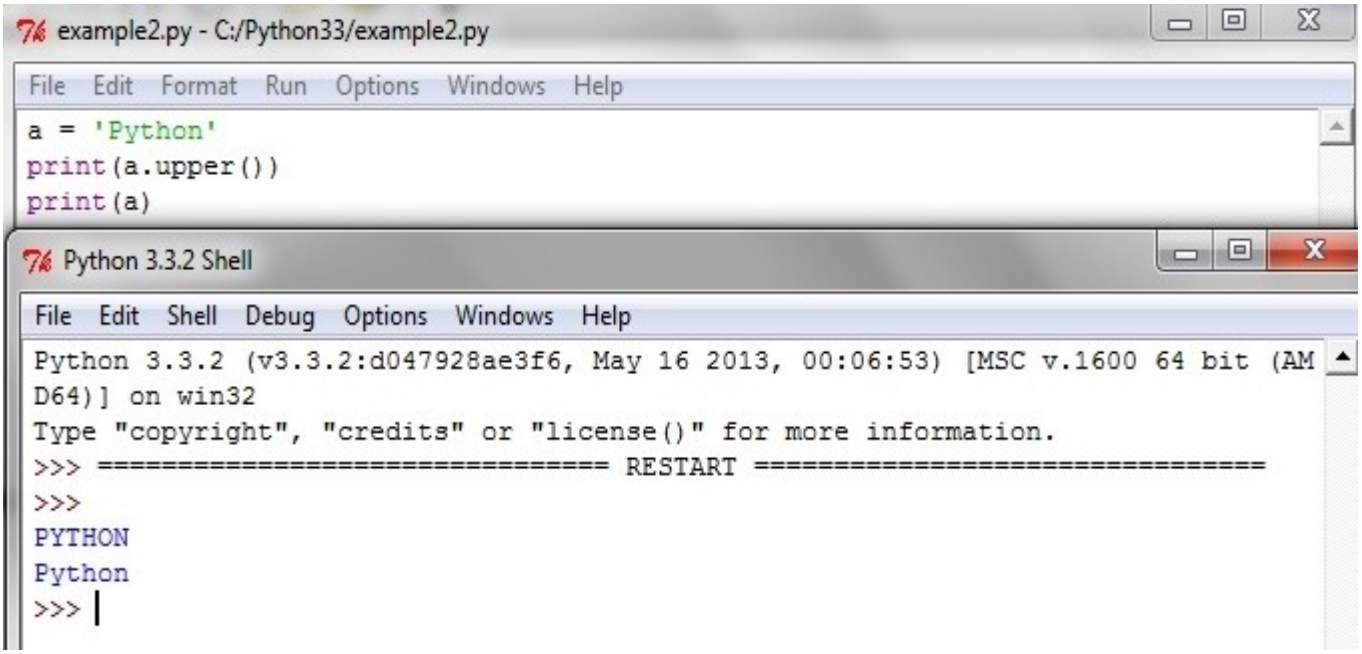


- Μπορούμε να επιστρέψουμε και περισσότερες από μια τιμές κάποιας συνάρτησης.



Αγνές Συναρτήσεις και Συναρτήσεις Τροποποίησης

Αγνές Συναρτήσεις (Pure Functions) ονομάζονται οι συναρτήσεις που δεν προκαλούν καμία αλλαγή στο αντικείμνο στο οποίο καλούνται. Αντιθέτως, **συναρτήσεις τροποποίησης (modifier functions)** καλούνται αυτές που μπορούν να προκαλέσουν αλλαγές στο συγκεκριμένο αντικείμενο. Αυτές οι συναρτήσεις λέμε ότι έχουν και παρενέργειες (side effects), καθώς πέρα από την διεργασία που επιτελούν, μεταβάλουν και την τιμή του αντικειμένου στο οποίο κλήθηκαν.

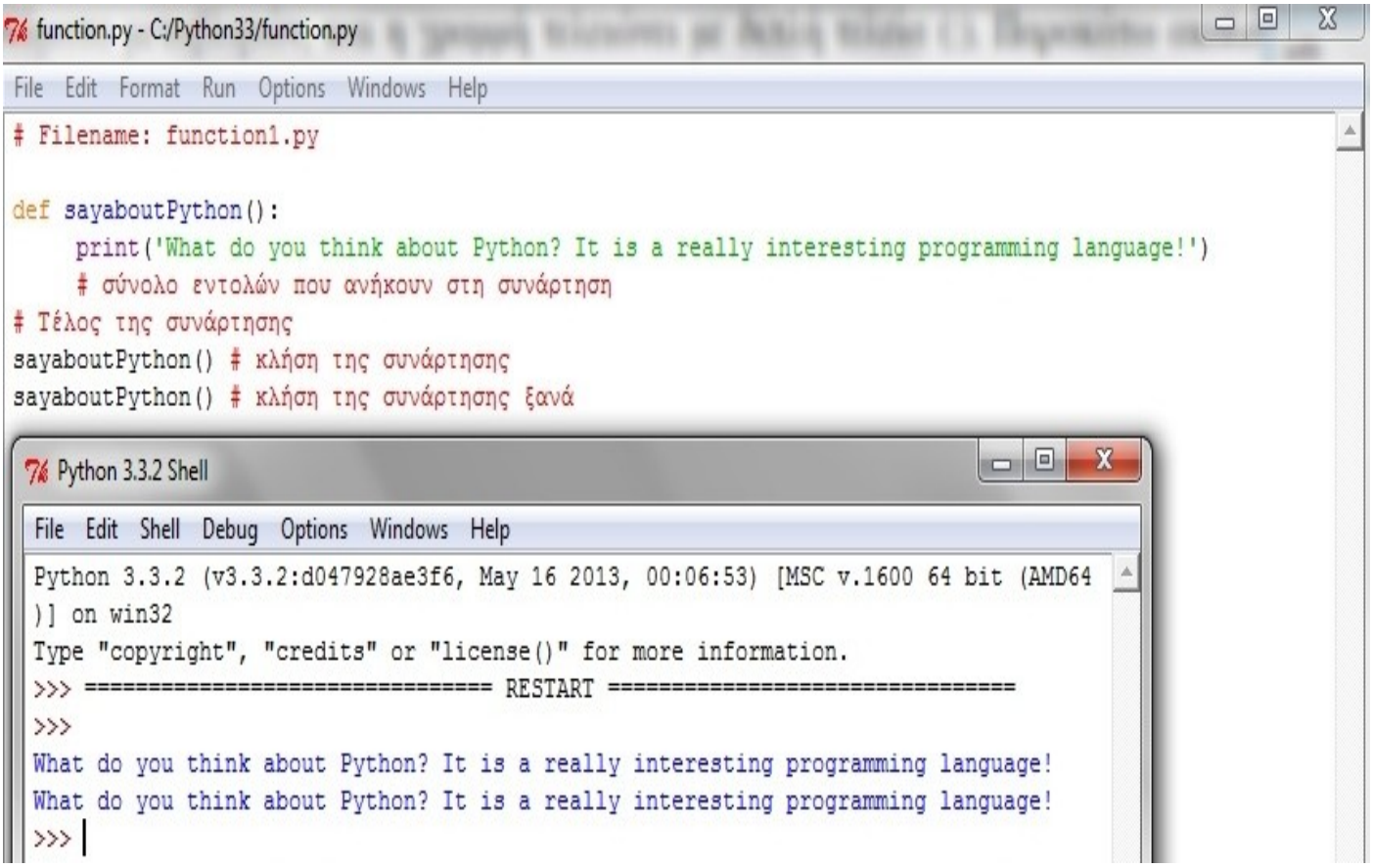


Στο παραπάνω παράδειγμα επειδή ένα αλφαριθμητικό είναι σταθερό, η συνάρτηση upper() δεν θα μπορούσε παρά να είναι μια αγνή συνάρτηση καθώς δεν γίνεται να μεταβληθεί ένα σταθερό αντικείμενο. Έτσι, αυτό που κάνει είναι να δημιουργεί ένα καινούργιο string που έχει όλα τα γράμματα κεφαλαία χωρίς να επηρεάζεται το a.

Αντιθέτως, πάνω σε ένα μεταβλητό αντικείμενο (mutable) , μια συνάρτηση μπορεί να είναι είτε αγνή, είτε συνάρτηση τροποποίησης. Επομένως, ενώ όπως περιμέναμε, η count() δεν επηρεάζει την λίστα b, η συνάρτηση reverse() επιδρά απευθείας πάνω σε αυτή και την αντιστρέφει.

```
b = ['a', 'b', 'c', 'd' ,]

print(b.count('a'))

print(b)

print(b.reserve())

print(b)
```

- Ακολουθεί ένα απλό παράδειγμα με ένα σύνολο εντολών που αποτελούν μέρος αυτής της συνάρτησης.



Πως δουλεύει το παραπάνω πρόγραμμα :

Ορίζουμε μια συνάρτηση με το όνομα sayaboutPython ακολουθώντας τη σύνταξη όπως αναλύσαμε παραπάνω. *Αυτή η συνάρτηση δεν έχει παραμέτρους γι' αυτό δε δηλώνονται καθόλου μεταβλητές ανάμεσα στις παρενθέσεις.*

Οι παράμετροι στις συναρτήσεις είναι απλά η είσοδος στη συνάρτηση ώστε να περνάμε διαφορετικές τιμές στη συνάρτηση και να παίρνουμε αντίστοιχα αποτελέσματα. **Σημειώστε ότι μπορούμε να καλούμε την ίδια συνάρτηση δύο φορές, δηλαδή δε χρειάζεται να γράφουμε τον ίδιο κώδικα δύο φορές.**

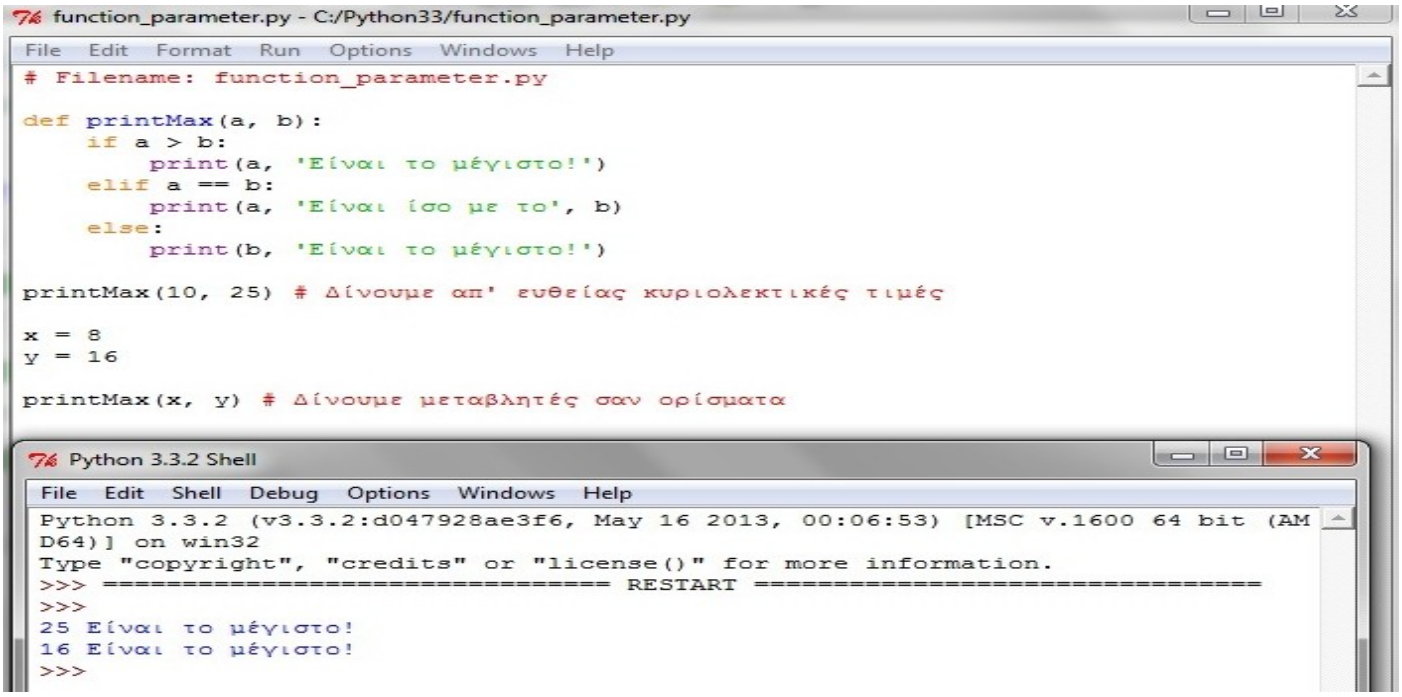
Παράμετροι συναρτήσεων

Μια συνάρτηση μπορεί να δεχθεί παραμέτρους, οι οποίες είναι τιμές που δίνετε στη συνάρτηση, έτσι ώστε αυτή να μπορεί να κάνει κάτι αξιοποιώντας αυτές τις τιμές. Αυτές οι παράμετροι μοιάζουν με τις μεταβλητές, διαφέροντας ως προς το ότι οι τιμές αυτών των μεταβλητών ορίζονται όταν καλούμε τη συνάρτηση και τους έχουν ήδη εκχωρηθεί τιμές όταν τρέχει η συνάρτηση.

Οι παράμετροι καθορίζονται μέσα στο ζευγάρι των παρενθέσεων στον ορισμό της συνάρτησης και διαχωρίζονται με κόμμα. Όταν καλούμε τη συνάρτηση δίνουμε και τις τιμές με τον ίδιο τρόπο.

Σημείωση για την ορολογία που χρησιμοποιείται: Οι ονομασίες που δίνετε στον ορισμό της συνάρτησης ονομάζονται **παράμετροι** ενώ οι τιμές που δίνετε όταν καλείτε τη συνάρτηση ονομάζονται **ορίσματα**.

Ακολουθεί ένα παράδειγμα :



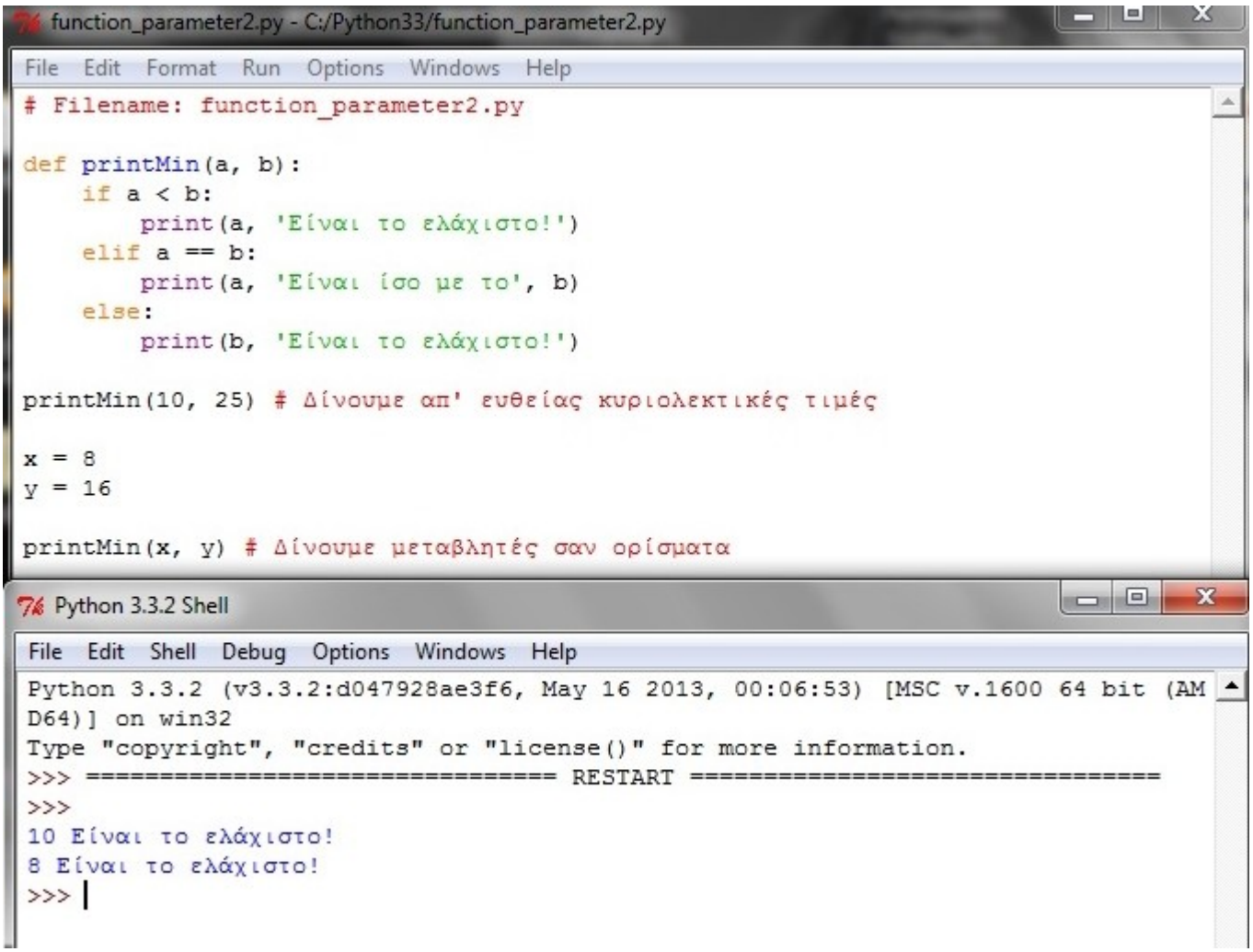
Πώς δουλεύει το παραπάνω πρόγραμμα :

Ορίζουμε μια συνάρτηση που ονομάζεται printMax με δυο παραμέτρους τις a και b. Βρίσκουμε το μεγαλύτερο νούμερο χρησιμοποιώντας μια απλή εντολή if..else και μετά τυπώνουμε το μεγαλύτερο νούμερο.

Στην πρώτη χρήση της printMax, απευθείας δίνουμε τους αριθμούς, δηλαδή τα ορίσματα. Στη δεύτερη χρήση της, καλούμε τη συνάρτηση χρησιμοποιώντας μεταβλητές. Η printMax(x, y) προκαλεί την τιμή του ορίσματος x να δοθεί στην παράμετρο a και την τιμή του ορίσματος y να δοθεί στην παράμετρο b. Η συνάρτηση printMax λειτουργεί με τον ίδιο τρόπο και στις δυο περιπτώσεις.

- Στη συνέχεια θα κατασκευάσουμε ένα πρόγραμμα και θα ορίσουμε μια συνάρτηση που ονομάζεται printMin με δύο παραμέτρους τις a και b. Βρίσκουμε το μικρότερο νούμερο χρησιμοποιώντας μια απλή if…else και μετά τυπώνουμε το μικρότερο νούμερο.

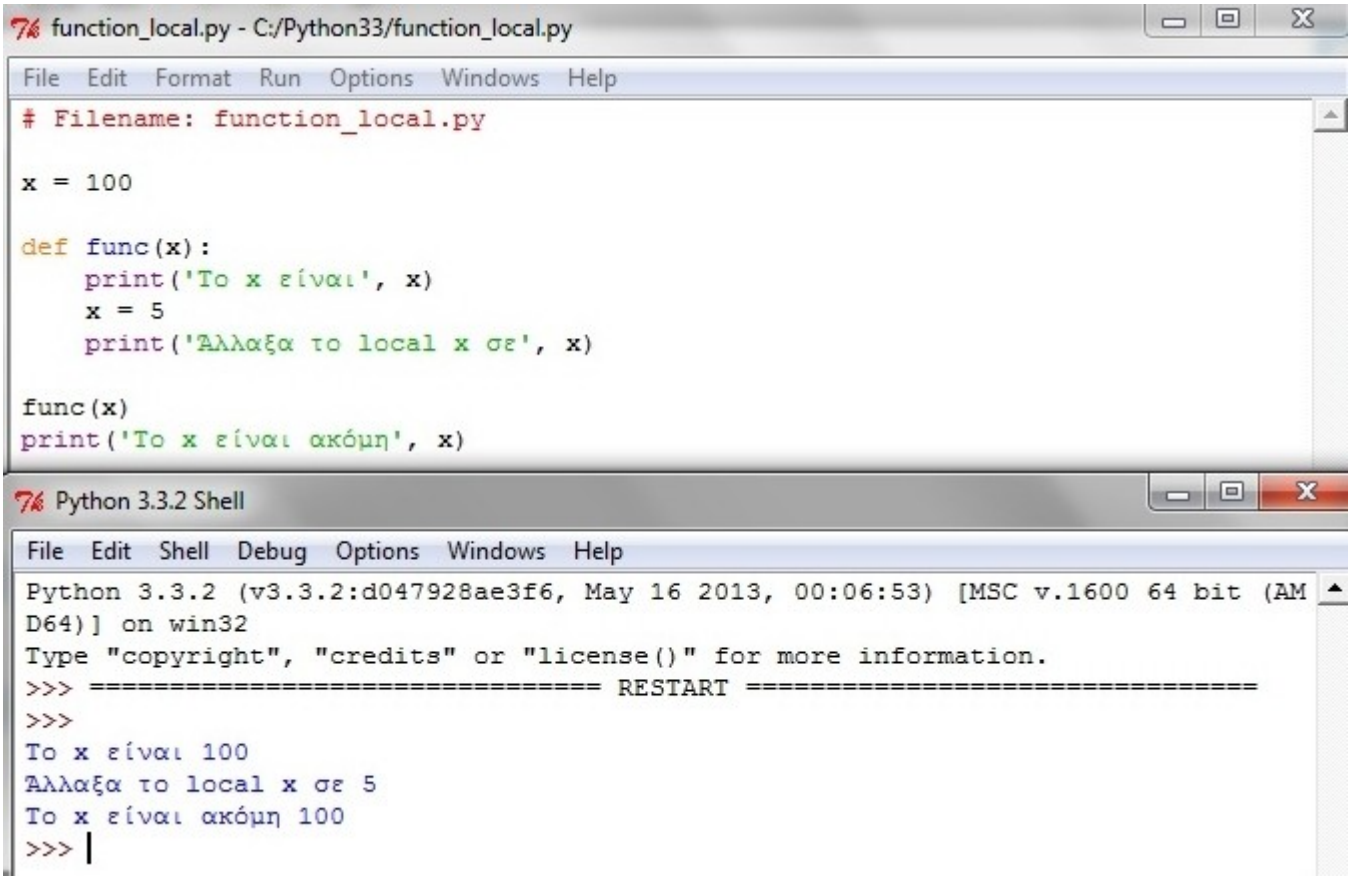
Στην πρώτη χρήση της printMax, απευθείας δίνουμε τους αριθμούς, δηλαδή τα ορίσματα. Στη δεύτερη χρήση της, καλούμε τη συνάρτηση χρησιμοποιώντας μεταβλητές. Η printMin (x, y) προκαλεί την τιμή του ορίσματος x να δοθεί στην παράμετρο a και την τιμή του ορίσματος y να δοθεί στην παράμετρο b. Η συνάρτηση printMin λειτουργεί με τον ίδιο τρόπο και στις δυο περιπτώσεις.



Τοπικές μεταβλητές (Local variables)

Όταν δηλώνετε μεταβλητές μέσα σε ένα ορισμό συνάρτησης, αυτές δεν έχουν καμία σχέση με άλλες μεταβλητές που έχουν την ίδια ονομασία και χρησιμοποιούνται έξω απο αυτή τη συνάρτηση, δηλαδή τα ονόματα των μεταβλητών χρησιμοποιούνται μόνο τοπικά στη συνάρτηση. Αυτό ονομάζεται ***εμβέλεια (scope) των μεταβλητών***. Όλες οι μεταβλητές έχουν την εμβέλεια του τμήματος όπου έχουν δηλωθεί, αρχίζοντας από το σημείο στο οποίο ορίζεται το όνομα.

Ακολουθεί παράδειγμα :



Πως δουλεύει το παραπάνω πρόγραμμα :

Στη συνάρτηση ,την πρώτη φορά που χρησιμοποιούμε την τιμή με το όνομα x, η Python χρησιμοποιεί την τιμή της παραμέτρου που έχει δηλωθεί στη συνάρτηση.

Κατόπιν εκχωρούμε την τιμή 5 στο x. ***Η ονομασία x είναι τοπική στη συνάρτησή μας***. Έτσι όταν αλλάζουμε την τιμή του x στη συνάρτηση, το x που ορίστηκε στο κύριο τμήμα δεν επηρεάζεται.

Στην τελευταία κλήση της συνάρτησης print, παρουσιάζουμε την τιμή x στο κύριο τμήμα και επιβεβαιώνουμε ότι δεν έχει επηρεαστεί.

Εντολή global

Εάν θέλετε να εκχωρήσετε μια τιμή σε ένα όνομα που ορίζεται στο κορυφαίο επίπεδο του προγράμματος (δηλαδή όχι μέσα σε κάποιου είδους εμβέλεια όπως σε συναρτήσεις ή κλάσεις), τότε πρέπει να πείτε στην Python ότι το όνομα δεν είναι τοπικό αλλά καθολικό (global). Αυτό γίνεται με τη χρήση της εντολής global.

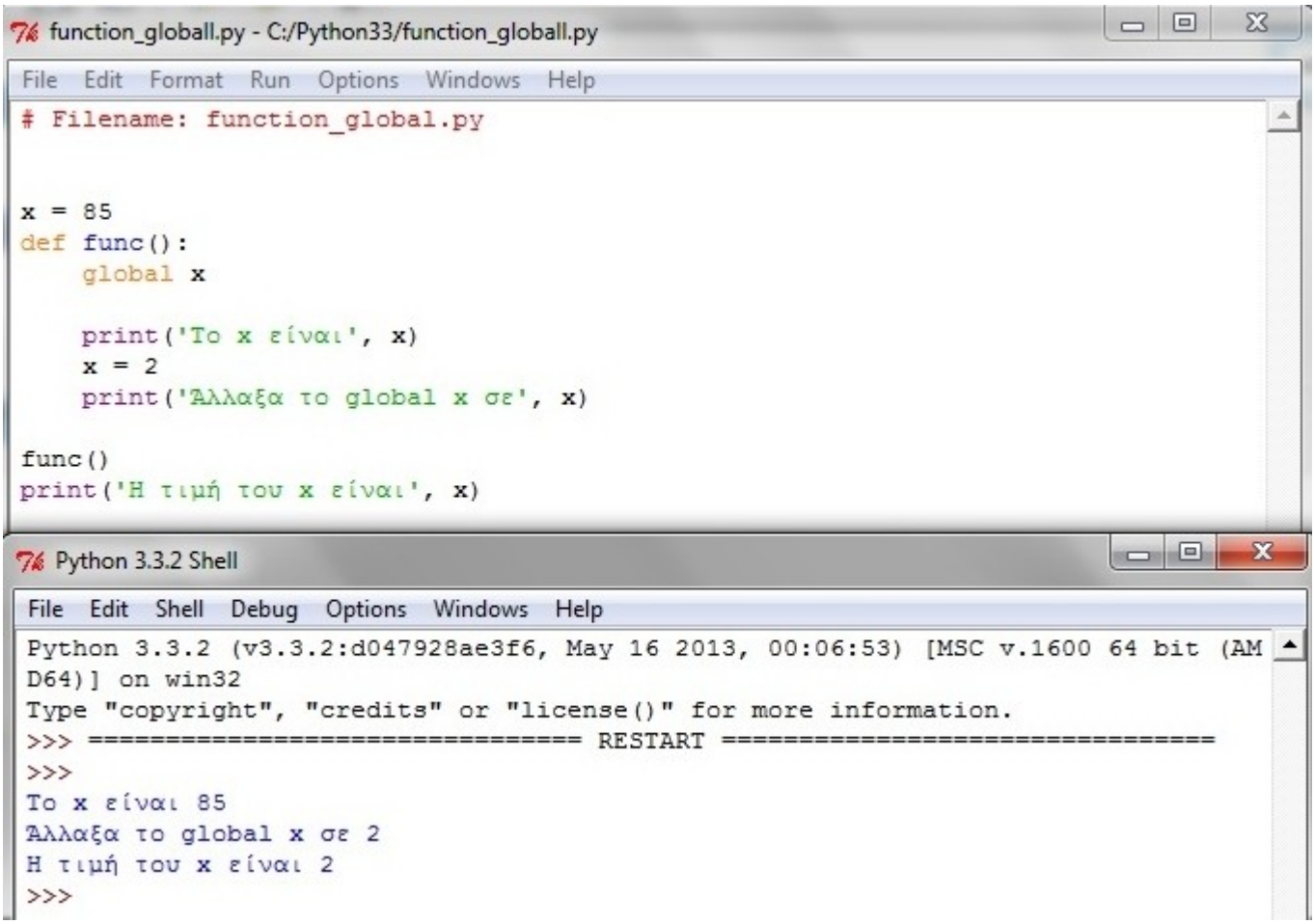
Είναι αδύνατον να εκχωρήσετε μια τιμή σε μια μεταβλητή που ορίζεται εκτός μιας συνάρτησης χωρίς την εντολή global.

Μπορείτε να χρησιμοποιήσετε τις τιμές τέτοιων μεταβλητών που ορίζονται έξω από τη συνάρτηση (υποθέτοντας ότι δεν υπάρχουν μεταβλητές με το ίδιο όνομα μέσα

στη συνάρτηση). Ωστόσο, κάτι τέτοιο δεν προτείνεται και πρέπει να αποφεύγεται μιας και δεν είναι ξεκάθαρο για τον αναγνώστη του προγράμματος για το πού βρίσκεται ο ορισμός της μεταβλητής.

Χρησιμοποιώντας την εντολή global γίνεται ξεκάθαρο ότι η μεταβλητή βρίσκεται σε ένα εξωτερικό τμήμα εντολών.

Ακολουθεί παράδειγμα χρήσης της μεταβλητής global :



Πως δουλεύει το παραπάνω πρόγραμμα :

Η εντολή global χρησιμοποιείται για να δηλώσει ότι το x είναι μια καθολική μεταβλητή, γι' αυτό το λόγο όταν εκχωρούμε μια τιμή για το x μέσα στη συνάρτηση, αυτή η αλλαγή απεικονίζεται όταν χρησιμοποιούμε την τιμή του x στο κυρίως τμήμα.

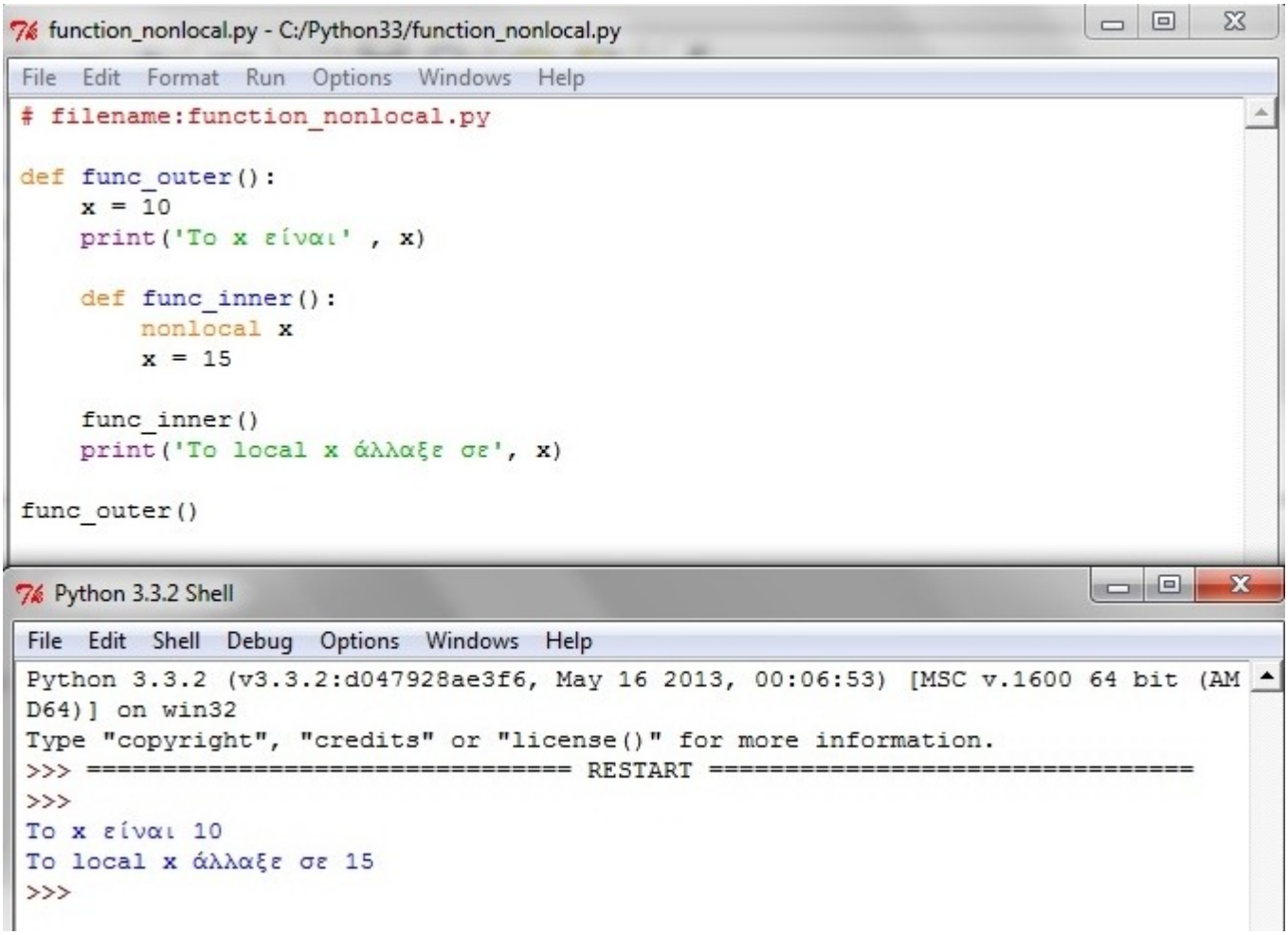
Μπορείτε να καθορίσετε περισσότερες από μία καθολικές μεταβλητές χρησιμοποιώντας την ίδια εντολή global. Για παράδειγμα global x, y, z.

Εντολή nonlocal

Έχουμε δει μέχρι τώρα πώς να έχουμε πρόσβαση σε μεταβλητές σε τοπική και καθολική εμβέλεια. Αλλά υπάρχει και ένα άλλο είδος εμβέλειας που ονομάζεται μη τοπική εμβέλεια (nonlocal scope) και είναι μεταξύ των δυο ανωτέρων τύπων. Οι μεταβλητές μη τοπικής εμβέλειας παρατηρούνται όταν ορίζετε συναρτήσεις μέσα στις συναρτήσεις.

Αφού όλα στη Python είναι εκτελέσιμος κώδικας, μπορείτε να ορίσετε συναρτήσεις οπουδήποτε.

Ακολουθεί παράδειγμα χρήσης της εντολής nonlocal :



Πώς δουλεύει το παραπάνω πρόγραμμα :

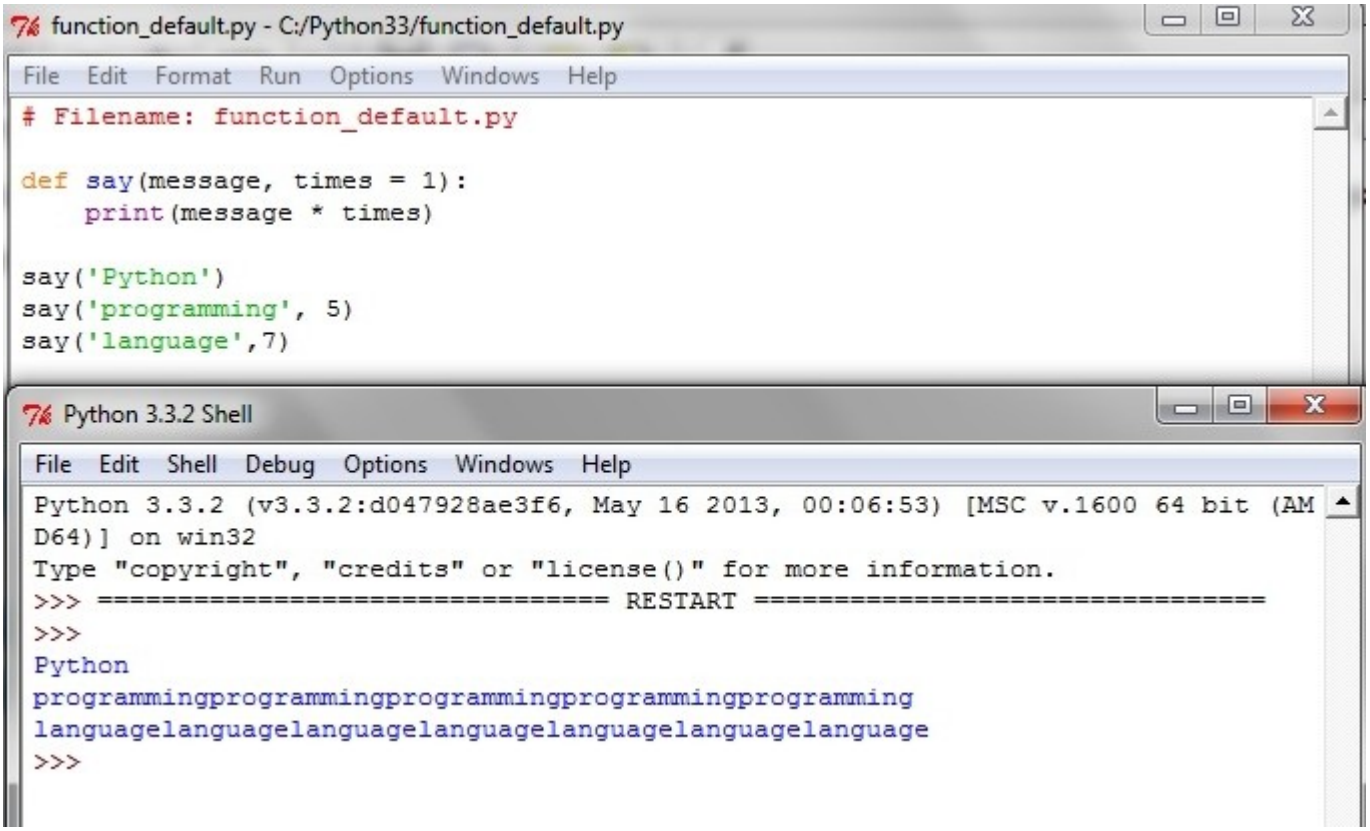
Όταν είμαστε μέσα στην func_inner, το 'x' που ορίζεται στην πρώτη γραμμή του func_outer, σχετικά δεν είναι ούτε σε τοπική ούτε σε καθολική εμβέλεια. Δηλώνουμε ότι χρησιμοποιούμε αυτό το x με το nonlocal x και έτσι αποκτούμε πρόσβαση σε αυτή τη μεταβλητή. Αλλάζοντας το nonlocal x σε global x, και επίσης μετακινώντας την ίδια τηνεντολή, παρατηρούμε τη διαφορά στην συμπεριφορά σε αυτές τις δύο περιπτώσεις.

Προεπιλεγμένες τιμές ορίσματος

Σε κάποιες συναρτήσεις ίσως να θέλουμε να κάνουμε μερικές παραμέτρους τους προαιρετικές και να χρησιμοποιήσουμε προεπιλεγμένες τιμές εάν ο χρήστης δε θέλει να δώσει τιμές σε τέτοιες παραμέτρους. Αυτό μπορεί να επιτευχθεί με τη βοήθεια των προεπιλεγμένων τιμών ορίσματος. Μπορείτε να καθορίσετε προεπιλεγμένες τιμές ορισμάτων για παραμέτρους, τοποθετώντας μετά το όνομα της παραμέτρου στον ορισμό της συνάρτησης τον τελεστή εκχώρησης (=) να ακολουθείται απο την προεπιλεγμένη τιμή.

Η προεπιλεγμένη τιμή ορίσματος πρέπει να είναι μια σταθερά. Για την ακρίβεια η προεπιλεγμένη τιμή ορίσματος πρέπει να είναι αμετάβλητη.

Στη συνέχεια ακολουθεί σχετικό παράδειγμα :



Πώς δουλεύει το παραπάνω πρόγραμμα:

Η συνάρτηση με το όνομα say χρησιμοποιείται για να τυπώσει μια συμβολοσειρά, τόσες φορές όσες έχει καθοριστεί. Εάν δεν έχει δοθεί τιμή ,τότε από προεπιλογή η συμβολοσειρά τυπώνεται μια φορά. Αυτό το πετυχαίνουμε καθορίζοντας μια προεπιλεγμένη τιμή ίση με 1 για τη παράμετρο times.

Στην πρώτη χρήση της συνάρτησης say παρέχουμε μόνο τη συμβολοσειρά και τυπώνει τη συμβολοσειρά μόνο μια φορά. Στη δεύτερη χρήση της say παρέχουμε και τη συμβολοσειρά και ένα όρισμα 5 δηλώνοντας έτσι ότι θέλουμε να πούμε (say) το μήνυμα της συμβολοσειράς 5 φορές.Στην τρίτη χρήση της say παρέχουμε τη συμβολοσειρά και ένα όρισμα 7 δηλώνοντας έτσι ότι θέλουμε να πούμε (say) το μήνυμα της συμβολοσειράς αυτής 7 φορές.

Μόνο σε εκείνες τις παραμέτρους, οι οποίες βρίσκονται στο τέλος της λίστας παραμέτρων, μπορούν να δοθούν προεπιλεγμένες τιμές ορίσματος, δηλαδή δε μπορούμε να έχουμε μια παράμετρο με προεπιλεγμένη τιμή ορίσματος πριν από μια παράμετρο χωρίς προεπιλεγμένη τιμή ορίσματος, στη σειρά των παραμέτρων που έχουν δηλωθεί στη λίστα παραμέτρων της συνάρτησης. Αυτό συμβαίνει διότι οι τιμές εκχωρούνται στις παραμέτρους με τη θέση τους. Για παράδειγμα η def func(a, b=5) ισχύει αλλά η def func(a=5, b) δεν ισχύει.

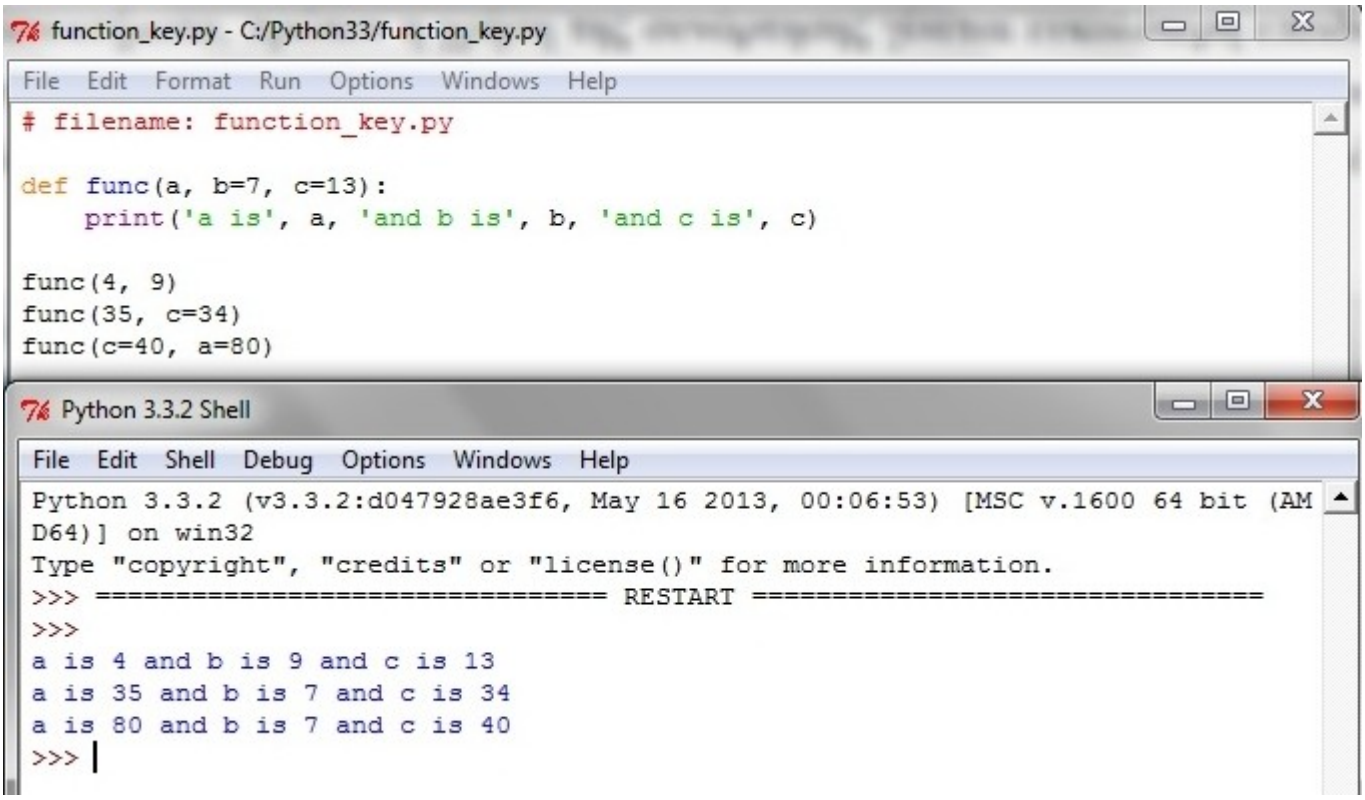
Ορίσματα με λέξεις-κλειδιά (Keyword Arguments)

Εάν έχετε κάποιες συναρτήσεις με πολλές παραμέτρους και θέλετε να καθορίσετε με ακρίβεια μόνο μερικές από αυτές ,τότε μπορείτε να δώσετε τιμές για τέτοιες παραμέτρους ονομάζοντας αυτές, δηλ. χρησιμοποιείτε την ονομασία (keyword) αντί της θέσης τους (που έχουμε χρησιμοποιήσει σε όλα τα άλλα) για να καθορίσουμε τα ορίσματα στη συνάρτηση.

Έτσι έχουμε ***δύο πλεονεκτήματα***:

1. Η χρήση της συνάρτησης γίνεται ευκολότερη επειδή δεν χρειάζεται να ανησυχούμε για τη διάταξη των ορισμάτων.
2. Μπορούμε να δώσουμε τιμές μόνο σε εκείνες τις παραμέτρους που θέλουμε, προνοώντας ότι οι άλλες παράμετροι έχουν τις προεπιλεγμένες τιμές ορίσματος.

Ακολουθεί παράδειγμα ορίσματος με λέξεις-κλειδιά :



Πώς δουλεύει το παραπάνω πρόγραμμα :

Η συνάρτηση με την ονομασία func έχει μια παράμετρο χωρίς προεπιλεγμένη τιμή ορίσματος και ακολουθείται από δύο παραμέτρους με προεπιλεγμένες τιμές ορίσματος. Στην πρώτη χρήση, func(4, 9) η παράμετρος a παίρνει την τιμή 4, η παράμετρος b παίρνει την τιμή 9 και η c 13.

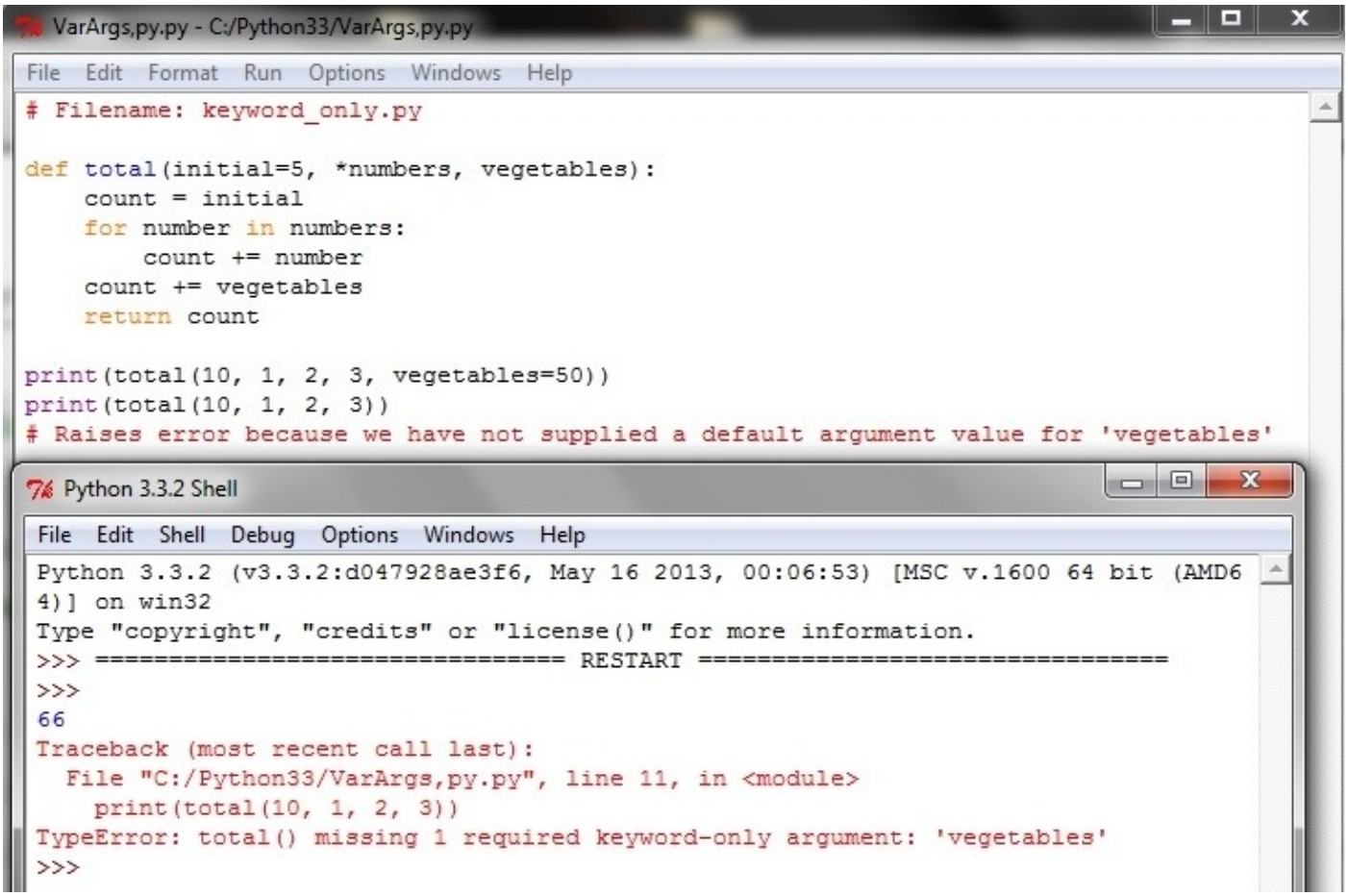
Στην δεύτερη χρήση func(35, c=34), η μεταβλητή a παίρνει την τιμή του 35 εξαιτίας της θέσης του ορίσματος. Κατόπιν η παράμετρος c παίρνει την τιμή του 34 εξαιτίας της ονομασίας δηλ. ορίσματα με λέξεις κλειδιά. Η μεταβλητή b παίρνει την προεπιλεγμένη τιμή του 7.

Στην τρίτη χρήση func(c=40, a=80) ,χρησιμοποιούμε μόνο ορίσματα με λέξεις κλειδιά για να καθορίσουμε τις τιμές. Σημειώστε ότι καθορίζουμε τιμή για την παράμετρο c πριν καθορίσουμε για την a ακόμα κι αν η a είναι ορισμένη πριν από τη c στον ορισμό της συνάρτησης.

Παράμετροι μόνο με λέξεις-κλειδιά

Αν θέλουμε να καθορίσουμε παραμέτρους με λέξεις-κλειδιά έτσι ώστε να είναι διαθέσιμες μόνον σαν λέξεις κλειδιά και όχι σαν ορίσματα σε συγκεκριμένες θέσεις, τότε αυτές πρέπει να δηλωθούν μετά από μια παράμετρο αστερίσκο.

Ακολουθεί σχετικό παράδειγμα :



Πώς δουλεύει υο παραπάνω πρόγραμμα :

Η δήλωση παραμέτρων μετά από μια παράμετρο με αστερίσκο, έχει ως αποτέλεσμα ορίσματα με λέξεις κλειδιά μόνο. Εάν σε αυτά τα ορίσματα δε δίνεται μια προεπιλεγμένη τιμή, τότε οι κλήσεις στη συνάρτηση θα δώσουν σφάλμα εάν το όρισμα με λέξη-κλειδι δεν δίνεται, όπως φαίνεται και παραπάνω.

Εάν θέλετε να έχετε ορίσματα με λέξεις-κλειδιά μόνον αλλά δεν χρειάζεστε μια παράμετρο με αστερίσκο, τότε χρησιμοποιείτε μόνο του τον αστερίσκο χωρις καμία ονομασία, όπως το def total(initial=5, *,vegetables).

Η εντολή return

Η εντολή return χρησιμοποιείται για να έχουμε επιστροφή από μια συνάρτηση, δηλαδή να βγούμε από τη συνάρτηση. Μπορούμε επίσης, προαιρετικά, να επιστρέψουμε μια τιμή από την συνάρτηση.

Ας δούμε δύο παραδείγματα :

1ο παράδειγμα :

function_return.py - C:/Python33/function_return.py

File Edit Format Run Options Windows Help

```
# Filename: function_return.py

def maximum(x, y):
    if x > y:
        return x
    elif x == y:
        return 'Οι αριθμοί είναι ίσοι'
    else:
        return y

print(maximum(8, 9))
```

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
9
>>>
```

Η συνάρτηση maximum επιστρέφει το μέγιστο (maximum) των παραμέτρων, και σε αυτή την περίπτωση των αριθμών που παρέχονται στη συνάρτηση. Χρησιμοποιεί μια απλή εντολή if..else για να βρει τη μεγαλύτερη τιμή και μετά επιστρέφει αυτή την τιμή.

2ο παράδειγμα :

triangle_return.py - C:/Python33/triangle_return.py

File Edit Format Run Options Windows Help

```
# Filename:triangle_return.py

def triangle_area_return(a, b):
    area = a * b / 2
    return area

def print_triangle_area_return(data):
    print("Το εμβαδό του τριγώνου είναι:", data)

myData = triangle_area_return(2, 120)
print_triangle_area_return(myData)
```

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Το εμβαδό του τριγώνου είναι: 120.0
>>>
```

Δημιουργούμε μία δεύτερη, πολύ απλή συνάρτηση η οποία λαμβάνει στην είσοδο ένα εμβαδό και τυπώνει ένα ειδικά διαμορφωμένο μήνυμα.Στη συνέχεια, στην τιμή myData ανατίθεται το αποτέλεσμα της συνάρτησης υπολογισμού του εμβαδού.Και τέλος, τέλος τροφοδοτούμε το περιεχόμενο του myData στην συνάρτηση εκτύπωσης.

Επιστροφή του αποτελέσματος αντί για εκτύπωση

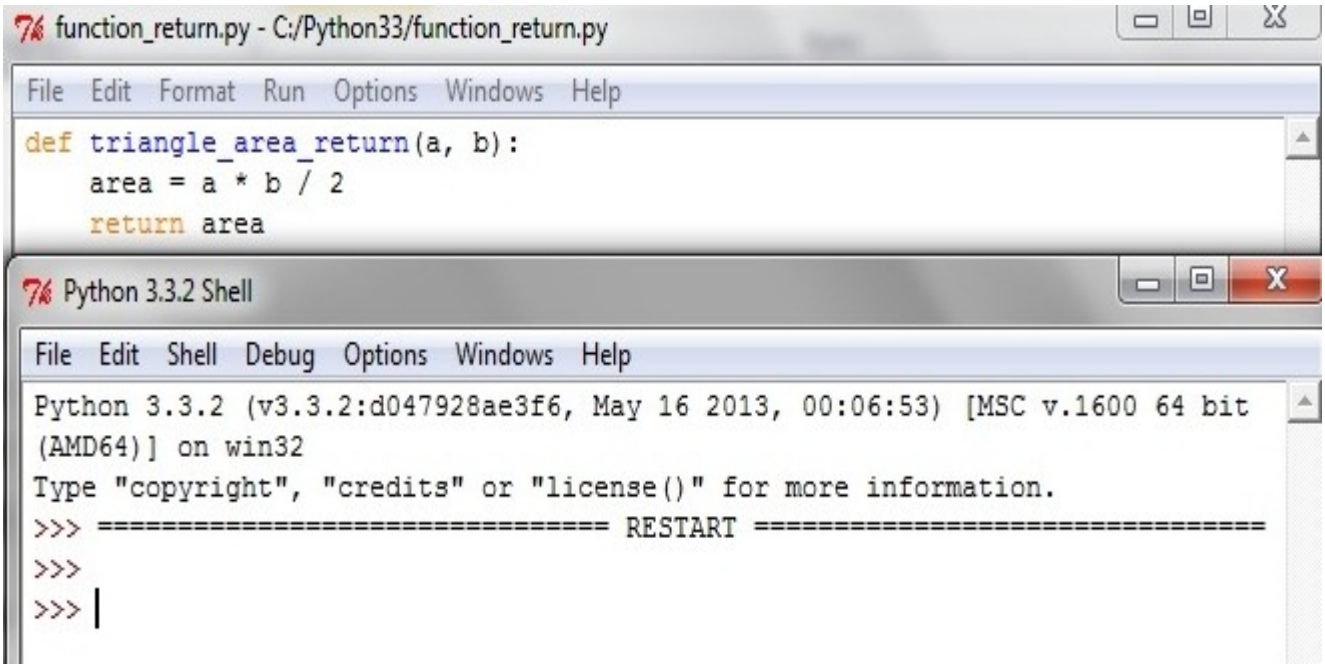
```
def triangle_area_return(a, b):

area = a * b / 2

return area
```

Σε αυτήν την εκδοχή της συνάρτησής μας ο προσεκτικός αναγνώστης παρατηρεί (εκτός από το διαφορετικό όνομα της συνάρτησης) μία ακόμη σημαντική διαφορά. Αντί για τη γραμμή εκτύπωσης του αποτελέσματος, χρησιμοποιούμε την δεσμευμένη λέξη return. Αυτή η αλλαγή έχει σαν αποτέλεσμα η τιμή του αποτελέσματος να επιστρέφεται αντί να εκτυπώνεται.

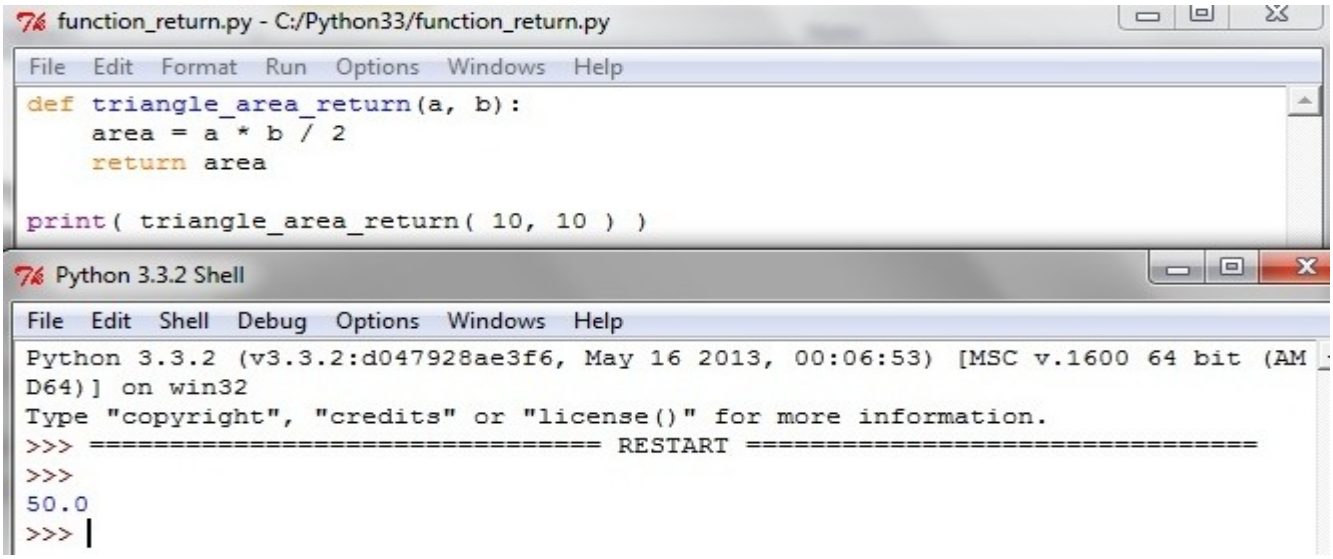
Έτσι, αν θέλουμε να εμφανίσουμε το αποτέλεσμα της συνάρτησης αυτής θα πρέπει να εκτελέσουμε την παρακάτω γραμμή κώδικα:

```
print( triangle_area_return( 10, 10 ) )
```

Στην πράξη, δοκιμάζοντας να εκτελέσουμε την ίδια γραμμή χωρίς να περικλείεται από την εντολή print( ) και θα διαπιστώσετε ότι τίποτε δε συμβαίνει.



ΚΑΙ



Σημειώνουμε ότι η εντολή return χωρίς μια τιμή είναι ισοδύναμη με την return None. Το None είναι ένας ειδικός τύπος στην Python που αντιπροσωπεύει τη μηδαμινότητα. Για παράδειγμα, χρησιμοποιείται για να δείξει ότι μια μεταβλητή δεν έχει καμία τιμή αν έχει την τιμή None.

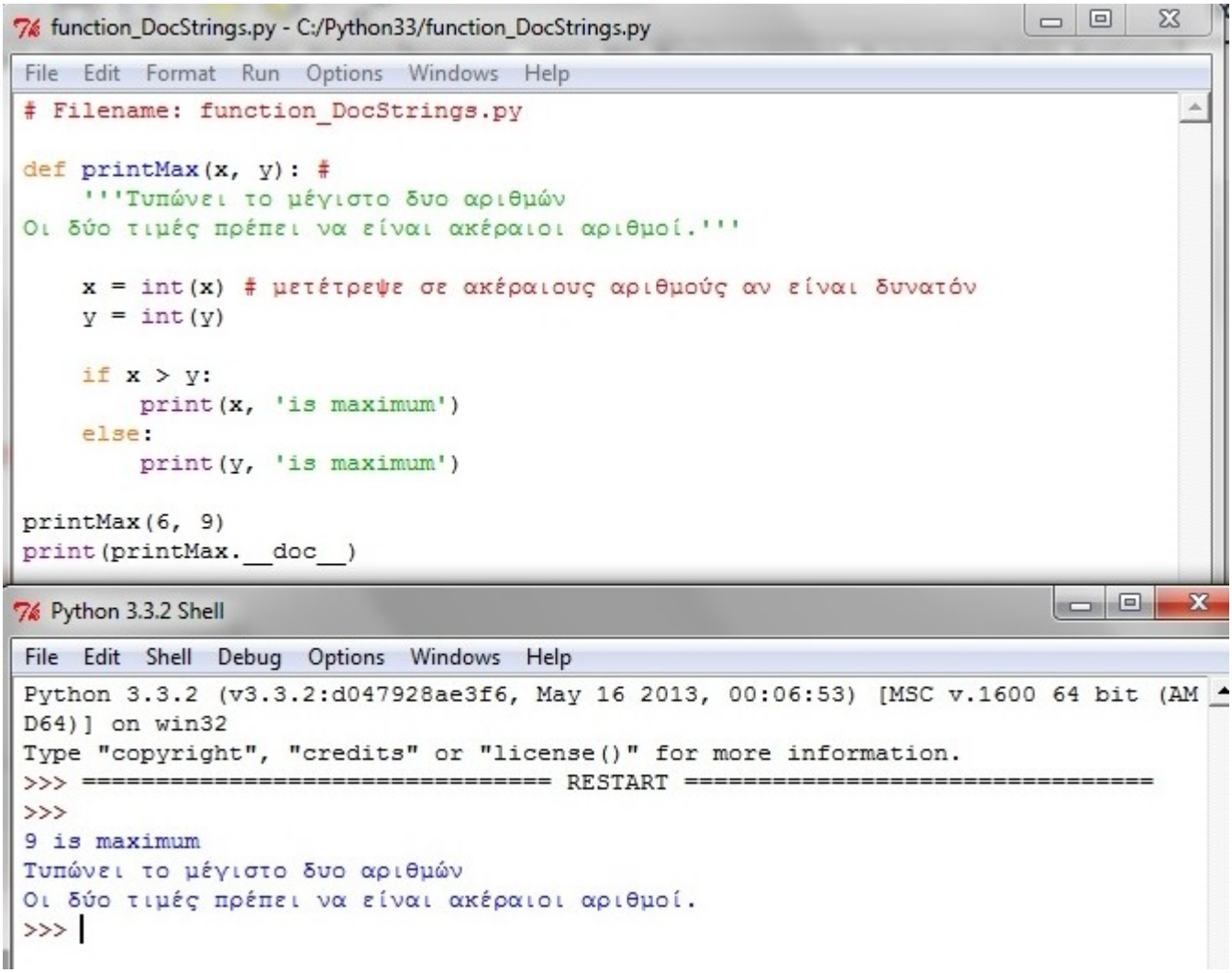
→ Κάθε συνάρτηση σιωπηρά περιέχει μια εντολή return None στο τέλος της εκτός κι αν έχουμε γράψει τη δική μας εντολή επιστροφής. Αυτό μπορείτε να το δείτε τρέχοντας την print(someFunction()) όπου η συνάρτηση someFunction δε χρησιμοποιεί την εντολή return:

```
def someFunction():

pass
```

✓ Η εντολή pass χρησιμοποιείται στην Python για να δείξει ένα άδειο τμήμα εντολών.

Συμβολοσειρές τεκμηρίωσης (DocStrings)

Η Python έχει ένα θαυμάσιο χαρακτηριστικό που ονομάζεται συμβολοσειρές τεκμηρίωσης (documentation strings) και συνήθως αναφέρεται με τη συντομογραφία **DocStrings**. Οι συμβολοσειρές τεκμηρίωσης είναι ένα σπουδαίο εργαλείο που πρέπει να χρησιμοποιείτε διότι βοηθάει να τεκμηριώσετε το πρόγραμμα καλύτερα και έτσι γίνεται πιο εύκολα κατανοητό. Το εντυπωσιακό είναι ότι ***μπορούμε να πάρουμε επιστροφή τη συμβολοσειρά τεκμηρίωσης από μια συνάρτηση για παράδειγμα, ενώ το πρόγραμμα πραγματικά τρέχει!***



Πώς δουλεύει το παραπάνω πρόγραμμα :

Μία συμβολοσειρά στην πρώτη λογική γραμμή της συνάρτησης είναι η συμβολοσειρά τεκμηρίωσης για αυτή τη συνάρτηση. Σημειώστε ότι οι συμβολοσειρές τεκμηρίωσης εφαρμόζονται και στα αρθρώματα (modules) και στις κλάσεις (classes), τα οποία θα μελετήσουμε στη συνέχεια.

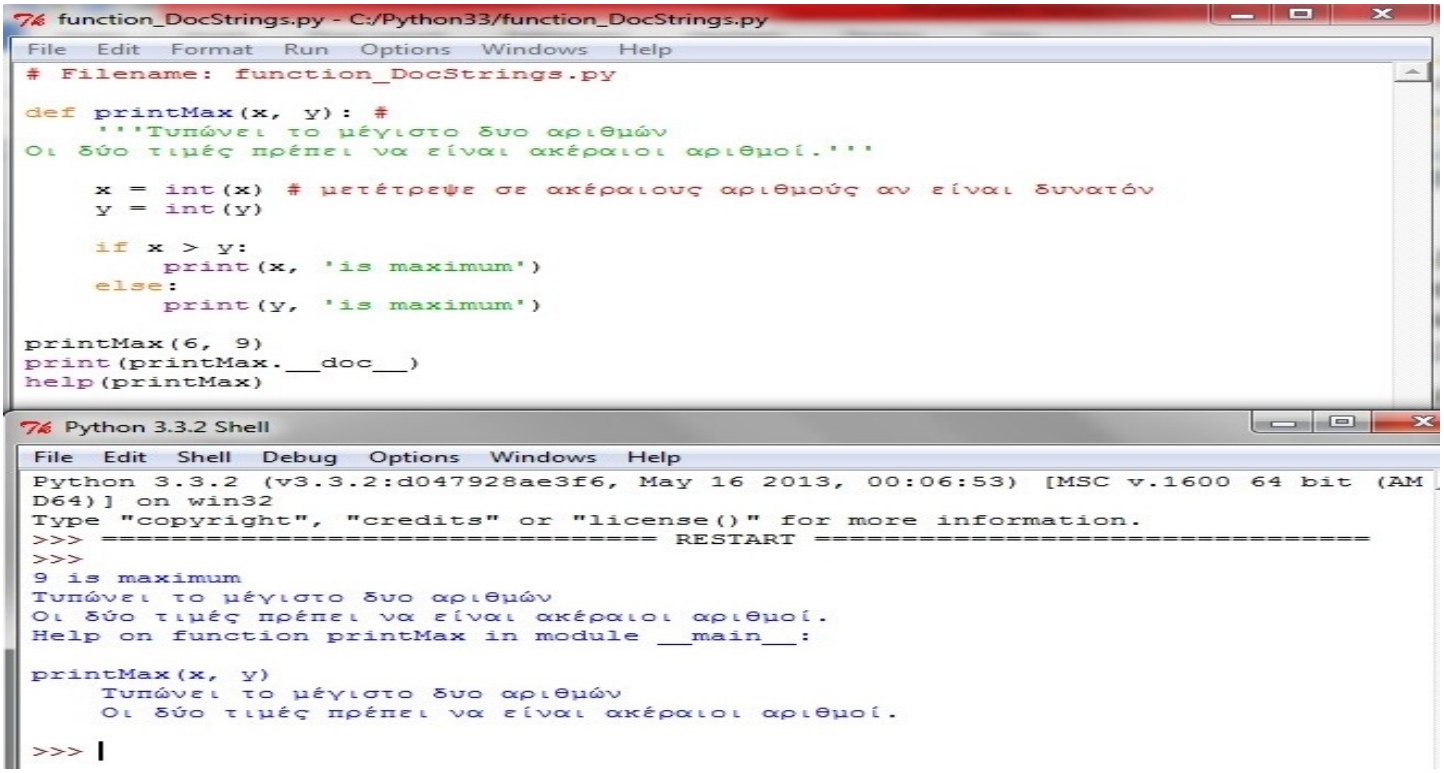
Η σύμβαση που ακολουθείται για μια συμβολοσειρά τεκμηρίωσης είναι μια συμβολοσειρά πολλών γραμμών, όπου η πρώτη γραμμή αρχίζει με ένα κεφαλαίο

γράμμα και τελειώνει με μια τελεία. Κατόπιν η δεύτερη γραμμή είναι κενή και ακολουθεί η τρίτη γραμμή που έχει κάθε λεπτομερή εξήγηση. Προτείνεται να ακολουθείτε ακριβώς αυτή τη σύμβαση για όλες τις στοιχειοσειρές τεκμηρίωσης που αφορούν όλες τις μη τετριμμένες συναρτήσεις (non-trivial functions).

Μπορούμε να έχουμε πρόσβαση στη συμβολοσειρά τεκμηρίωσης της συνάρτησης printMax χρησιμοποιώντας το ιδιοχαρακτηριστικό (ονομασία που ανήκει σε) __doc__ (σημειώστε τις διπλές κάτω παύλες) της συνάρτησης. Θυμηθείτε όμως ότι η Python αντιμετωπίζει τα πάντα σαν αντικείμενα, και αυτό συμπεριλαμβάνει και τις συναρτήσεις.

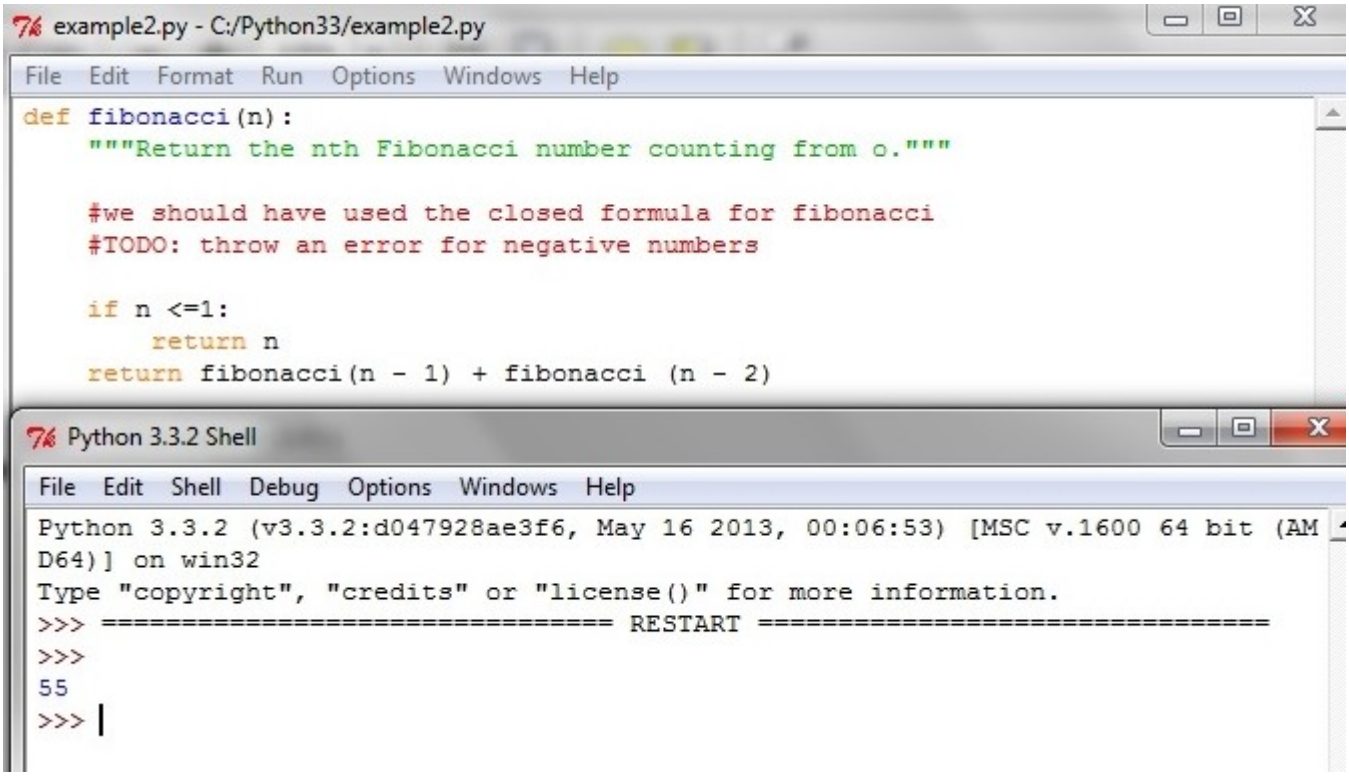
➔ help() στην Python→χρήση των συμβολοσειρών τεκμηρίωσης.
Αυτό που κάνει η help() είναι να φέρνει το ιδιοχαρακτηριστικό __doc__ της συνάρτησης και να το εμφανίζει με ένα καλοφτιαγμένο τρόπο.

Μπορούμε να το δοκιμάσουμε στην παραπάνω συνάρτηση αρκεί να συμπεριλάβουμε την help(printMax) στο πρόγραμμά μας.



Μερικά αυτοματοποιημένα εργαλεία μπορούν να ανακτήσουν την τεκμηρίωση από το πρόγραμμά σας με αυτόν τον τρόπο. Συνεπώς συνιστάται να χρησιμοποιείτε συμβολοσειρές τεκμηρίωσης σε κάθε μη τετριμμενη συνάρτηση που γράφετε. Η **εντολή pydoc** που συνοδεύει την Python λειτουργεί **παρόμοια με τη help()** χρησιμοποιώντας τις docstrings.

Επιπλέον παράδειγμα



Στην Python, υπάρχει η σύμβαση αφού ορίσουμε μια συνάρτηση να περιγράφουμε χρησιμοποιώντας ένα αλφαριθμητικό το οποίο μπορεί να συνεχιστεί σε παραπάνω από μια γραμμές , το σκοπό της συνάρτησης.

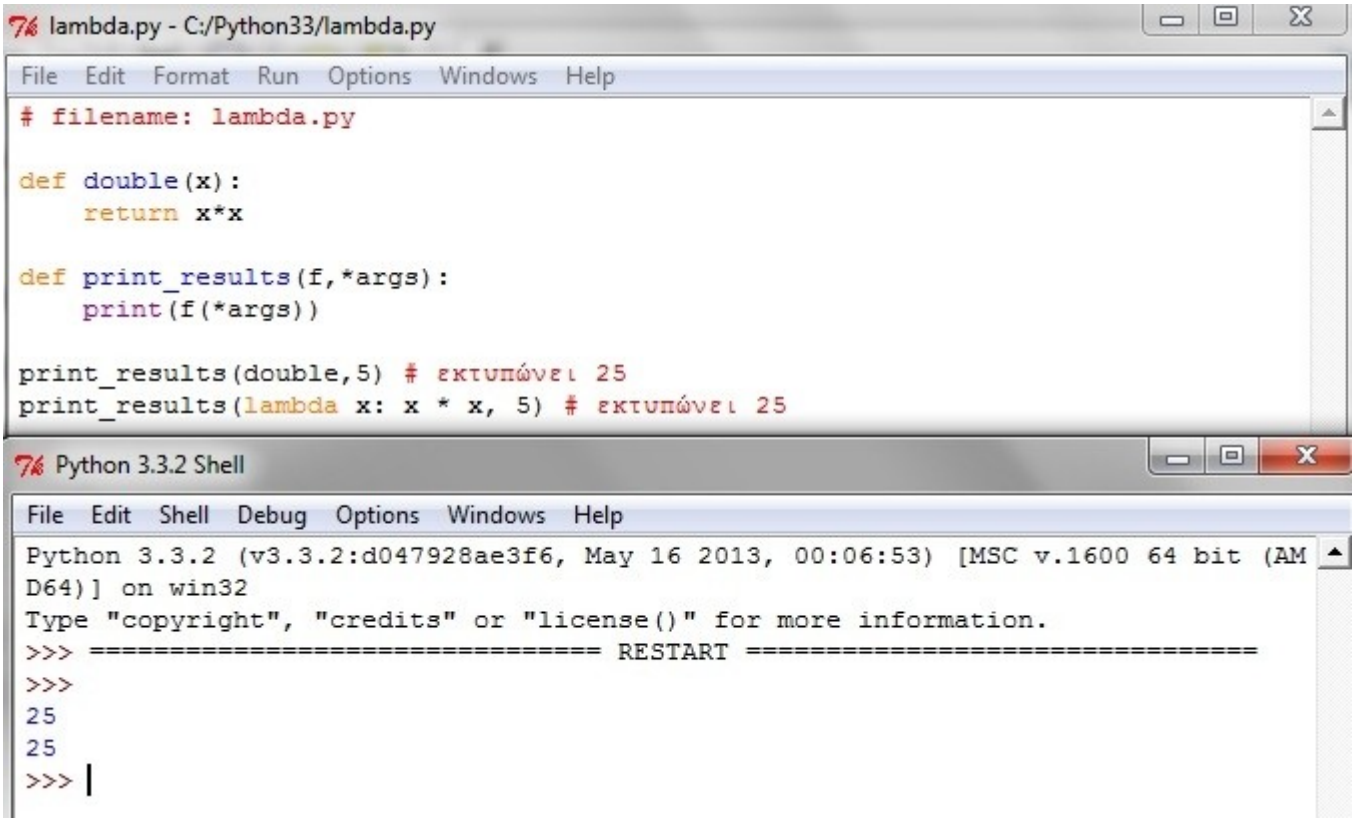
Συνήθως, και ακολουθώντας το PEP 257, η πρώτη γραμμή είναι σε προστακτική και αναφέρεται στο τι θέλουμε να κάνει η συνάρτηση ενώ στη συνέχεια και αφήνοντας μια γραμμή κενή, μπορούμε να εξηγήσουμε (προαιρετικά) με περισσότερα λόγια την συνάρτησή μας.

Αν ακολουθούμε αυτό τον τρόπο περιγραφής των συναρτήσεών μας, τότε υπάρχουν αρκετά αυτόματα εργαλεία που θα παράγουν αυτόματα την τεκμηρίωση του κώδικά μας σε μια φιλική μορφή με βάση τις συμβολοσειρές τεκμηρίωσης. Αντίθετα με αυτές, τα απλά σχόλια αγνοούνται.

Ανώνυμες συναρτήσεις

Υπάρχουν φορές που θέλουμε να ορίσουμε μια συνάρτηση ώστε να την περάσουμε ως όρισμα κάπου αλλού. Παράδειγμα αποτελούν οι αλγόριθμοι ταξινόμησης στους οποίους μπορούμε να περνάμε δικές μας συναρτήσεις σύγκρισης στοιχείων. Για αυτό τον λόγο, υπάρχουν οι ανώνυμες συναρτήσεις.

Στη συνέχεια, παραθέτω ένα παράδειγμα για το πώς μετατρέπουμε μια συνάρτηση σε ανώνυμη.



Εξήγηση → Θα χρησιμοποιήσουμε τη λέξη κλειδί lambda από το ελληνικό γράμμα λ. Αυτός ο όρος χρησιμοποιείται ως αναφορά σε κλάδο των μαθηματικών που ασχολείται με συναρτήσεις. Πρακτικά, η λέξη lambda αντικαθιστλα τη λέξη κλειδί def. Στη συνέχεια παραλείπουμε το όνομα της συνάρτησης καθώς και τις παρενθέσεις γύρω από τα ορίσματα. Τέλος, γράφουμε τι επιστρέφει η συνάρτηση, χωρίς όμως να χρησιμοποιούμε τη λέξη κλειδί return.

Διακοσμητές (Decorators)

Οι διακοσμητές είναι ειδικές συναρτήσεις στις οποίες μπορούμε να χρησιμοποιήσουμε αν θέλουμε να κάνουμε κάτι επιπρόσθετο πριν ή μετά την κλήση μιας συνάρτησης . Για παράδειγμα αν θέλουμε το αποτέλεσμα μια συνάρτησης να το προσαρμόσουμε σε μια συγκεκριμένη διαμόρφωση, ή αν θέλουμε να αλλάξουμε την σειρά των περιεχομένων σε μια λίστα ώστε να δημιουρήσουμε έναν τυχαιοποιημένο αλγόριθμο (randomized algorithm).

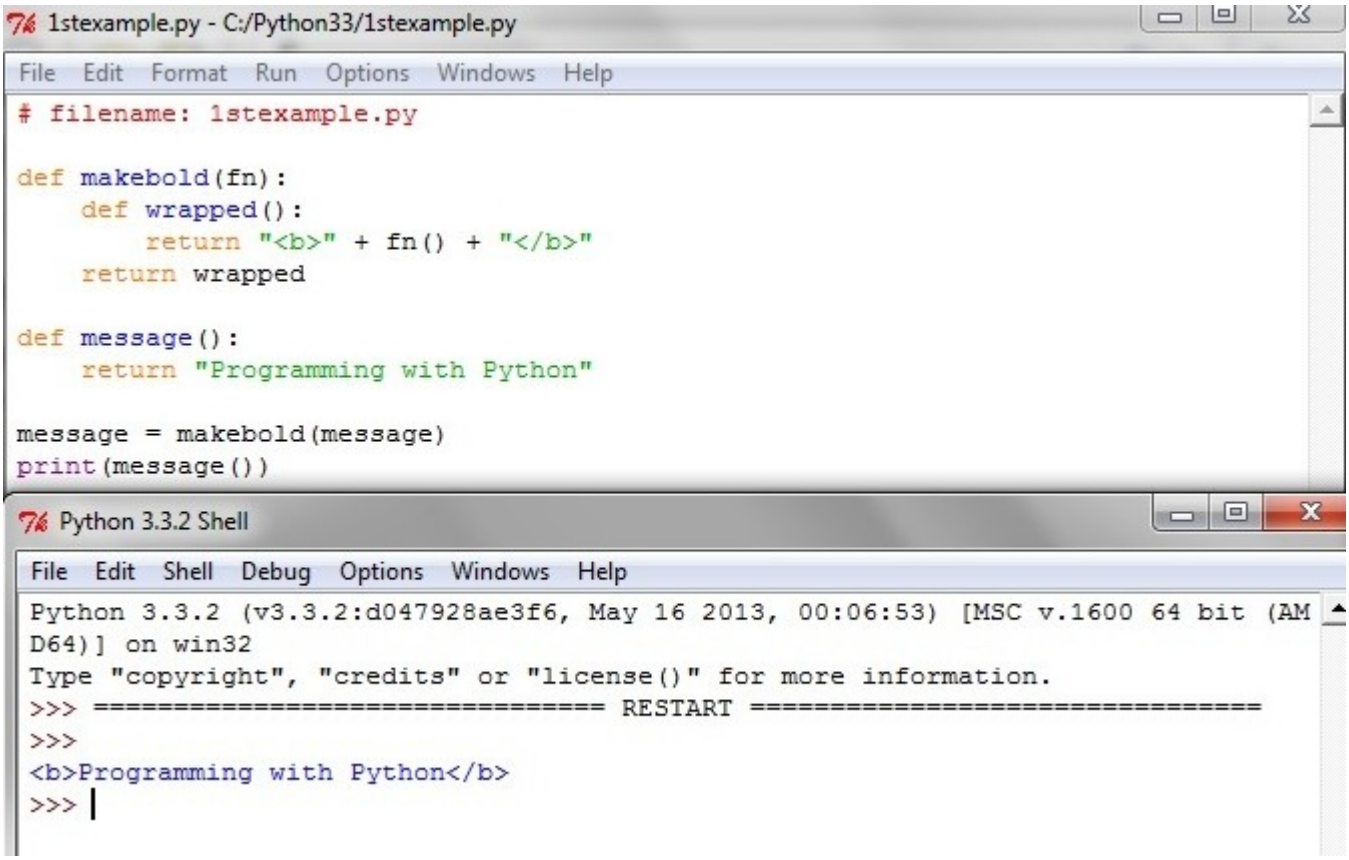
Ο λόγος που αυτό είναι εφικτό είναι επειδή στην Python τα πάντα είναι αντικείμενα. Επομένως και τις συναρτήσεις μπορούμε να τις χρησιμοποιήσουμε ως αντικείμενα. Μέχρι τώρα είχαμε συνηθίσει να χρησιμοποιούμε πάντα μια συνάρτηση με τις παρενθέσεις δίπλα της. Έτσι, όταν καλέσουμε την συνάρτηση, εκτελείται το σώμα της . Αν δεν χρησιμοποιήσουμε αυτές τις παρενθέσεις, τότε αναφερόμαστε απευθείας στο αντικείμενο που προσδιορίζει αυτή. Έτσι λοιπόν, παίρνοντας αυτό το όρισμα ως αντικείμενο σε μια άλλη συνάρτηση είναι δυνατό τελικά να

τροποποιήσουμε τα αποτελέσματα που εξάγονται ή τα ορίσματα που περνιούνται σε αυτή τη συνάρτηση.

Οι διακοσμητές είναι ένα ιδίωμα της Python που επιτρέπει την κλήση κάποιας συνάρτησης η οποία τροποποιεί ελαφρώς την συμπεριφορά της συνάρτησης που καλούμε.

Για να γίνει αυτό πιο ξεκάθαρο , παραθέτω ένα παράδειγμα που έχει την ίδια συμπεριφορά. Στο 1ο παράδειγμα,καλούμε απλά άλλη μια συνάρτηση ώστε να τροποποιηθεί η έξοδος, ενώ στο 2ο παράδειγμα κάνουμε ακριβώς το ίδιο πράγμα με την χρήση διακοσμητών (decorators).

1ο παράδειγμα :



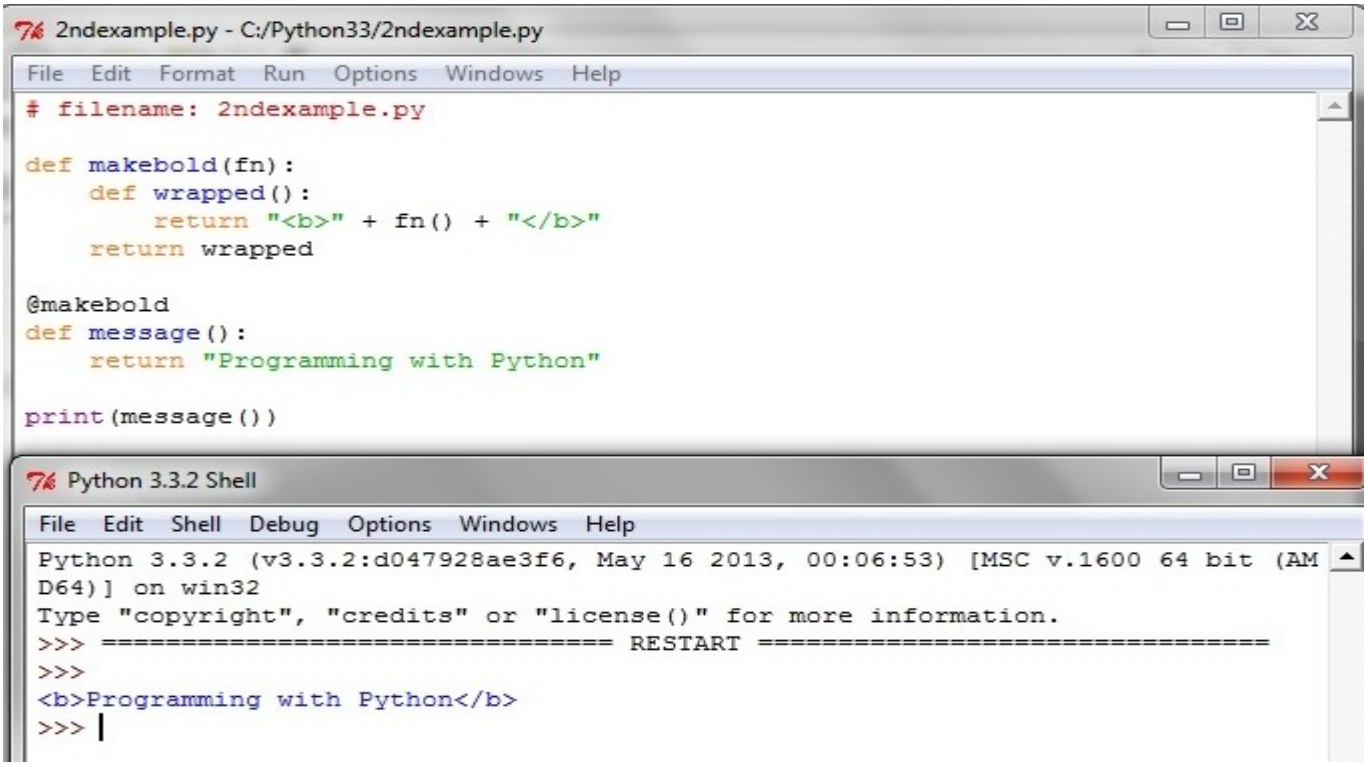
Μπορούμε να ορίσουμε μια καινούρια συνάρτηση g μέσα σε μια προϋπάρχουσα συνάρτηση f. Έτσι η g φαίνεται μόνο μέσα στην συνάρτηση f. Στο παραπάνω παράδειγμα , βλέπουμε ότι μέσα στην συνάρτηση makebold() ορίζεται η συνάρτηση wrapped() . Προσοχή πρέπει να δοθεί στο ότι δε καλείται αυτόματα η συνάρτηση wrapped() με τον ορισμό της. Πρέπει να την καλέσουμε εμείς ρητά στο σημείο που την χρειαζόμαστε. Επίσης, δεν χρειάζεται να περάσουμε ως όρισμα το αντικείμενο fn στην wrapped() ώστε να μπορέσουμε να κάνουμε πάξεις με αυτό. Η wrapped() το βλέπει ήδη καθώς εμπεριέχεται στο σώμα της makebold οπότε στο περιβάλλον της είναι ότιδήποτε έχει το περιβάλλον της makebold().

Ίσως φανεί παράξενο ότι στο message εκχωρούμε την makebold(message). Όπως είπαμε και πριν, στην Python είναι όλα αντικείμενα. Συνεπώς, με την makebold(message) έχουμε δημιουργήσει μια καινούργια συνάρτηση την οποία εκχωρούμε στην message. Έτσι πλέον η message δείχνει σε καινούργια περιεχόμενα πια και έχει τροποποιημένη συμπεριφορά, την οποία και βλέπουμε όταν εκτυπώνουμε ό,τι επιστρέφει.

Αυτό που κάνει η makebold() , είναι πως μέσα της δημιουργείται μια συνάρτηση wrapped(). Η wrapped() χρησιμοποιεί το όρισμα της makebold() και συνενώνει το string που επιστρέφει η fn με κάποια άλλα. Το τελικό αντικείμενο που δημιουργείται επιστρέφεται. Επειδή αυτό το αντικείμενο είναι συνάρτηση, εμείς μπορούμε να το αναθέσουμε σε μια μεταβλητή και στην συνέχεια να το χρησιμοποιήσουμε κανονικά σαν συνάρτηση.

Η ίδια λειτουργικότητα, με πιο «κομψό» κώδικα επιτυγχάνεται όπως φαίνεται στο παράδειγμα που ακολουθεί με την χρήση διακοσμητών.

2ο παράδειγμα :



Το αποτέλεσμα είναι ΑΚΡΙΒΩΣ ΤΟ ΙΔΙΟ, μόνο που σε αυτή την περίπτωση πτιν ορίσουμε την συνάρτηση message() χρησιμοποιούμε τον διακοσμητή ο οποίος προσδιορίζει την συμπεριφορά που αυτός περιγράφει και θα περιλαμβάνει η συνάρτησή μας , και στην συνέχεια υλοποιούμε κανονικά την συνάρτηση. Από εκεί και πέρα, όπουε καλούμε την συνάρτηση message(), αυτή θα περιλαμβάνει και τη συμπεριφορά που ορίζεται από την makebold.

Αρθρώματα

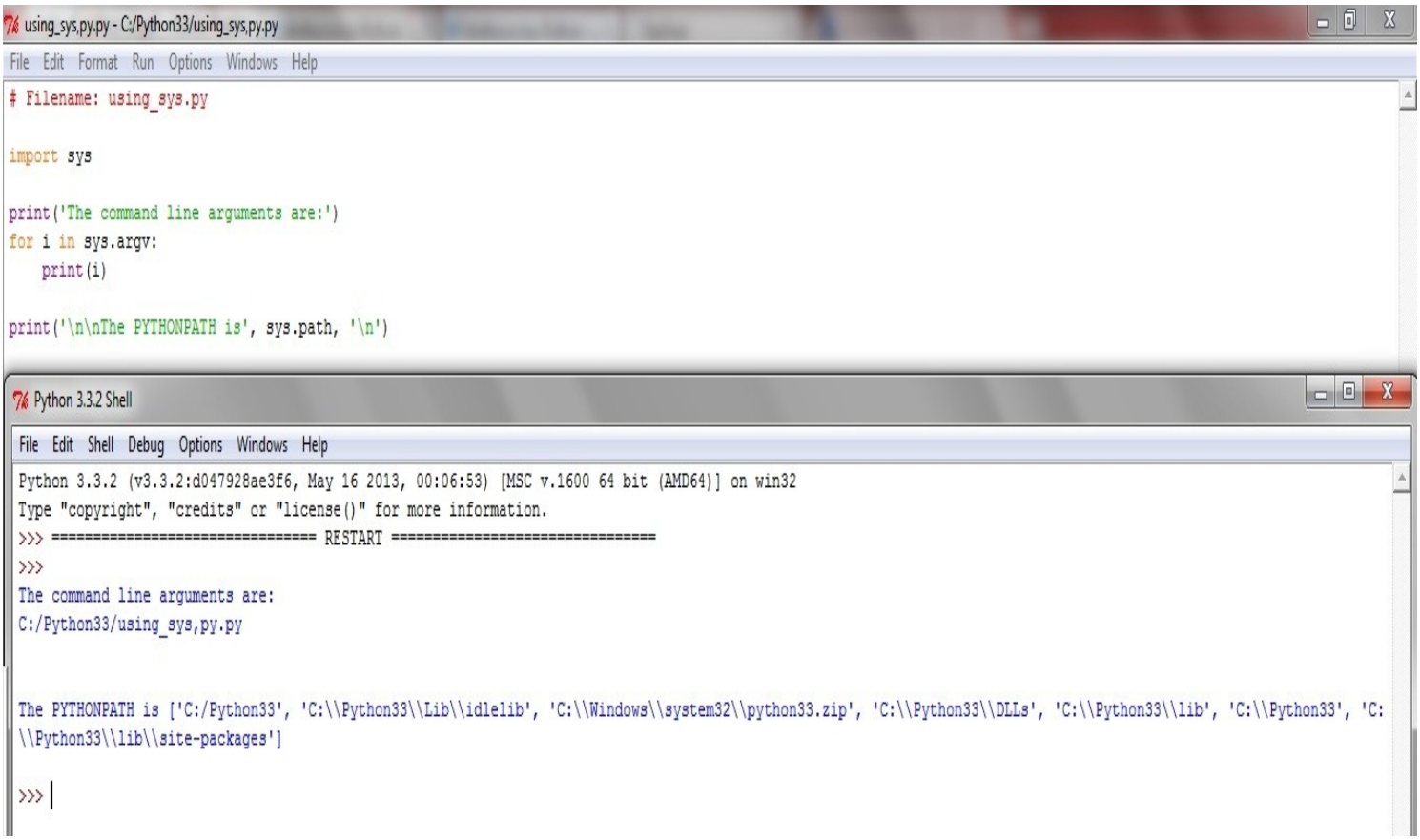
Έχετε δει πώς μπορείτε να επαναχρησιμοποιήσετε κώδικα στο πρόγραμμά σας ορίζοντας μια φορά συναρτήσεις. Τι κάνετε όμως αν θέλετε να επαναχρησιμοποιήσετε έναν αριθμό συναρτήσεων σε άλλα προγράμματα που γράφετε; Όπως ίσως έχετε μαντέψει η απάντηση είναι τα ***αρθρώματα***.

Υπάρχουν διάφορες μεθόδοι γαι να γράφετε αρθρώματα, αλλά ο απλούστερος τρόπος είναι *δημιουργώντας ένα αρχείο με επέκταση .py το οποίο θα περιέχει συναρτήσεις και μεταβλητές.*

Ένας άλλος τρόπος είναι να γράφετε τα αρθρώματα στην αρχική γλώσσα στην οποία γράφτηκε ο διερμηνευτής της Python. Για παράδειγμα μπορείτε να γράψετε αρθρώματα στη γλώσσα προγραμματισμού C και μόλις μεταγλωτιστούν, να χρησιμοποιηθούν από τον κώδικα που γράφετε σε Python όταν χρησιμοποιείτε τον πρότυπο διερμηνευτή της Python.

- ✓ Ένα άρθρωμα μπορεί να εισαχθεί από ένα άλλο πρόγραμμα για να κάνουμε χρήση της λειτουργικότητάς του. Έτσι μπορούμε να χρησιμοποιήσουμε επίσης την πρότυπη βιβλιοθήκη της Python. Αρχικά θα δούμε πώς να χρησιμοποιούμε τα αρθρώματα της πρότυπης βιβλιοθήκης.

Ακολουθεί παράδειγμα :



Πώς λειτουργεί το παραπάνω πρόγραμμα :

➔ Αρχικά εισάγετε το άρθρωμα sys χρησιμοποιώντας την εντολή import. Κατά κύριο λόγο αυτό για την Python σημαίνει ότι θέλουμε να χρησιμοποιήσουμε αυτό το άρθρωμα. Το άρθρωμα sys περιέχει λειτουργικότητα που έχει σχέση με τον διερμηνευτή της Python και το περιβάλλον του δηλ. το system.

Όταν η Python εκτελεί την εντολή import sys, ψάχνει για το άρθρωμα sys. Σε αυτή την περίπτωση, είναι ένα από τα ενσωματωμένα αρθρώματα, και γι αυτό η Python ξέρει που θα το βρει.

Εάν δεν ήταν ένα ενσωματωμένο άρθρωμα δηλ. ένα άρθρωμα γραμμένο σε Python, τότε ο διερμηνευτής της Python θα το έψαχνε στους καταλόγους που βρίσκονται στη μεταβλητή sys.path. Εάν το άρθρωμα βρεθεί τότε οι εντολές στο κυρίως τμήμα του αρθρώματος τρέχουν και τότε το άρθρωμα σας γίνεται διαθέσιμο για να το χρησιμοποιήσετε. Σημειώστε ότι η αρχικοποίηση γίνεται μόνο την πρώτη φορά που εισάγουμε ένα άρθρωμα.

- ✓ Η μεταβλητή argv στο άρθρωμα sys εισάγεται χρησιμοποιώντας συμβολισμό με τελείες δηλ. sys.argv. Αυτό δείχνει ξεκάθαρα ότι αυτή η ονομασία είναι μέρος του αρθρώματος sys. Ένα ακόμα πλεονέκτημα αυτής της προσέγγισης είναι ότι αυτή η ονομασία δεν συγκρούεται με καμμία άλλη μεταβλητή argv που χρησιμοποιείται στο πρόγραμμά σας.
- ✓ Η μεταβλητή sys.argv είναι μια λίστα συμβολοσειρών. Ειδικά η sys.argv περιέχει τον κατάλογο των ορισμάτων της γραμμής εντολών (command line arguments) δηλ. τα ορίσματα που έχουν περάσει στο πρόγραμμά σας χρησιμοποιώντας την γραμμή εντολών.

➔ Εάν χρησιμοποιείτε κάποιο IDE (Integrated Development Enviroment δηλ. ολοκληρωμένο περιβάλλον ανάπτυξης) όπως εγώ στην παρούσα εργασία, για να γράψετε και να τρέξετε αυτά τα προγράμματα, ψάξτε στα μενού για έναν τρόπο να καθορίζετε ορίσματα γραμμής εντολών στο πρόγραμμα.

➔ Εδώ όταν εκτελούμε python using_sys.py we are arguments, τρέχουμε το άρθρωμα using_sys.py με την εντολή python και τα άλλα στοιχεία που ακολουθούν είναι ορίσματα που περνούν στο πρόγραμμα. Η Python αποθηκεύει τα ορίσματα της γραμμής εντολών στη μεταβλητή sys.argv για να τα χρησιμοποιήσετε.

➔ Θυμηθείτε ότι η ονομασία του σεναρίου εντολών που τρέχει είναι πάντα το πρώτο όρισμα στη λίστα sys.argv. Έτσι σε αυτή την περίπτωση θα έχουμε το 'using_sys.py' σαν sys.argv[0], το 'we' σαν sys.argv[1], το 'are' σαν sys.argv[2] και το 'arguments' σαν sys.argv[3]. Σημειώστε ότι **η Python αρχίζει να μετράει από το 0 και όχι από το 1.**

- ⇨ Το sys.path περιέχει τη λίστα με τα ονόματα καταλόγων από όπου εισάγονται τα αρθρώματα. Παρατηρήστε ότι η πρώτη συμβολοσειρά στο sys.path είναι άδεια. Αυτή η άδεια συμβολοσειρά δείχνει ότι ο τρέχων κατάλογος είναι επίσης μέρος του sys.path, ο οποίος είναι το ίδιο με τη μεταβλητή περιβάλλοντος PYTHONPATH. Αυτό σημαίνει ότι μπορείτε απευθείας να εισάγετε αρθρώματα που βρίσκονται στον τρέχοντα κατάλογο. Διαφορετικά πρέπει να τοποθετήσετε το άρθρωμά σας σε έναν από τους καταλόγους που βρίσκονται στο sys.path.
- ⇨ Σημειώστε ότι ο τρέχων κατάλογος είναι ο κατάλογος από όπου το πρόγραμμα εκκινείται. Τρέξτε import os; print(os.getcwd()) για να βρείτε τον τρέχοντα κατάλογο του προγράμματός σας.

Μεταγλωττισμένα Byte αρχεία με επέκταση .pyc

Η εισαγωγή ενός αρθρώματος είναι μια υπόθεση που κοστίζει, έτσι η Python κάνει μερικά κόλπα για να το κάνετε γρηγορότερα. Ένας τρόπος είναι η δημιουργία μεταγλωττισμένων byte αρχείων (byte-compiled files) με την επέκταση .pyc η οποία είναι μια ενδιάμεση μορφή στην οποία η Python μετατρέπει το πρόγραμμα . Αυτό το **αρχείο με επέκταση .pyc** είναι χρήσιμο την επόμενη φορά που θα εισάγετε το άρθρωμα από ένα διαφορετικό πρόγραμμα. Αυτό θα είναι πολύ ταχύτερο γιατί ένα τμήμα της διεργασίας για την εισαγωγή ενός αρθρώματος έχει ήδη γίνει. **Επίσης τα byte-compiled αρχεία είναι ανεξάρτητα πλατφόρμας (platform-independent).**

Τα αρχεία .pyc συνήθως δημιουργούνται στον ίδιο κατάλογο με τα αντίστοιχα αρχεία .py. Εάν η Python δεν έχει άδεια για να γράψει σε αρχεία σε αυτόν τον κατάλογο, τότε τα αρχεία .pyc δε θα δημιουργηθούν.

Η εντολή from ... import ...

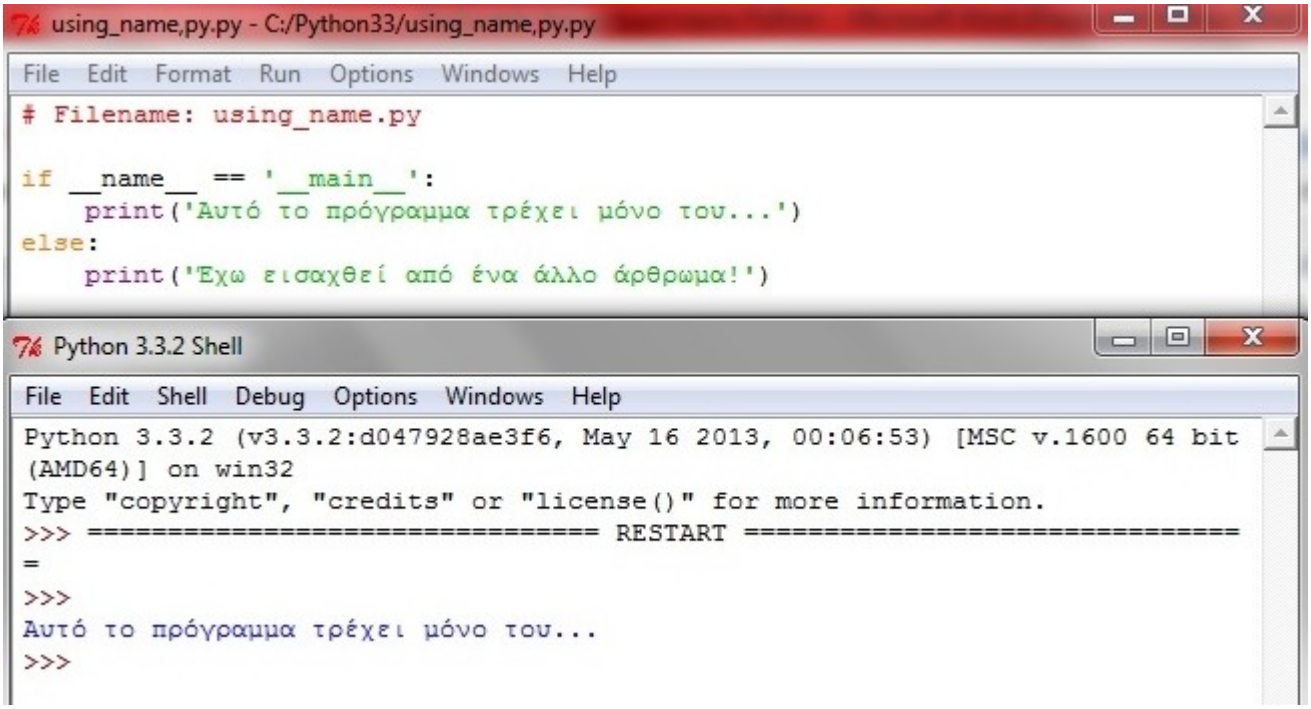
Εάν θέλετε να εισάγετε απευθείας τη μεταβλητή argv μέσα στο πρόγραμμά σας (για να αποφύγετε την πληκτρολόγηση του sys. κάθε φορά), τότε μπορείτε να χρησιμοποιήσετε την εντολή from sys import argv. Εάν θέλετε να εισάγετε όλες τις ονομασίες που χρησιμοποιούνται στο άρθρωμα sys, τότε μπορείτε να χρησιμοποιήσετε την εντολή `from sys import *`. Αυτό λειτουργεί σε κάθε άρθρωμα.

Γενικά πρέπει να αποφεύγετε να χρησιμοποιείτε αυτή την εντολή και αντ' αυτής να χρησιμοποιείτε την εντολή import επειδή έτσι το πρόγραμμά σας θα αποφύγει την σύγκρουση των ονομασιών και θα είναι πιο ευανάγνωστο.

Ονομασία αρθρώματος (module's__name__)

Κάθε άρθρωμα έχει μια ονομασία και οι εντολές σε ένα άρθρωμα μπορούν να ανακαλύψουν το όνομα του αρθρώματός τους. Αυτό είναι εύχρηστο στην ειδική κατάσταση του υπολογισμού για το εάν το άρθρωμα τρέχει μόνο του ή εισάγεται. Όπως αναφέρθηκε προηγουμένως, όταν ένα άρθρωμα εισάγεται για πρώτη φορά, ο κώδικας σε αυτό το άρθρωμα εκτελείται. Μπορούμε να χρησιμοποιήσουμε αυτή την έννοια για να αλλάξουμε τη συμπεριφορά του αρθρώματος εάν το πρόγραμμα χρησιμοποιείται από μόνο του και όχι όταν εισάγεται από ένα άλλο άρθρωμα. Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας το ιδιοχαρακτηριστικό __name__ του αρθρώματος.

Ακολουθεί σχετικό παράδειγμα :



Πώς λειτουργεί το παραπάνω πρόγραμμα :

Σε κάθε άρθρωμα στην Python έχει ορισθεί η ονομασία του __name__, και εάν αυτή είναι '__main__', τότε συνεπάγεται ότι τρέχει μόνο του από το χρήστη και μπορούμε να κάνουμε κανονικές ενέργειες.

Πώς να φτιάξουμε τα δικά μας αρθρώματα;

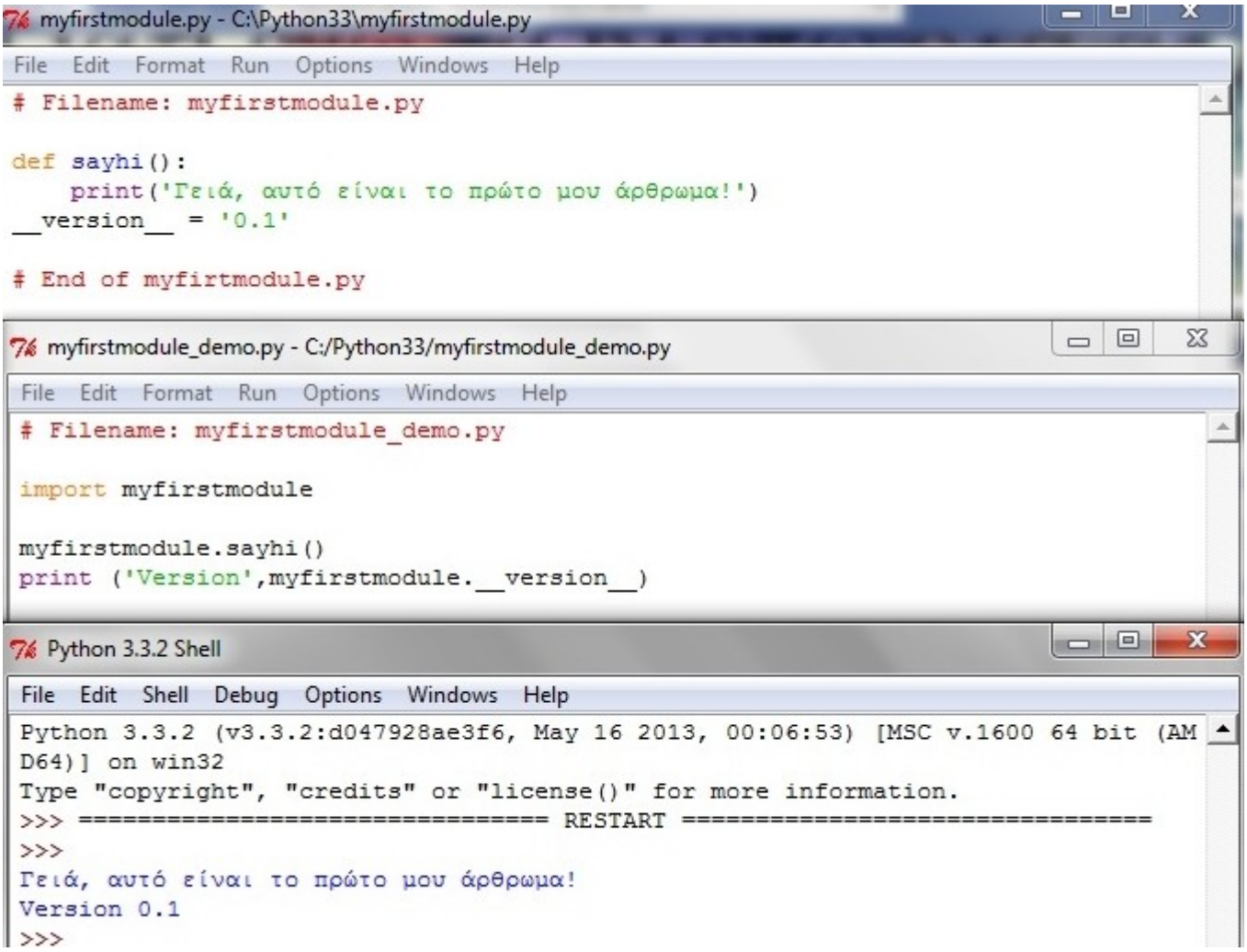
Η δημιουργία των δικών μας αρθρωμάτων είναι εύκολη, το έχουμε κάνει ήδη, κι αυτό διότι κάθε πρόγραμμα στη Python είναι επίσης κι ένα άρθρωμα. Για το μόνο που πρέπει να είμαστε σίγουροι είναι να έχει μια επέκταση .py.

Ακολουθεί ένα παράδειγμα το οποίο στόχο έχει να ξεκαθαρίσει το τοπίο :

```
# Filename: myfirstmodule.py

def sayhi():

    print('Γεια, αυτό είναι το πρώτο μου άρθρωμα!')

__version__ = '0.1'

# End of myfirstmodule.py
```

Το παραπάνω είναι ένα δείγμα 'αρθρώματος. Όπως μπορείτε να δείτε δεν υπάρχει καμμία ιδιαιτερότητα συγκρινόμενο με ένα συνηθισμένο πρόγραμμα σε Python.

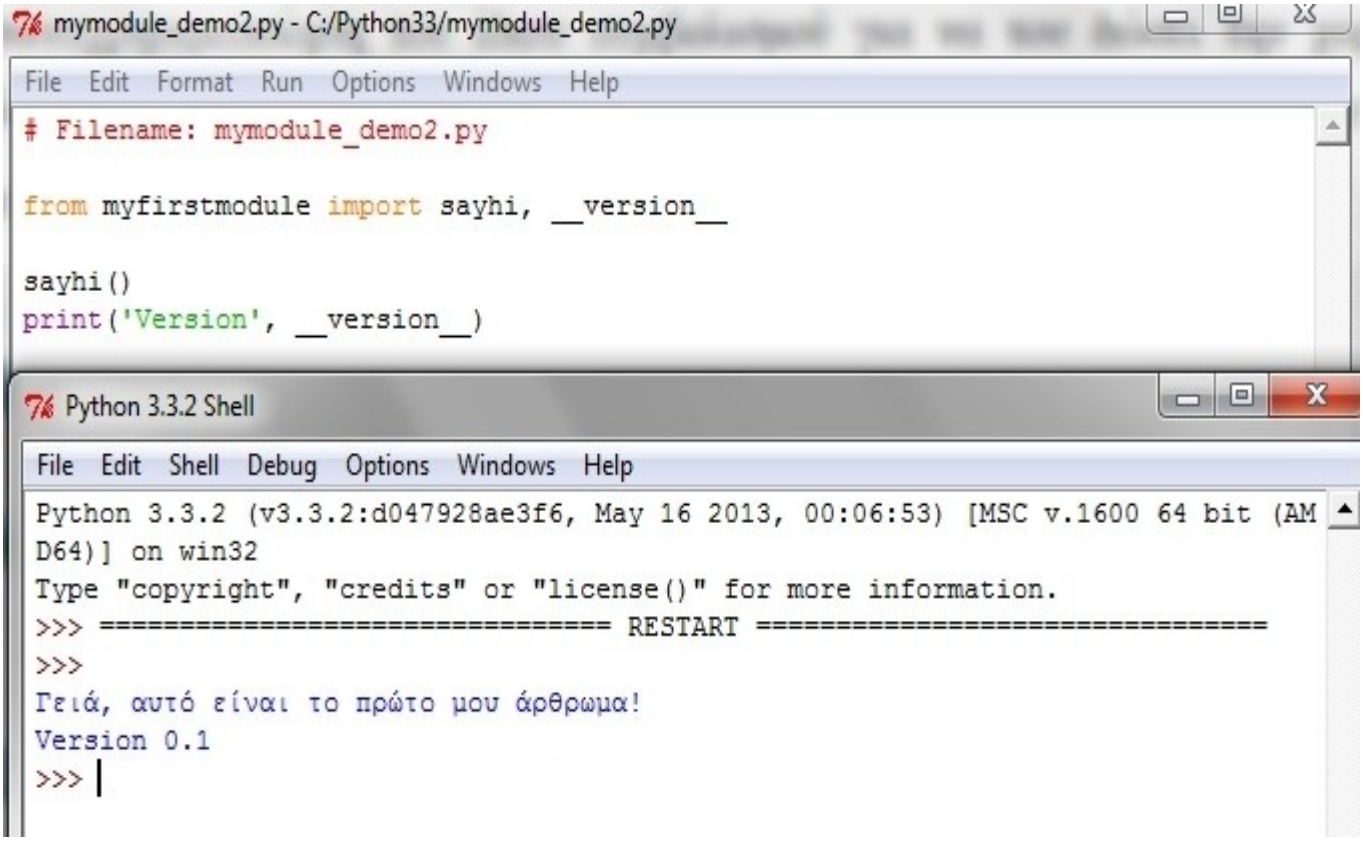
Θυμηθείτε ότι το άρθρωμα πρέπει να τοποθετείται στον ίδιο κατάλογο με το πρόγραμμα που το εισάγουμε, ή το άρθρωμα πρέπει να βρίσκεται σε έναν από τους καταλόγους που είναι στη λίστα του sys.path.



Παρατηρούμε ότι χρησιμοποιούμε τον ίδιο συμβολισμό με τελείες για να εισάγουμε μέλη στο άρθρωμα. Η Python κάνει σωστή επαναχρησιμοποίηση του ίδιου

συμβολισμού για να του δώσει την χαρακτηριστική Pythόνια αίσθηση, έτσι ώστε να μην χρειάζεται να μαθαίνουμε καινούργιους τρόπους για να κάνουμε πράγματα.

- Ας δούμε τώρα μια εκδοχή όπου γίνεται χρήση της σύνταξης from..import :



Βλέπουμε ότι το αποτέλεσμα του mymodule_demo2.py είναι το ίδιο με το αποτέλεσμα του myfirstmodule_demo.py.

Σημαντική παρατήρηση → Εάν ήταν ήδη δηλωμένο το όνομα της έκδοσης `__version__` στο άρθρωμα το οποίο εισάγει το myfirstmodule, θα γινόταν σύγκρουση. Αυτό είναι επίσης πιθανόν διότι είναι κοινή πρακτική για κάθε άρθρωμα να δηλώνει το νούμερο έκδοσής του χρησιμοποιώντας αυτό το όνομα. Γι' αυτό το λόγο συνιστάται να προτιμάτε την εντολή import ακόμα κι αν έτσι γίνεται το πρόγραμμά σας μεγαλύτερο.

Επίσης θα μπορούσατε να χρησιμοποιήσετε:

```
from mymodule import *
```

Αυτό θα εισήγαγε όλα τα δημόσια ονόματα όπως το sayhi αλλά δε θα εισήγαγε το `__version__` επειδή αρχίζει με διπλές κάτω παύλες.

Η συνάρτηση dir

Μπορείτε να χρησιμοποιήσετε την ενσωματωμένη συνάρτηση dir για να δείτε μια λίστα με τα αναγνωριστικά που ορίζει ένα αντικείμενο. Για παράδειγμα, σε ένα άρθρωμα, τα αναγνωριστικά περιλαμβάνουν τις συναρτήσεις, τις κλάσεις και τις μεταβλητές που ορίζονται σε αυτό το άρθρωμα.

Όταν δίνετε το όνομα ενός αρθρώματος στη συνάρτηση dir, αυτή επιστρέφει τη λίστα των ονομάτων που ορίζονται σε αυτό το άρθρωμα. Όταν δεν παρέχεται κανένα όρισμα, τότε επιστρέφει τη λίστα των ονομάτων που ορίζονται στο τρέχον άρθρωμα.

```
>>> import sys
```

παίρνει λίστα των ιδιοχαρακτηριστικών, σε αυτή την περίπτωση, για το άρθρωμα sys

```
>>> dir(sys)
```
(εμφανίζει μια τεράστια λίστα ιδιοχαρακτηριστικών)

```
>>> dir()
```

δίνει λίστα ιδιοχαρακτηριστικών για το τρέχον άρθρωμα

```
>>> a = 5          # δημιουργεί μια νέα μεταβλητή 'a'
```

```
>>> dir()
```

```
>>> del a          # διαγράφει ή αφαιρεί ένα όνομα
```

```
>>> dir()
```

Εξήγηση →

- Αρχικά βλέπουμε τη χρήση της συνάρτησης dir στο εισαχθέν άρθρωμα dir. Μπορούμε να δούμε την τεράστια λίστα των ιδιοχαρακτηριστικών που περιέχει.
- Κατόπιν χρησιμοποιούμε την συνάρτηση dir χωρίς να της περάσουμε παραμέτρους. Από προεπιλογή, επιστρέφει τη λίστα του τρέχοντος αρθρώματος. Παρατηρήστε ότι η λίστα των εισαχθέντων αρθρωμάτων είναι επίσης μέρος αυτής της λίστας.
- Για να δείτε σε δράση τη dir, ορίζουμε μια καινούργια μεταβλητή a και εκχωρούμε σ' αυτή μια τιμή, μετά τσεκάρουμε τη dir και παρατηρούμε ότι υπάρχει μια επιπρόσθετη τιμή στη λίστα του ίδιου ονόματος.

- Αφαιρούμε τη μεταβλητή/ιδιοχαρακτηριστικό του τρέχοντος αρθρώματος χρησιμοποιώντας την εντολή del και η αλλαγή αντανακλάται πάλι στο αποτέλεσμα της συνάρτησης dir.

Σχόλια – Παρατηρήσεις :

→ Η εντολή del χρησιμοποιείται για να διαγράψει μια μεταβλητή/όνομα και αφού η εντολή έχει τρέξει, σε αυτή την περίπτωση del a, δε μπορείτε πια να εισάγετε τη μεταβλητή a –είναι σαν να μην υπήρξε πριν καθόλου.

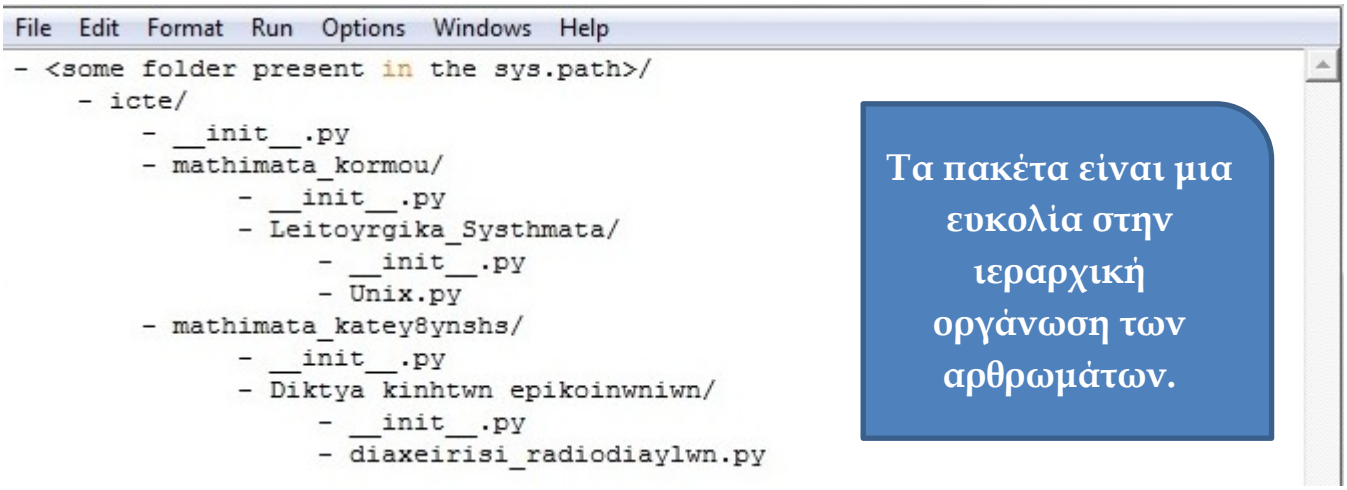
→ Η συνάρτηση dir() λειτουργεί σε οποιοδήποτε αντικείμενο. Για παράδειγμα, τρέξτε dir(print) για να μάθετε σχετικά με τα ιδιοχαρακτηριστικά της συνάρτησης print, ή dir(str) για τα ιδιοχαρακτηριστικά της κλάσης str.

Πακέτα (Packages)

Υπάρχει ιεραρχία με την οποία οργανώνονται τα προγράμματά στην Python. Οι μεταβλητές συνήθως πηγαίνουν μέσα στις συναρτήσεις. Οι συναρτήσεις και οι καθολικές μεταβλητές συνήθως πηγαίνουν μέσα στα αρθρώματα. Τι θα γινόταν όμως αν θέλατε να οργανώσετε τα αρθρώματα; Γι' αυτό έρχονται εδώ στο προσκήνιο **τα πακέτα**.

Τα πακέτα είναι απλώς φάκελοι αρθρωμάτων με ένα ειδικό αρχείο `__init__.py` που δείχνει στην Python ότι αυτός ο φάκελος είναι ειδικός διότι περιέχει αρθρώματα Python.

Ας προχωρήσουμε σε ένα παράδειγμα για να γίνει πιο κατανοητή η χρήση των πακέτων : Ας πούμε ότι θέλετε να δημιουργήσουμε ένα πακέτο που ονομάζεται 'icte' με υποπακέτα που ονομάζονται 'mathimata_kormou', 'mathimata_katey8inshs' και αυτά τα υποπακέτα με τη σειρά τους περιέχουν αρθρώματα όπως 'Leitoyrgika_Systhmata', 'Diktya Kinhtwn epikoinwniwn'.

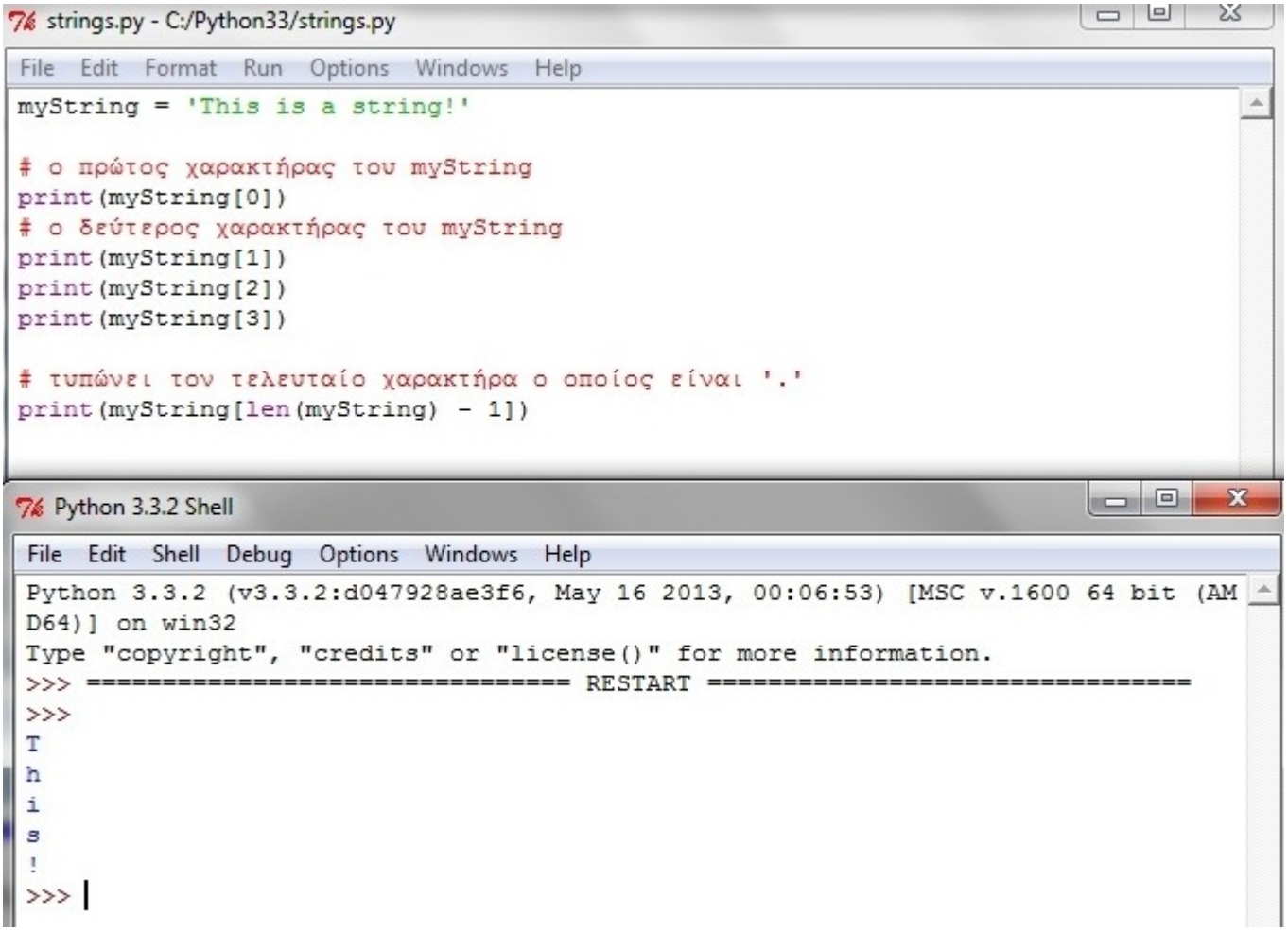


Αλφαριθμητικά

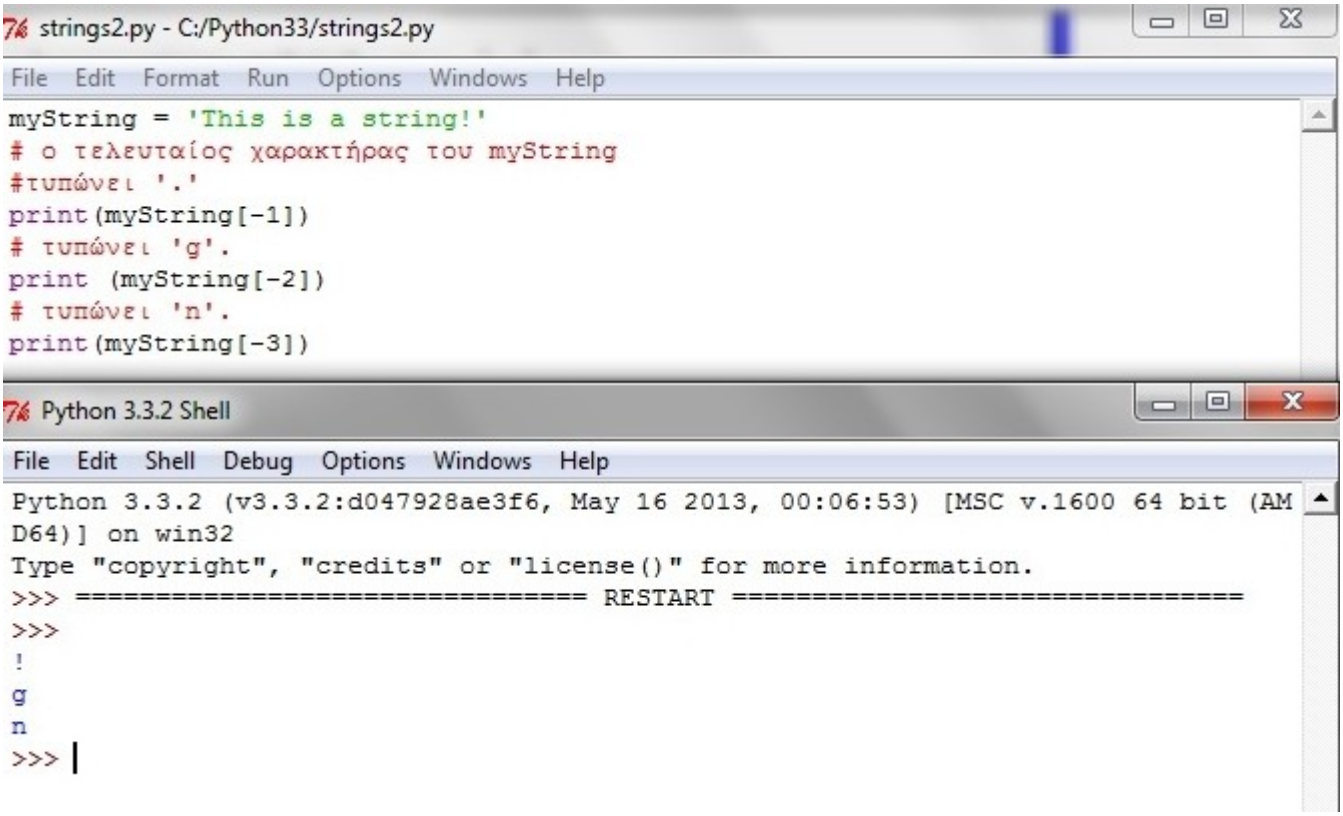
Βασικά στοιχεία αλφαριθμητικών

Τα αλφαριθμητικλα, όπως και οι πλειάδες (που θα αναλύσουμε παρακάτω) , είναι αμετάβλητα, που σημαίνει έπειτα από την δημιουργία τους δεν μπορούν να αλλάξουν τιμή. Ο τελεστής ανάθεσης, μπορεί να κάνει μια μεταβλητή να δείχνει σε ένα αλφαριθμητικό όπως και στους απλούς τύπους. Η προεπιλεγμένη κωδικοποίηση για ένα αλφαριθμητικό είναι UTF-8.

Τα αλφαριθμητικά αποτελούν ένα είδος ακολουθίας(sequence). Αυτό σημαίνει πως κάθε στοιχείο είναι σε μια αριθμημένη σειρά, με βάση και την οποία μπορεί να προσπελαστεί.

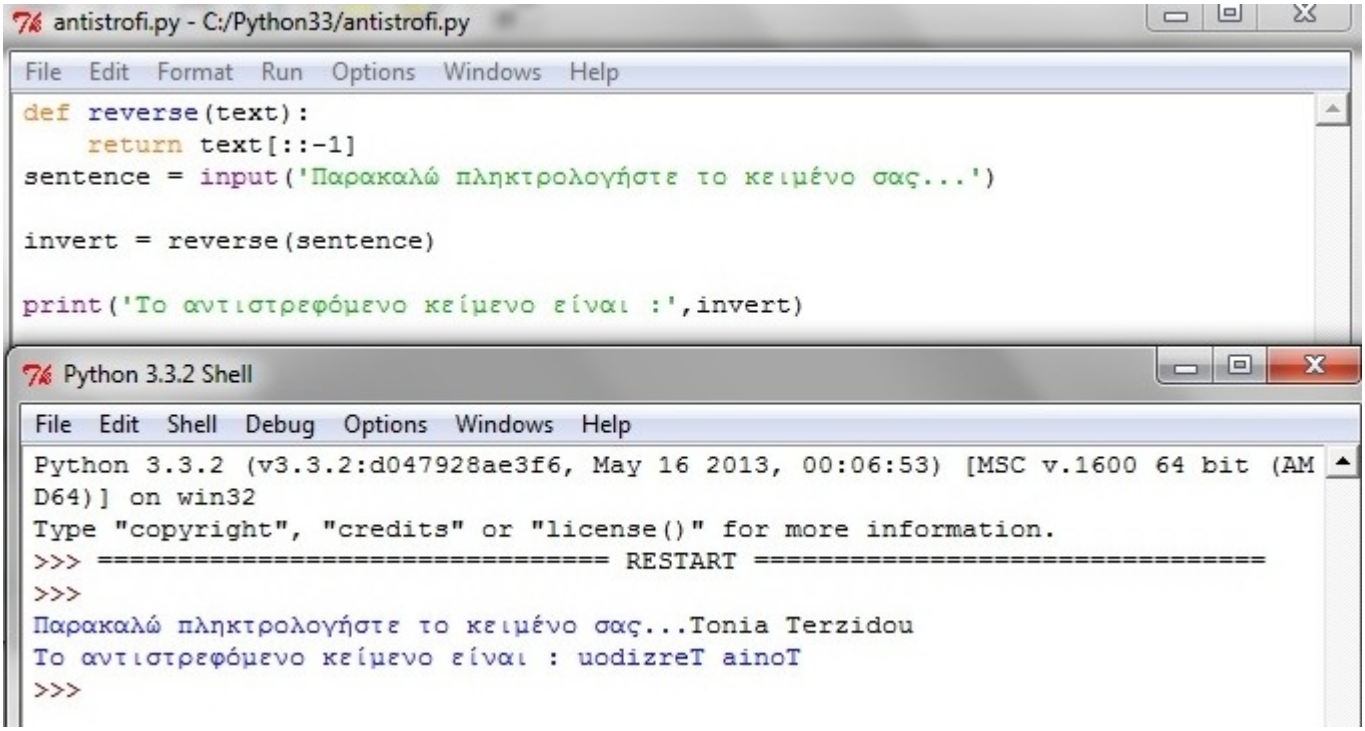


Μπορούμε να αρχίσουμε την προσπέλαση των στοιχείων μετρώντας από το τέλος του αλφαριθμητικού προς την αρχή του αν χρησιμοποιήσουμε το σύμβολο - (μείον) πριν από τον δέκτη (index).

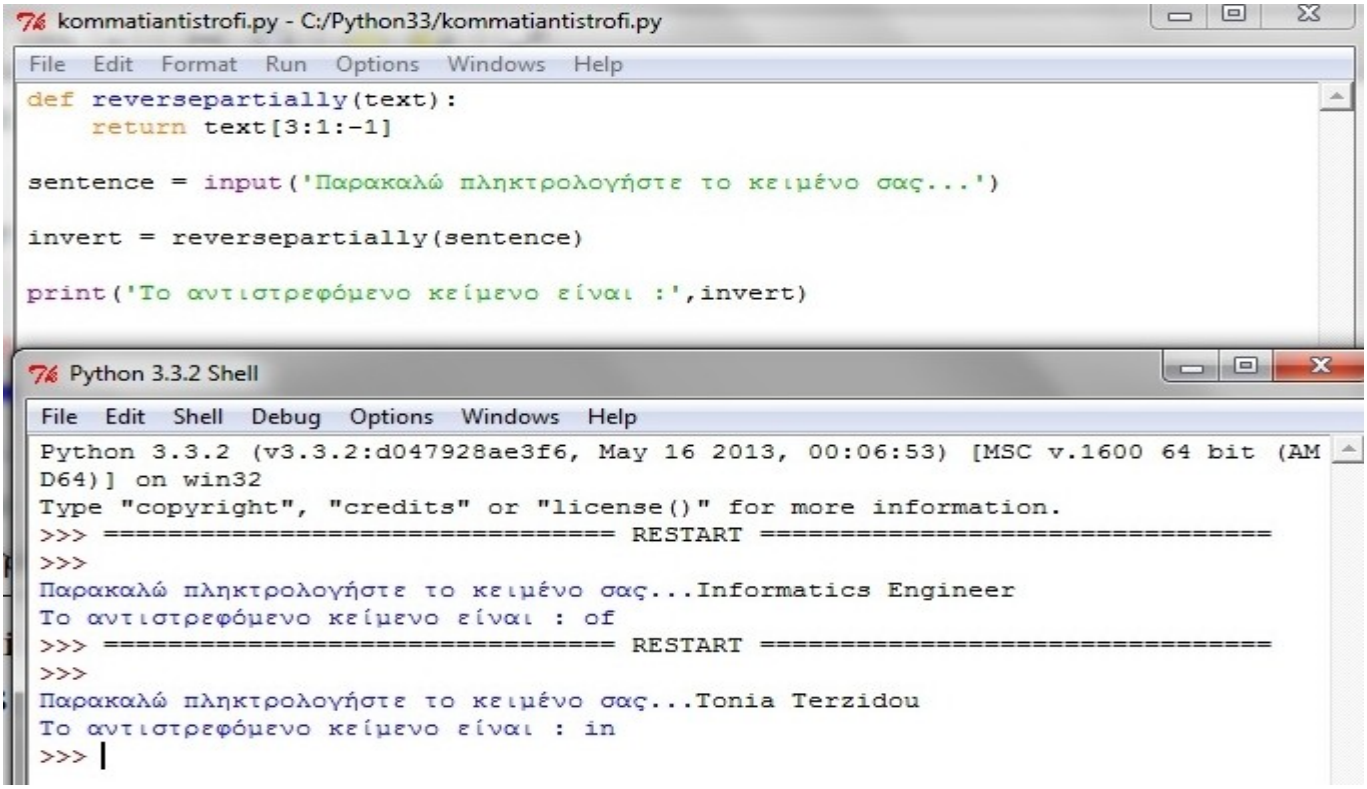


Αντιστροφή Αλφαριθμητικού

Αφού μπορούμε να προσπελάσουμε ένα αλφαριθμητικό ως μια ακολουθία, μπορούμε να χρησιμοποιήσοθμε όποια γνωρίσματα μιας ακολουθίας επιθυμούμε για να επιτύχουμε επιθυμητές ενέργειες. Έτσι, για παράδειγμα η αντιστροφή ενός αλφαριθμητικού μπορεί να γίνει πολύ εύκολα.



Θα μπορούσαμε αντίστοιχα να αντιστρέψουμε μόνο ένα κομμάτι του αλφαριθμητικού με τον παρακάτω τρόπο :



Πως δουλεύει αυτό το πρόγραμμα :

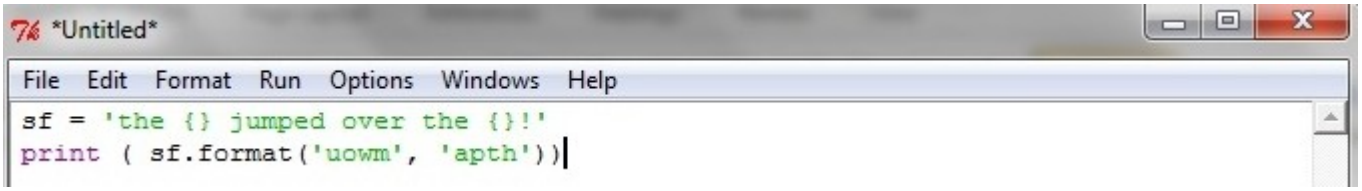
Ξεκινάμε από τον χαρακτήρα με δείκτη 3 και θα τυπώσουμε όλους μέχρι και τον χαρακτήρα με δείκτη 2, ένα πριν δηλαδή. Τονίζω πως στην Python, όπως και σε πολλές ακόμη άλλες γλώσσες προγραμματισμου, η δεικτοδότηση αρχίζει από το μηδέν (0).

Μορφοποίηση αλφαριθμητικού

Η μορφοποίηση ενός αλφαριθμητικού μπορεί να γίνει μέσω της συνάρτησης

`string.format()` .Στην Python, τα αλφαριθμητικά strings είναι ο προεπιλεγμένος τύπος και δεν χρειάζονται κάποια δήλωση.

Για να μπορέσουμε να δείξουμε ποια μεταβλητή θα πάει σε ποια θέση, χρησιμοποιούμε το `{x}` ,όπου x είναι ένας δείκτης που δείχνει ποια από τις ακόλουθες μεταβλητές θα τοποθετηθεί στην θέση του.



Δομές δεδομένων

Οι δομές δεδομένων μας παρέχουν έναν αποτελεσματικό τρόπο διαχείρησης της διαθέσιμης πληροφορίας. Αποτελούν την ραχοκοκαλιά σχεδόν κάθε προγράμματός μας και η κατάλληλη επιλογή του μπορεί να οδηγήσει σε πολύ μεγάλα οφέλη στην ταχύτητα εκτέλεσης των υπολογισμών. Ο αποτελεσματικός τους σχεδιασμός και η υλοποίησή τους αποτελούν σημαντικό μέρος της επιστημονικής έρευνας στον χώρο των υπολογιστών. Συνήθως όμως, ένας προγραμματιστής δεν χρειάζεται να τις κατασκευάσει από την αρχή και μπορεί να χρησιμοποιήσει κάποια από αυτές που μοιράζονται με τη βασική βιβλιοθήκη, η οποία είναι αρκετά πλούσια στην Python και η οποία μας διευκολύνει αρχικά στην χρήση τους.

Άρα, οι δομές δεδομένων είναι αυτό που λέει το όνομά τους, δηλαδή είναι δομές που μπορούν να κρατήσουν μαζί μερικά δεδομένα. Με άλλα λόγια χρησιμοποιούνται για να αποθηκεύουν δεδομένα που έχουν σχέση μεταξύ τους.

Υπάρχουν τέσσερις δομές δεδομένων ενσωματωμένες στη Python:

1. οι λίστες
2. οι πλειάδες
3. τα λεξικά
4. τα σύνολα

Λίστα

Μια λίστα είναι μια δομή δεδομένων που συγκρατεί μια διατεταγμένη συλλογή στοιχείων, δηλαδή μπορείτε να αποθηκεύσετε μια ακολουθία (sequence) αντικειμένων στη λίστα. Αυτό είναι εύκολο να το φανταστείτε αν σκεφτείτε μια λίστα για αγορές, όπου έχετε μια λίστα με αντικείμενα που πρέπει να αγοράσετε, και εκτός αυτού πιθανόν έχετε κάθε στοιχείο σε διαφορετική γραμμή στη λίστα αγορών σας, ενώ στην Python τοποθετείτε κόμματα ανάμεσα στα στοιχεία.

Η λίστα των στοιχείων πρέπει να κλείνεται σε αγκύλες (δηλαδή [και]) έτσι ώστε να καταλαβαίνει η Python ότι καθορίζετε μια λίστα. Αφού έχετε δημιουργήσει μια λίστα μια φορά μπορείτε να προσθέσετε, να μετακινήσετε ή να ψάξετε για στοιχεία σ' αυτή τη λίστα. Από τη στιγμή που μπορούμε να προσθέσουμε και να μετακινήσουμε στοιχεία, λέμε ότι η λίστα είναι ένας ***μεταβλητός τύπος δεδομένων (mutable data type)***, δηλαδή αυτός ο τύπος μπορεί να αλλαχθεί.

Μια σύντομη ματιά στα αντικείμενα (objects) και τις κλάσεις (classes)

Η λίστα είναι ένα παράδειγμα χρήσης αντικειμένων και κλάσεων. Όταν χρησιμοποιούμε τη μεταβλητή i και εκχωρούμε σ' αυτήν μια τιμή, ας πούμε τον ακέραιο αριθμό 5, αυτό μπορούμε να το σκεφτούμε σαν τη δημιουργία ενός αντικειμένου (δηλ. υπόστασης (instance)) i της κλάσης (δηλ. τύπου (type)) int. Στην πραγματικότητα μπορείτε να διαβάσετε τη βοήθεια στο help(int) για να το καταλάβετε καλύτερα.

Μια κλάση μπορεί επίσης να έχει μεθόδους (methods), δηλαδή συναρτήσεις που ορίστηκαν για χρήση που έχει σχέση μόνο με αυτή την κλάση. Μπορείτε να χρησιμοποιήσετε αυτά τα μέρη λειτουργικότητας μόνο όταν έχετε ένα αντικείμενο σε αυτή την κλάση. Για παράδειγμα η Python παρέχει μια μέθοδο append για την κλάση list, που σας επιτρέπει να προσθέσετε ένα αντικείμενο στο τέλος της λίστας. Για παράδειγμα, το mylist.append('an item') θα προσθέσει αυτή τη συμβολοσειρά στο τέλος του mylist. Σημειώστε τη χρήση του συμβολισμού με τελείες για να εισαχθούν οι μέθοδοι των αντικειμένων.

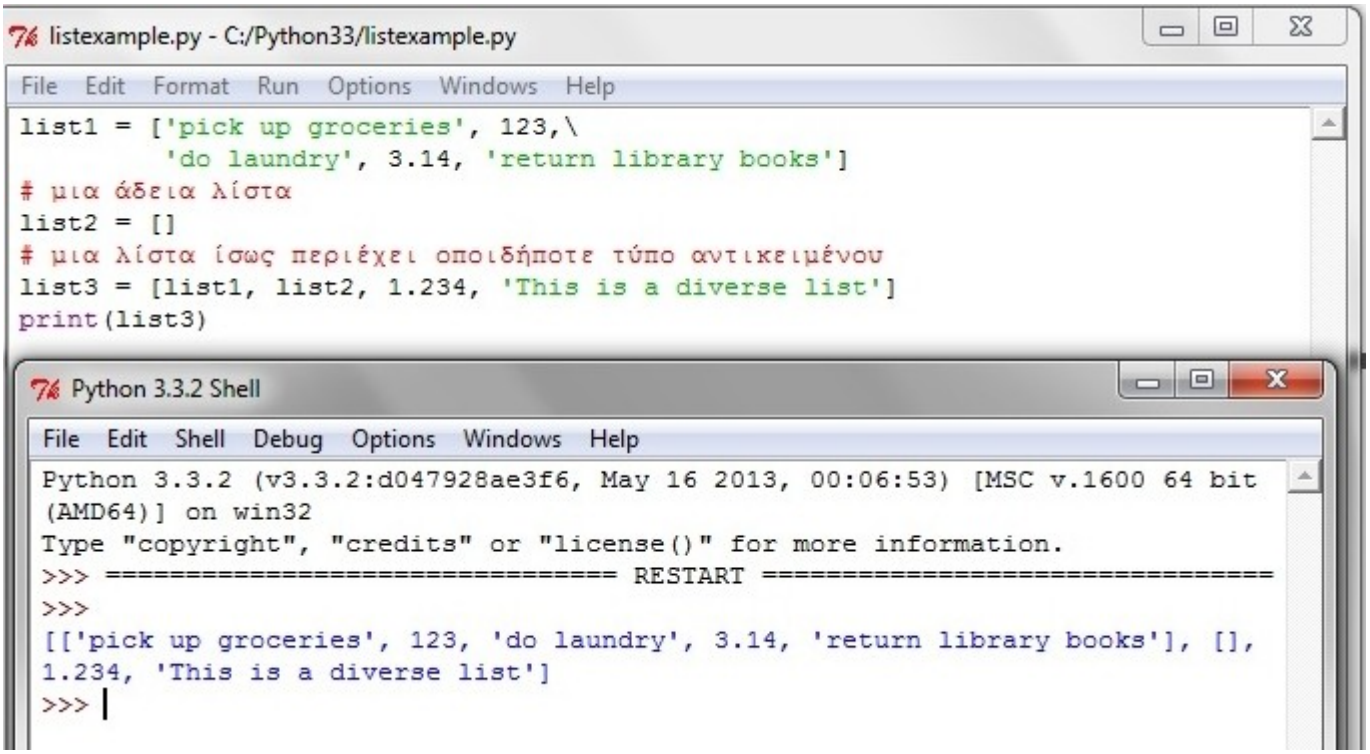
Μια κλάση μπορεί επίσης να έχει **πεδία (fields)**, που δεν είναι τίποτα άλλο παρά μεταβλητές που ορίστηκαν για χρήση που έχει σχέση μόνο με αυτή την κλάση. Μπορείτε να χρησιμοποιήσετε αυτές τις μεταβλητές/ονομασίες μόνο όταν έχετε ένα αντικείμενο αυτής της κλάσης. Τα πεδία επίσης εισάγονται με συμβολισμό με τελείες, για παράδειγμα, mylist.field.

Οι λίστες χρησιμοποιούνται αρκετά συχνά στον προγραμματισμό. Αντικαθιστούν την έννοια του πίνακα, που γνωρίζουμε από άλλες γλώσσες προγραμματισμού. Η κύρια διαφορά τους στην Python είναι ότι γίνεται αυτόματα η κατάλληλη δέσμευση της μνήμης που αυτοί απαιτούν. Επομένως δεν πρέπει να τις συγχέουμε με τις διασυνδεδεμένες λίστες που ίσως γνωρίζουμε . Η συμπεροφορά τους έχει όλα τα γνωρίσματα των πινάκων.

Για να ορίσουμε μια λίστα, αρκεί να περιλάβουμε μέσα σε αγκύλες τα αντικείμενα που θέλουμε να περιέχει.

Όπως βλέπουμε παρακάτω, μια λίστα μπορεί να περιέχει οποιουδήποτε τύπου αντικείμενα. Έτσι, μπορούμε να προσθέσουμε δικά μας αντικείμενα, άλλες λίστες ακόμα και αντικείμενα που αναπαριστούν συναρτήσεις.

Παραθέτω λοιπόν το εξής παράδειγμα για να στηρίξω όσα προειπώθηκαν :

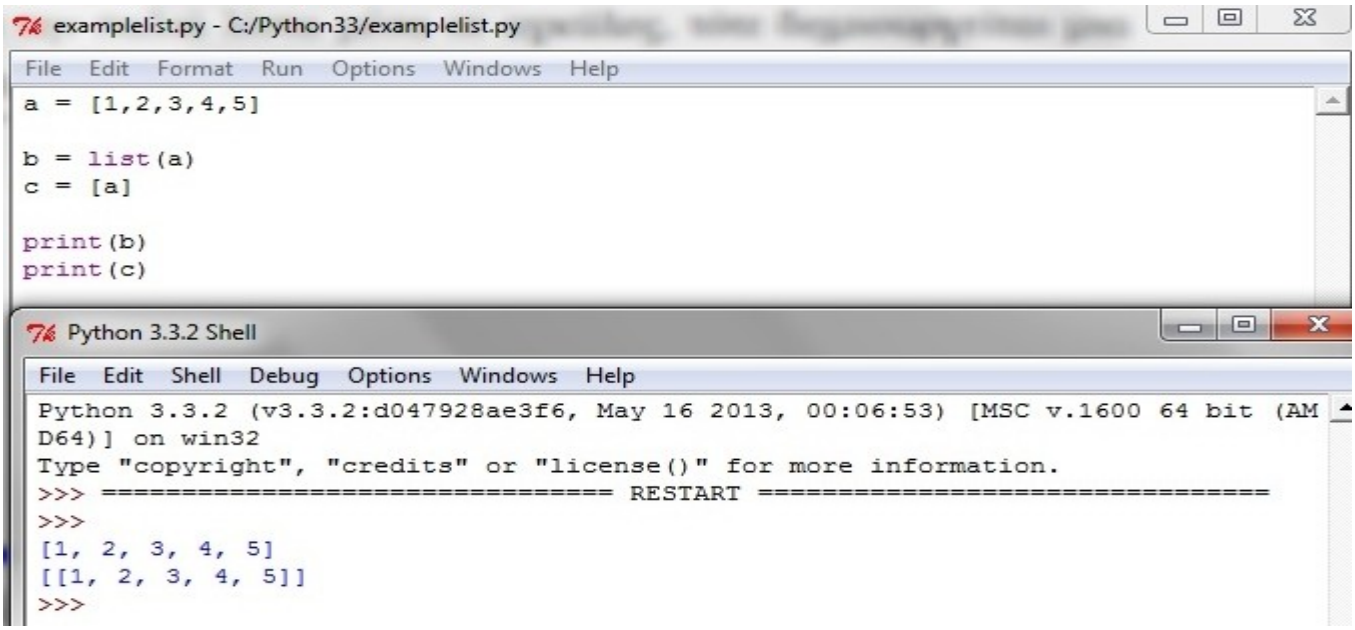


Δημιουργία λίστας

Ο πιο απλός τρόπος να δημιουργήσουμε μια κενή λίστα είναι με την χρήση των τετράγωνων αγκυλών.

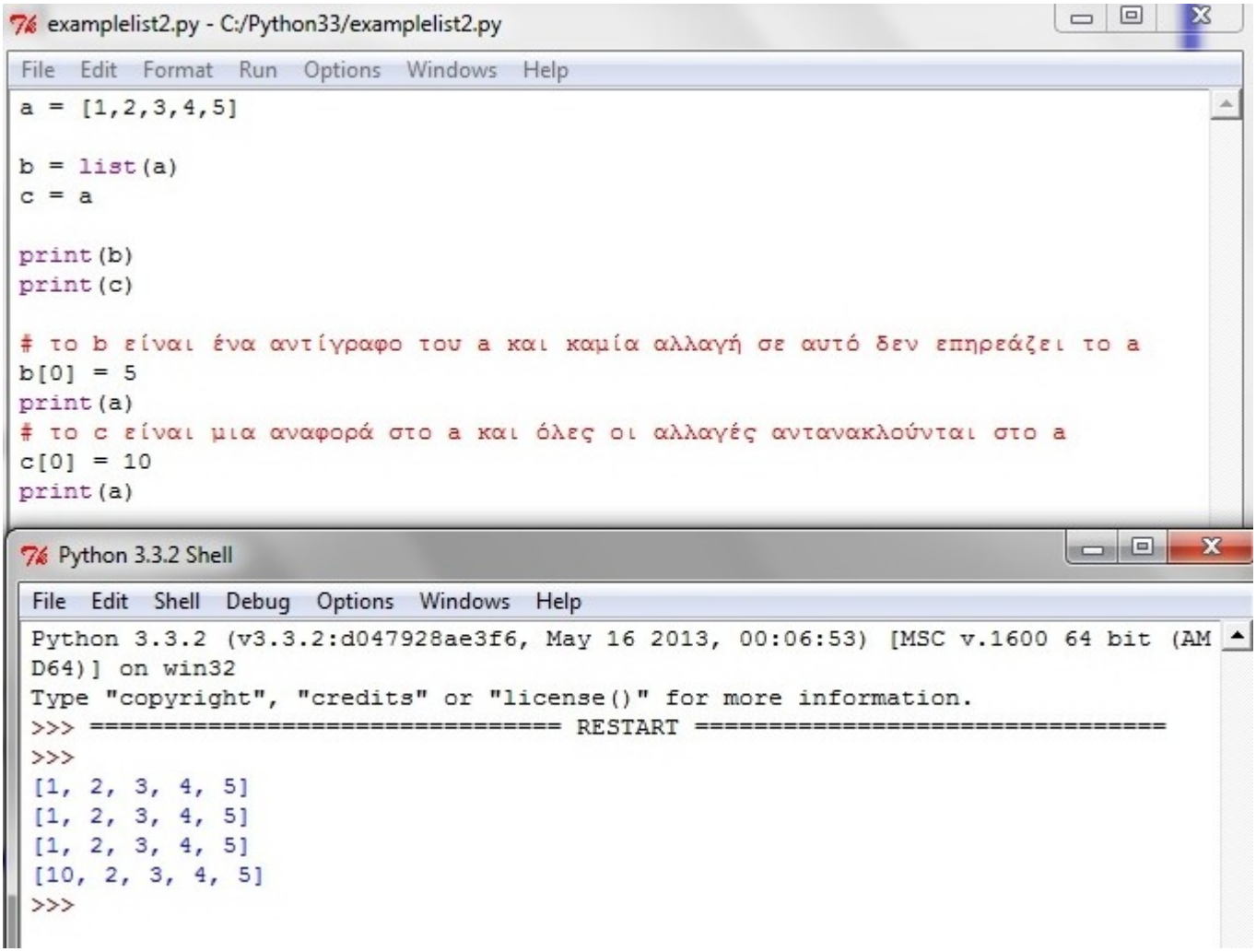
List = []

Υπάρχουν και λεπτομέρειες που συχνά μας ξεφεύγουν όσον αφορά τις λίστες. Αν χρησιμοποιήσουμε την συνάτηση list() σε μια λίστα, τότε μας επιστρέφεται ένα αντίγραφο αυτής της λίστας χωρίς να έχει αλλάξει κάτι. Αν όμως περιλάβουμε την παλιά λίστα μέσα σε αγκύλες, τότε δημιουργείται μια καινούρια που περιέχει ένα αντικείμενο.



Προσοχή στην διαφορά που παρατηρείται αν χρησιμοποιήσουμε την συνάρτηση list() ή όχι. Αυτό γιατί η list() δημιουργεί ένα αντίγραφο της λίστας από προεπιλογή, ενώ αλλιώς η Python επιστρέφει απλώς μια αναφορά για το αντικείμενο που πλέον μπορούμε να το προσπελάσουμε με έναν ακόμα τρόπο.

Ακολουθεί σχετικό παράδειγμα :



Πρόσβαση σε στοιχεία λίστας

Η πρόσβαση στα στοιχεία μιας λίστας είναι αρκετά εύκολη, αρκεί να θυμάστε πως η μέτρηση, όπως σχεδόν σε όλα στους υπολογιστές, αρχίζει από το μηδέν και όχι από το ένα. Επομένως το πρώτο στοιχείο μίας λίστας αριθμείται με το μηδέν και όλα τα μετέπειτα έχουν δείκτη αυξημένο κατά ένα. Ένα άλλο σημαντικό στοιχείο που πρέπει να θυμόμαστε όταν προσπελαύνουμε στοιχεία είναι ότι το τελευταίο στοιχείο όταν διαλέγουμε από ένα σύνολο δεν είναι το στοιχείο list1[j] αλλά το `list1[j-1]`. Αν το σκεφτούμε λίγο είναι λογικό , καθώς αν μια λίστα έχει n στοιχεία , το τελευταίο της δεν είναι το list1[n] αλλά το `list1[n-1]` αφού η αρίθμηση αρχίζει από το μηδέν.

- **Πλήθος στοιχείων**

```
n = len (list1)
```

- **Πρώτο στοιχείο**

```
list1[0]
```

- **Στοιχεία στο [i,j):**

```
list1[ iI : j ]
```

Εκτός από το να διαλέξουμε από πού εώς που θα πάρουμε τα στοιχεία μιας λίστας, μπορούμε επίσης να καθορίσουμε και ανα πόσα στοιχεία θα τα λαμβάνουμε.

Αν έχουμε μια λίστα με όλους τους αριθμούς μέχρι το 10 και θέλουμε τους άρτιους από τον δεύτερο αριθμό εως τον όγδοο(χωρίς να τον συμπεριλαμβάνουμε) τότε μπορούμε να κάνουμε ακολούθως , όπου ο τελευταίος αριθμός στις αγκύλες είναι το βήμα :

```
a = [x for x in range (10)]

# μπορούμε επίσης να γράψουμε a =list(range(10))

print (a[2:8:2])
```

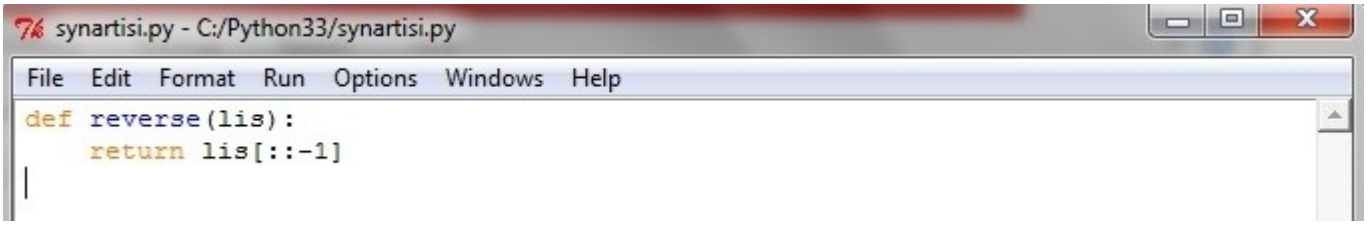
- **Στοιχεία στο [i,j) ανά k**

```
list1[i:j:k]
```

- **Το k μπορεί να πάρει αρνητικές τιμές**

```
list1[ :: k]
```

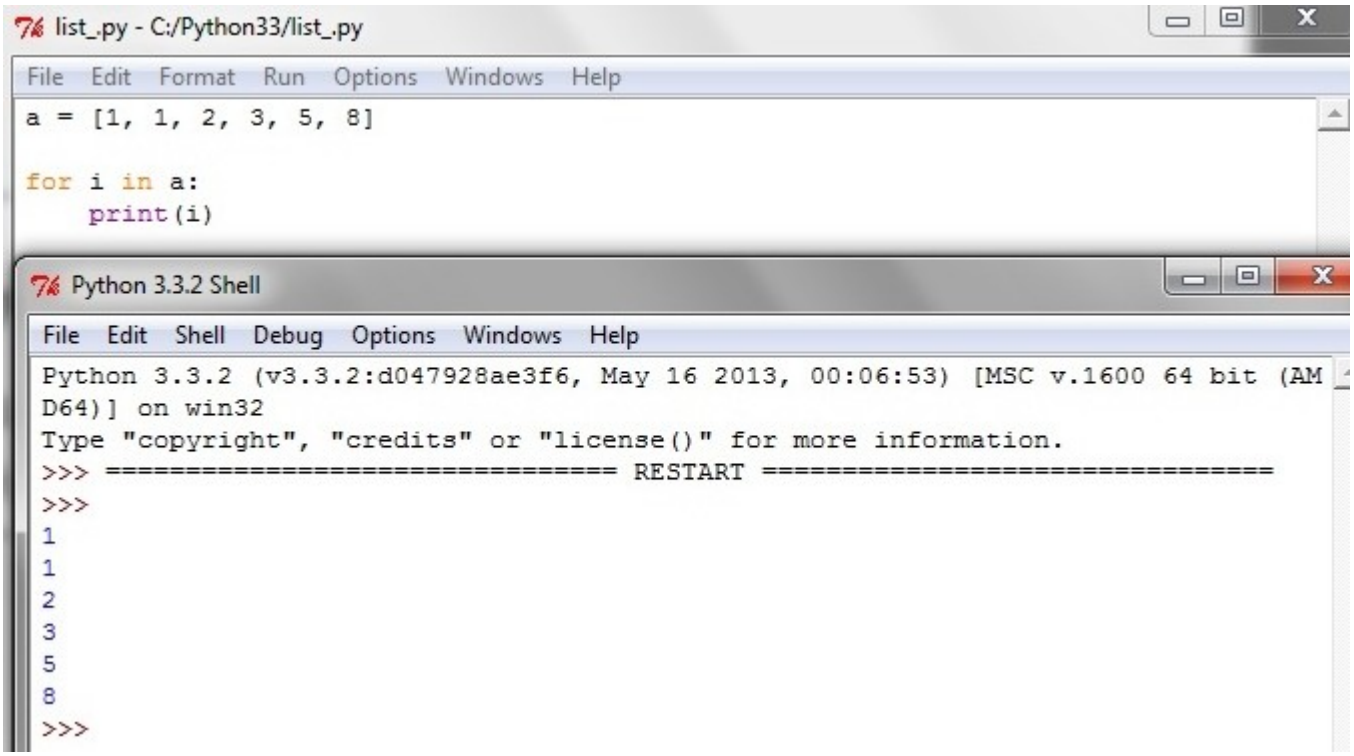
Παρατηρούμε ότι το k, που αποτελεί το βήμα μπορεί να πάρει αρνητικές τιμές, επομένως μπορούμε έτσι εύκολα να κατασκευάσουμε μια συνάρτηση αντιστροφής ενός πίνακα.



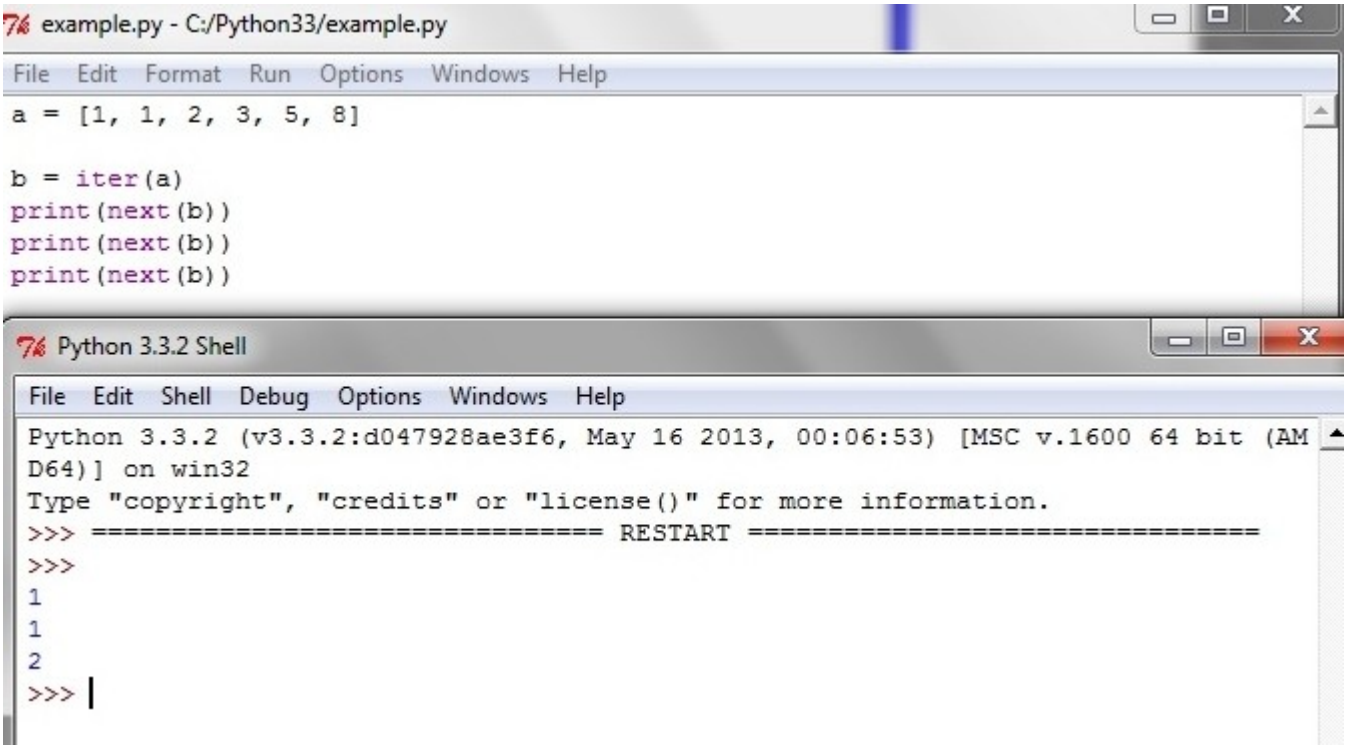
Διάτρεξη στοιχείων λίστας

Μπορούμε πολύ εύκολα να διατρέξουμε τα στοιχεία μίας λίστας ένα ένα χρησιμοποιώντας μια απλή δομή for.

Ακολουθεί παράδειγμα :



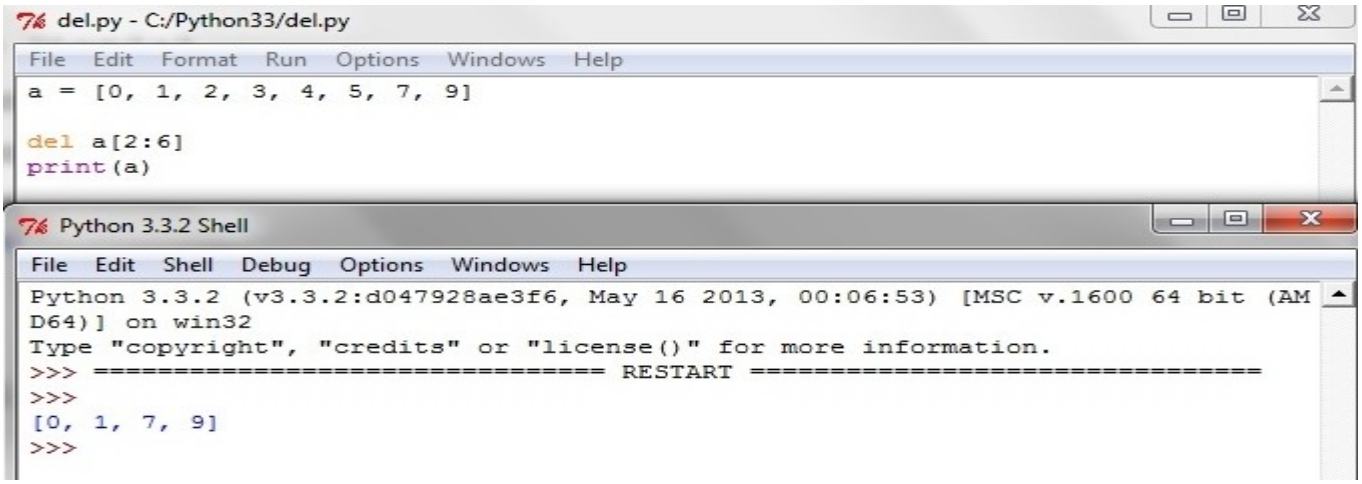
Αν θέλουμε, μπορούμε να χρησιμοποιήσουμε επαναλήπτες για την διάτρεξη των στοιχείων μιας λίστας , οι οποίοι αποτελούν και μια εισαγωγή στην έννοια των γεννήτορων.



Διαγραφή στοιχείων

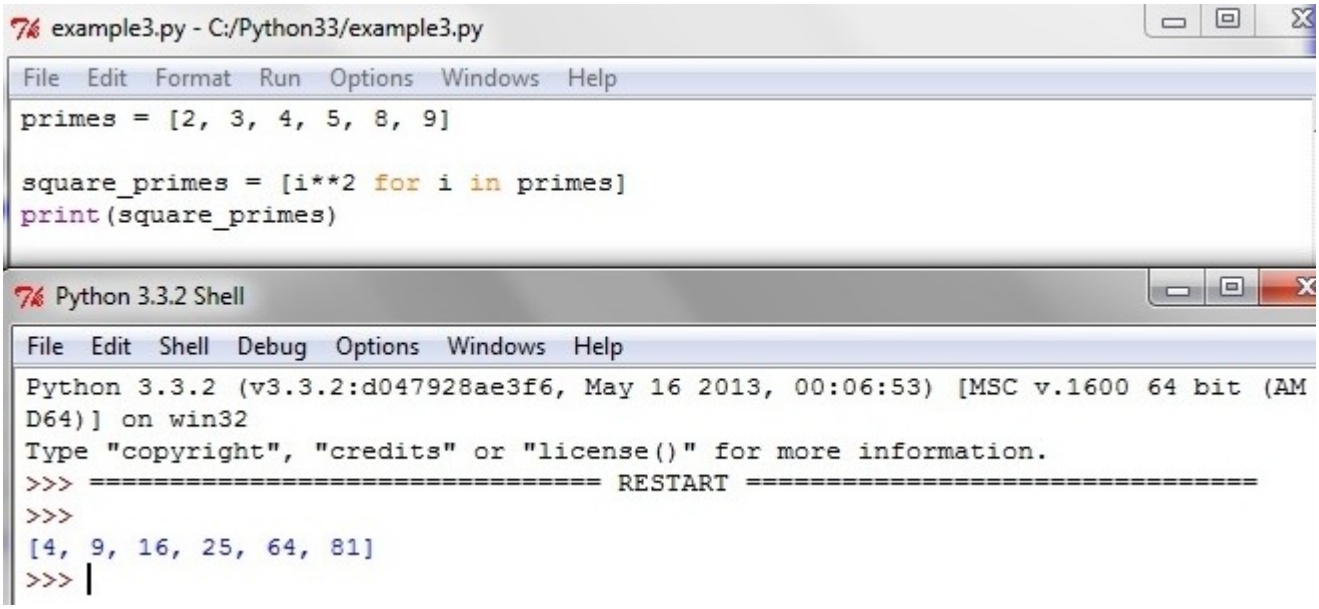
Για να διαγράψουμε κάποια στοιχεία από μια λίστα, προσθέτουμε μπροστά την λέξη κλειδί del και στην συνέχεια το κομμάτι της λίστας το οποίο θέλουμε να διαγράψουμε. Αυτό το κομμάτι το προσδιορίζουμε όπως ακριβώς θα το προσδιορίζαμε και αν θέλαμε να κάνουμε προσπέλαση στα στοιχεία αυτού του κομματιού.

Ακολουθεί παράδειγμα :



Κατανοήσεις λίστας (Lists comprehensions)

Μπορούμε επίσης να δημιουργήσουμε μια καινούργια λίστα με την χρήση κατανοήσεων λίστας (list comprehensions) . Στο παράδειγμα που ακολουθεί βλέπουμε μια ήδη υπάρχουσα λίστα της οποίας τα στοιχεία διατρέχονται ένα ένα και κάθε ένα το υψώνουμε στο τετράγωνο. Το υψωμένο στο τετράγωνο στοιχείο πλέον ανήκει στην καινούργια λίστα.

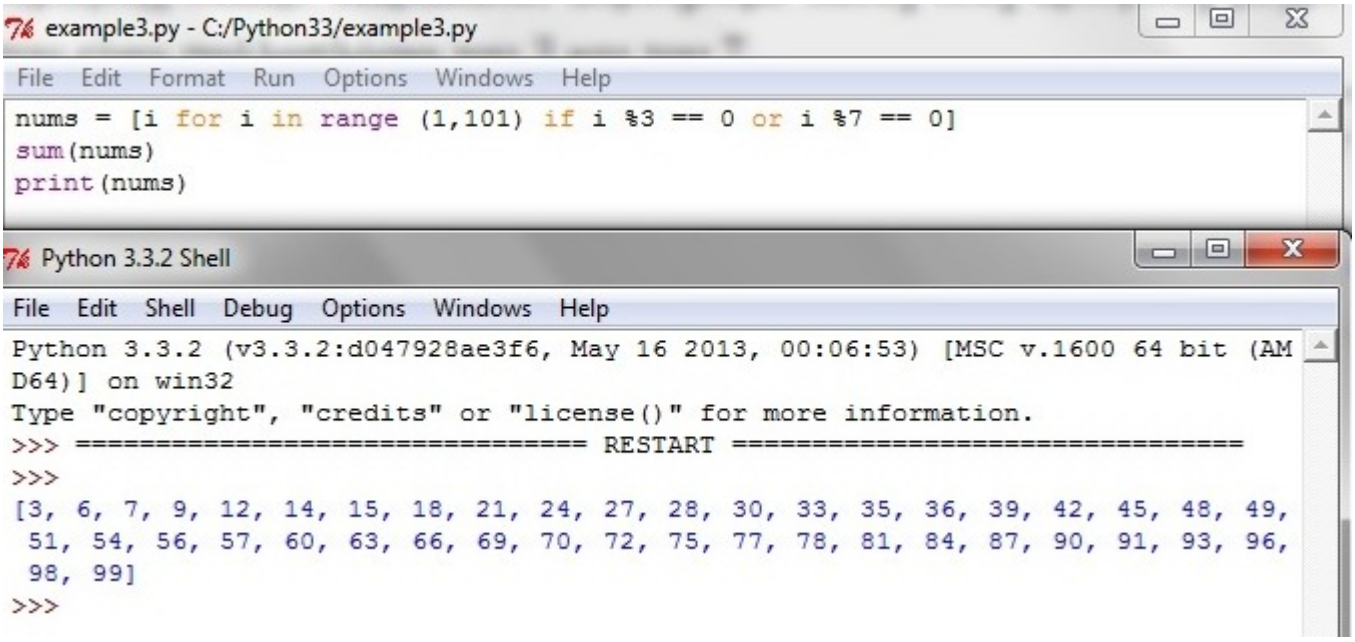


Μια list comprehension δημιουργείται μέσω ενός ειδικού συντακτικού με αγκύλες. Πάντα ως αποτέλεσμά του είναι ένα καινούργιο αντικείμενο λίστας. Το γενικό μοντέλο είναι :

```
result = [transform iteration filter ]
```

Το κομμάτι του μετασχηματισμού (transform) γίνεται κάθε φορά που προκύπτει από την διάτρεξη (iteration→ είναι η διαδικασία μέσω της οποίας παιρνουμε ένα ένα τα στοιχεία μίας ακολουθίας) και εφόσον ισχύει η συνθήκη φιλτραρίσματος (filter) , αν αυτή υπάρχει . Η σειρά με την οποία γίνεται η διάτρεξη είναι συγκεκριμένη και καθορίζει την σειρά εμφάνισης των αποτελεσμάτων στη λίστα.

Στην συνέχεια παραθέτω ένα παράδειγμα πλήρους χρήσης αυτού του μοντέλου. Με την χρήση λιστών, πολλές φορές μπορούμε να αποφύγουμε την συγγραφή βρόγχων for και να δημιουργήσουμε πιο αποδοτικό και ευανάγνωστο κώδικα. Αυτό μπορεί να γίνει αξιοποιώντας τις ειδικές λειτουργίες που μας προσφέρουν οι λίστες. Μπορούμε για παράδειγμα να αθροίσουμε όλους τους αριθμούς που περιέχονται σε μια λίστα μέσω της συνάρτησης sum(). Παρακάτω αθροίζουμε όλους τους αριθμούς μέχρι το 100 που είναι πολλαπλάσια του 3 και του 7.



Στοίβα

Με την χρήση μίας λίστας, είναι πολύ εύκολο να κατασκευάσουμε άλλες δομές. Ένα παράδειγμα είναι μια στοίβα. Η στοίβα έχει δυο βασικές λειτουργίες. Αυτές είναι οι :

→ push (ονομάζεται append στην Python)
→ pop

Έπειτα παραθέτω ένα απλό παράδειγμα υλοποίησης μιας στοίβας :

stoiva.py - C:/Python33/stoiva.py

File Edit Format Run Options Windows Help

```
a = ['Python', 'Hello']
a.append('everybody')
print(a)
b = a.pop()
print(b)
```

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

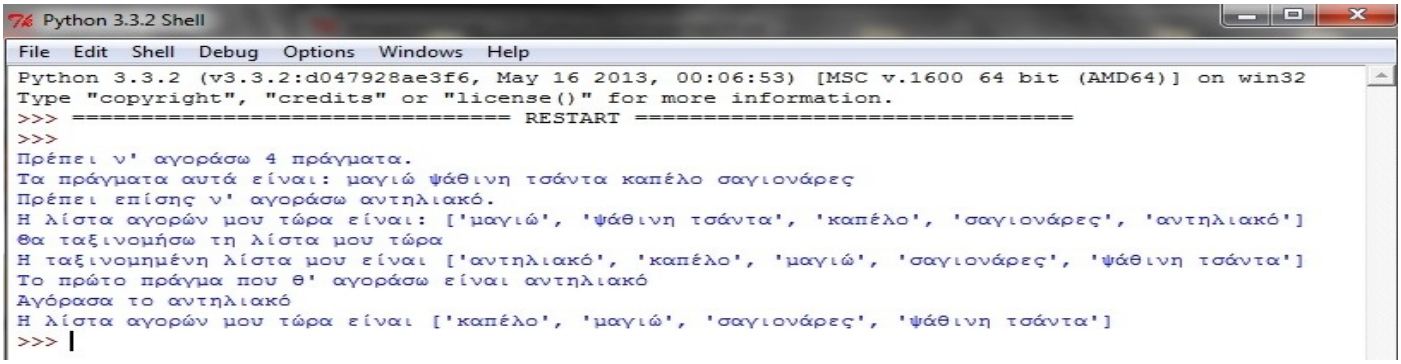
```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
['Python', 'Hello', 'everybody']
everybody
>>>
```

➔ Ακολουθεί ένα παράδειγμα για να κατανοήσουμε καλύτερα την σημασία και χρήση της λίστας :

shoppinglist.py - C:/Python33/shoppinglist.py

File Edit Format Run Options Windows Help

```
# Filename: shoppinglist.py
# Αυτή είναι η shopping list μου για το καλοκαίρι
shoppinglist = ['μαγιό', 'ψάθινη τσάντα', 'καπέλο', 'σαγιονάρες']

print('Πρέπει ν\' αγοράσω', len(shoppinglist), 'πράγματα.')

print('Τα πράγματα αυτά είναι:', end=' ')
for item in shoppinglist:
    print(item, end=' ')

print('\nΠρέπει επίσης ν\' αγοράσω αντηλιακό.')
shoppinglist.append('αντηλιακό')
print('Η λίστα αγορών μου τώρα είναι:', shoppinglist)

print('Θα ταξινομήσω τη λίστα μου τώρα')
shoppinglist.sort()
print('Η ταξινομημένη λίστα μου είναι', shoppinglist)

print('Το πρώτο πράγμα που θ\' αγοράσω είναι', shoppinglist[0])
olditem = shoppinglist[0]
del shoppinglist[0]
print('Αγόρασα το', olditem)
print('Η λίστα αγορών μου τώρα είναι', shoppinglist)
```



Πως δουλεύει το παραπάνω πρόγραμμα :

Η μεταβλητή shoppinglist είναι μια λίστα αγορών για κάποιον που πηγαίνει στην αγορά. Στη shoppinglist αποθηκεύουμε συμβολοσειρές των ονομάτων των αντικειμένων που θα αγοράσουμε, αλλά μπορείτε να προσθέσετε οποιοδήποτε είδος αντικειμένου στη λίστα συμπεριλαμβανομένων αριθμών ή ακόμα και άλλες λίστες. Επίσης έχουμε χρησιμοποιήσει το βρόχο for..in για να επανελέγξουμε τα αντικείμενα της λίστας. Μέχρι τώρα, πρέπει να έχετε καταλάβει ότι η λίστα είναι επίσης μια ακολουθία.

Παρατηρήστε πως χρησιμοποιούμε τη λέξη-κλειδί όρισμα end στη συνάρτηση print, για να δείξουμε ότι θέλουμε να τελειώσουμε (end) την έξοδο με ένα διάστημα (space) αντί της συνηθισμένης νέας γραμμής (line break).

Κατόπιν προσθέτουμε ένα στοιχείο στη λίστα χρησιμοποιώντας τη μέθοδο append του αντικειμένου λίστας. Τότε, ελέγχουμε ότι το στοιχείο έχει πραγματικά προστεθεί στη λίστα, τυπώνοντας τα περιεχόμενα της λίστας, απλά περνώντας τη λίστα στην εντολή print και την τυπώνει.

Τότε, ταξινομούμε τη λίστα χρησιμοποιώντας τη μέθοδο sort της λίστας. Είναι σημαντικό να καταλάβετε ότι αυτή η μέθοδος επηρεάζει την ίδια τη λίστα και δεν επιστρέφει μια τροποποιημένη λίστα, αυτή είναι η διαφορά από τον τρόπο που δουλεύουν οι συμβολοσειρές. Αυτό είναι που εννοούμε λέγοντας ότι ***οι λίστες είναι μεταβλητές (mutable) και οι συμβολοσειρές αμετάβλητες (immutable)***.

Κατόπιν, όταν αγοράσουμε ένα πράγμα, θέλουμε να το αφαιρέσουμε από τη λίστα. Αυτό το επιτυγχάνουμε χρησιμοποιώντας την εντολή del. Εδώ αναφέρουμε ποιο στοιχείο της λίστας θέλουμε να αφαιρέσουμε και η εντολή del το αφαιρεί από τη λίστα για μας. Καθορίζουμε ότι θέλουμε να αφαιρέσουμε το πρώτο αντικείμενο από τη λίστα και έτσι χρησιμοποιούμε την del shoplist[0] (θυμηθείτε ότι η Python αρχίζει να μετρά από το μηδέν). Εάν θέλετε να γνωρίσετε όλες τις μεθόδους που ορίζονται από τη λίστα αντικειμένου, κοιτάξτε το help(list) για λεπτομέρειες.

Πλειάδα

Οι πλειάδες χρησιμοποιούνται για να συγκρατήσουν μαζί πολλαπλά αντικείμενα. Πλειάδα είναι ένα στιγμιότυπο της λίστας. Σκεφτείτε τα σαν παρόμοια με τις λίστες, αλλά χωρίς την εκτεταμένη λειτουργικότητα που η κλάση της λίστας σας δίνει. Ένα κύριο χαρακτηριστικό των πλειάδων είναι ότι είναι **αμετάβλητες** όπως οι συμβολοσειρές, δηλαδή δε μπορείτε να τροποποιήσετε πλειάδες. Οι πλειάδες δεν αλλάζουν μέγεθος ούτε και στοιχεία.

Οι πλειάδες ορίζονται καθορίζοντας στοιχεία που διαχωρίζονται με κόμματα, μέσα σε ένα προαιρετικό ζευγάρι παρενθέσεων.

Οι πλειάδες χρησιμοποιούνται, συνήθως, στις περιπτώσεις όπου μια εντολή ή μια συνάρτηση οριζόμενη από το χρήστη, μπορεί με ασφάλεια να θεωρήσει ότι η συλλογή των τιμών δηλ. η πλειάδα των τιμών που χρησιμοποιούνται δε θα αλλάξει.

➔ Ακολουθεί παράδειγμα :

pleiades.py - C:/Python33/pleiades.py

File Edit Format Run Options Windows Help

```
# Filename: pleiades.py
# αλλαγή του προγράμματος σπουδών

lesson = ('Αρχιτεκτονική Η/Υ', 'Λειτουργικά Συστήματα', 'Παράλληλα και Κατανεμημένα Συστήματα') # οι παρενθέσεις είναι προαιρετικές
print('Ο αριθμός των μαθημάτων του κ. Δασυγένη είναι ', len(lesson))
new_lesson = ('Ψηφιακή σχεδίαση', 'Γραφικά Υπολογιστών', lesson)
print('Ο αριθμός των μαθημάτων που μπορει να αναλάβει με το νέο πρόγραμμα σπουδών είναι',len(new_lesson))
print('Όλα τα μαθήματα του κ. Δασυγένη είναι', new_lesson)
print('Όλα τα μαθήματα που είχε από παλιά ο κ. Δασυγένης είναι', new_lesson[2])
print('Το τελευταίο μάθημα από παλιά του κ. Δασυγένη είναι',new_lesson[2][2])
print('Ο αριθμός των μαθημάτων που έχει συνολικά ο κ. Δασυγένης είναι', len(new_lesson)-1+len(new_lesson[2]))
```

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Ο αριθμός των μαθημάτων του κ. Δασυγένη είναι  3
Ο αριθμός των μαθημάτων που μπορει να αναλάβει με το νέο πρόγραμμα σπουδών είναι 3
Όλα τα μαθήματα του κ. Δασυγένη είναι ('Ψηφιακή σχεδίαση', 'Γραφικά Υπολογιστών', ('Αρχιτεκτονική Η/Υ', 'Λειτουργικά Συστήματα', 'Παράλ
ληλα και Κατανεμημένα Συστήματα'))
Όλα τα μαθήματα που είχε από παλιά ο κ. Δασυγένης είναι ('Αρχιτεκτονική Η/Υ', 'Λειτουργικά Συστήματα', 'Παράλληλα και Κατανεμημένα Συστ
ήματα')
Το τελευταίο μάθημα από παλιά του κ. Δασυγένη είναι Παράλληλα και Κατανεμημένα Συστήματα
Ο αριθμός των μαθημάτων που έχει συνολικά ο κ. Δασυγένης είναι 5
>>>
```

Πώς δουλεύει το παραπάνω πρόγραμμα:

Η μεταβλητή lesson αναφέρεται σε μια πλειάδα στοιχείων. Βλέπουμε ότι η συνάρτηση len μπορεί να χρησιμοποιηθεί για να πάρει το μήκος της πλειάδας. Αυτό επίσης δείχνει ότι η πλειάδα είναι επίσης και μια ακολουθία.

Με την αλλαγή του προγράμματος σπουδών γίνεται ανάθεση επιπλέον μαθημάτων στον κ. Δασυγένη. Η πλειάδα new_lesson περιέχει κάποια μαθήματα που βρίσκονται ήδη στον οδηγό σπουδών, μαζί με τα μαθήματα που είχε ο κ. Δασυγένης και στον παλιό οδηγός σπουδών. Πολύ σημαντικό είναι ότι μια πλειάδα μέσα σε μια πλειάδα δε χάνει την ταυτότητά της.

Μπορούμε να έχουμε πρόσβαση στα αντικείμενα μέσα στην πλειάδα, καθορίζοντας τη θέση του στοιχείου μέσα σε ένα ζευγάρι αγκύλες, ακριβώς όπως κάναμε για τις λίστες. Αυτό ονομάζεται **τελεστής ευρετηρίασης (indexing operator)**. Παίρνουμε το τρίτο στοιχείο μέσα στο new_lesson καθορίζοντας new_lesson[2] και παίρνουμε το τρίτο στοιχείο μέσα στο τρίτο στοιχείο στην πλειάδα new_lesson καθορίζοντας new_lesson[2][2].

Χρησιμοποιώντας μια πλειάδα ουσιαστικά δημιουργούμε ένα αντικείμν το οποίο είναι μια ομάδα άλλων αντικειμένων. Αν για μια λειτουργία μπορούμε να χρησιμοποιήσουμε είτε πλειάδες είτε λίστες, προτιμούμε τις πλειάδες καθώς είναι ο πιο γρήγορος τρόπος!

Επιπρόσθετες παρατηρήσεις :

Παρενθέσεις → Αν και οι παρενθέσεις είναι προαιρετικές, καλό θα ήταν να τις έχουμε πάντα για να κάνουμε φανερό ότι αυτή είναι μια πλειάδα, ειδικά επειδή αποφεύγεται η ασάφεια. Για παράδειγμα, το print(1,2,3) και το print((1,2,3)) σημαίνουν δυο διαφορετικά πράγματα -το μεν τυπώνει τρεις αριθμούς, αντίθετα το δε τυπώνει μια πλειάδα (η οποία περιέχει τρεις αριθμούς).

Πλειάδα με 0 ή 1 στοιχεία →Μια άδεια πλειάδα δομείται από ένα άδειο ζευγάρι παρενθέσεων όπως το myempty = (). Πάντως, μια πλειάδα με ένα μόνο στοιχείο δεν είναι τόσο απλή. Πρέπει να την καθορίσετε χρησιμοποιώντας ένα κόμμα μετά το πρώτο και μοναδικό στοιχείο, έτσι ώστε η Python να μπορεί να διαφοροποιεί μια πλειάδα από ένα ζευγάρι παρενθέσεων που παρεμβάλλουν το αντικείμενο σε μια έκφραση, δηλαδή πρέπει να καθορίζετε singleton = (2 ,), εάν εννοείτε ότι θέλετε μια πλειάδα που περιέχει το στοχείο 2.

Για τα λεξικά και την χρήση τους κάναμε λόγο παραπάνω (παραπομπή στην σελίδα 69)

Ακολουθίες

Οι λίστες, οι πλειάδες και οι συμβολοσειρές είναι παραδείγματα ακολουθιών, αλλά τι είναι οι ακολουθίες και τι το ιδιαίτερο με αυτές; Τα κυρίαρχα χαρακτηριστικά είναι ότι έχουν **δοκιμές ένταξης (membership tests**, δηλ. τις εκφράσεις in και not in) και λειτουργίες ευρετηρίασης (indexing operations). Η λειτουργία ευρετηρίασης μας επιτρέπει να πάρουμε απ' ευθείας ένα συγκεκριμένο στοιχείο στην ακολουθία.

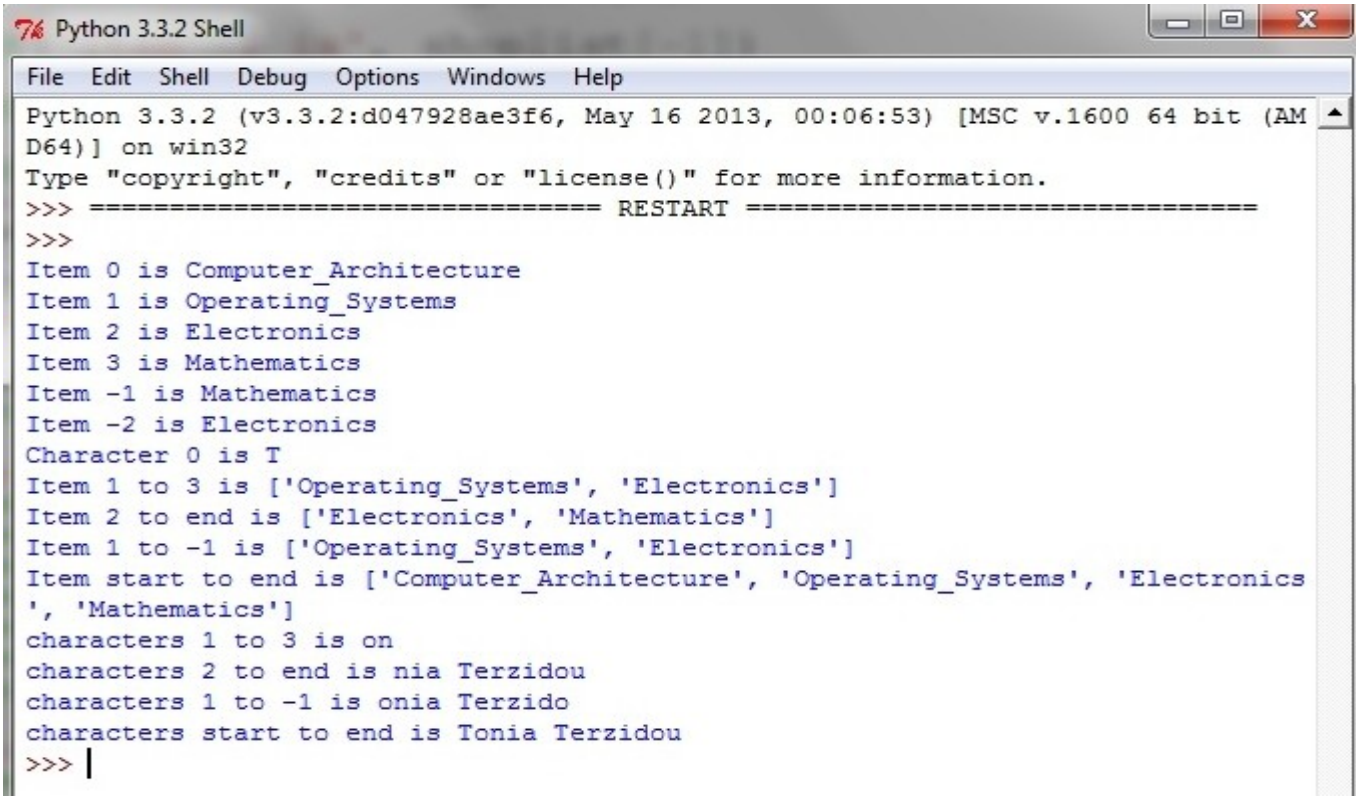
Οι τρεις τύποι ακολουθιών που αναφέρθηκαν παραπάνω, λίστες, πλειάδες και συμβολοσειρές έχουν επίσης μια λειτουργία τεμαχισμού (slicing operation), που μας επιτρέπει να ανακτούμε ένα μέρος (slice) της ακολουθίας.

➔ Ακολουθεί παράδειγμα :

sequenceexample.py - C:/Python33/sequenceexample.py

File Edit Format Run Options Windows Help

```
# Filename: sequenceexample.py

ictelessons_list = ['Computer_Architecture', 'Operating_Systems', 'Electronics', 'Mathematics']
name = 'Tonia Terzidou'

# Indexing or 'Subscription' operation
print('Item 0 is', ictelessons_list[0])
print('Item 1 is', ictelessons_list[1])
print('Item 2 is', ictelessons_list[2])
print('Item 3 is', ictelessons_list[3])
print('Item -1 is', ictelessons_list[-1])
print('Item -2 is', ictelessons_list[-2])
print('Character 0 is', name[0])

# Slicing on a list
print('Item 1 to 3 is', ictelessons_list[1:3])
print('Item 2 to end is', ictelessons_list[2:])
print('Item 1 to -1 is', ictelessons_list[1:-1])
print('Item start to end is', ictelessons_list[:])

# Slicing on a string
print('characters 1 to 3 is', name[1:3])
print('characters 2 to end is', name[2:])
print('characters 1 to -1 is', name[1:-1])
print('characters start to end is', name[:])
```



Πως λειτουργεί το παραπάνω πρόγραμμα :

Αρχικά βλέπουμε πώς να χρησιμοποιούμε ευρετήρια για να παίρνουμε μοναδικά στοιχεία της ακολουθίας. Αυτό αναφέρεται επίσης σαν συνδρομητική λειτουργία (subscription operation). Οποτεδήποτε καθορίζετε ένα νούμερο σε μια ακολουθία μέσα σε αγκύλες, όπως φαίνεται παραπάνω, η Python θα φέρνει το στοιχείο που αντιστοιχεί σε αυτή τη θέση στην ακολουθία. Θυμηθείτε ότι η Python αρχίζει να μετράει τα νούμερα από το μηδέν. Για αυτό το λόγο η ictelessons_list[0] κάνει μετάκληση του πρώτου στοιχείου και η ictelessons_list[3] κάνει μετάκληση του τέταρτου στοιχείου στην ακολουθία shoplist.

Το ευρετήριο μπορεί να είναι ένας αρνητικός αριθμός, στην οποία περίπτωση, η θέση υπολογίζεται από το τέλος της ακολουθίας. Έτσι, η ictelessons_list[-1] αναφέρεται στο τελευταίο στοιχείο στην ακολουθία και η ictelessons_list[-2] κάνει μετάκληση του δεύτερου από το τέλος στοιχείου στην ακολουθία.

Η λειτουργία τεμαχισμού χρησιμοποιείται καθορίζοντας την ονομασία της ακολουθίας, ακολουθούμενο από ένα προαιρετικό ζευγάρι αριθμών, που διαχωρίζονται από διπλή τελεία μέσα σε αγκύλες. Σημειώστε ότι αυτό είναι παρόμοιο με τη λειτουργία ευρετηρίασης, που έχει χρησιμοποιηθεί μέχρι τώρα. Θυμηθείτε ότι ***τα νούμερα είναι προαιρετικά αλλά η διπλή τελεία δεν είναι.***

Το πρώτο νούμερο (πριν από τη διπλή τελεία) στη λειτουργία τεμαχισμού αναφέρεται στη θέση από όπου το τμήμα (slice) αρχίζει και το δεύτερο νούμερο (μετά τη διπλή τελεία) δείχνει που θα σταματήσει το τμήμα. Εάν δεν καθορίζεται το

πρώτο νούμερο, η Python θα αρχίσει στην αρχή της ακολουθίας. Εάν το δεύτερο νούμερο παραλήφθηκε, η Python θα σταματήσει στο τέλος της ακολουθίας.

Το τμήμα που επιστρέφεται ξεκινά από την αρχική θέση και τελειώνει ακριβώς πριν από την τελική θέση, δηλαδή η αρχική θέση συμπεριλαμβάνεται αλλά η τελική θέση αποκλείεται από το τμήμα της ακολουθίας.

Έτσι η ictelessons_list[1:3] επιστρέφει ένα τμήμα της ακολουθίας αρχίζοντας στη θέση 1, περιλαμβάνει τη θέση 2, αλλά σταματάει στη θέση 3, συνεπώς, επιστρέφεται ένα τμήμα δύο αντικειμένων. Παρόμοια, η ictelessons_list[:] επιστρέφει ένα αντίγραφο όλης της ακολουθίας.

Μπορείτε επίσης να κάνετε τεμαχισμό με αρνητικές θέσεις. Οι αρνητικοί αριθμοί χρησιμοποιούνται για θέσεις από το τέλος της ακολουθίας. Για παράδειγμα, η ictelessons_list[:-1] θα επιστρέψει ένα τμήμα της ακολουθίας η οποία παραλείπει το τελευταίο στοιχείο της ακολουθίας, αλλά περιέχει κάθε άλλο.

Το σπουδαίο με τις ακολουθίες είναι ότι μπορούμε να έχουμε πρόσβαση σε πλειάδες, λίστες και συμβολοσειρές όλες με τον ίδιο τρόπο.

Σύνολο (Set)

Τα σύνολα είναι μη ταξινομημένες συλλογές απλών αντικεινένων. Αυτά χρησιμοποιούνται όταν η ύπαρξη ενός αντικειμένου σε μια συλλογή είναι πιο σπουδαία από την εντολή ή πόσες φορές αυτή συμβαίνει.

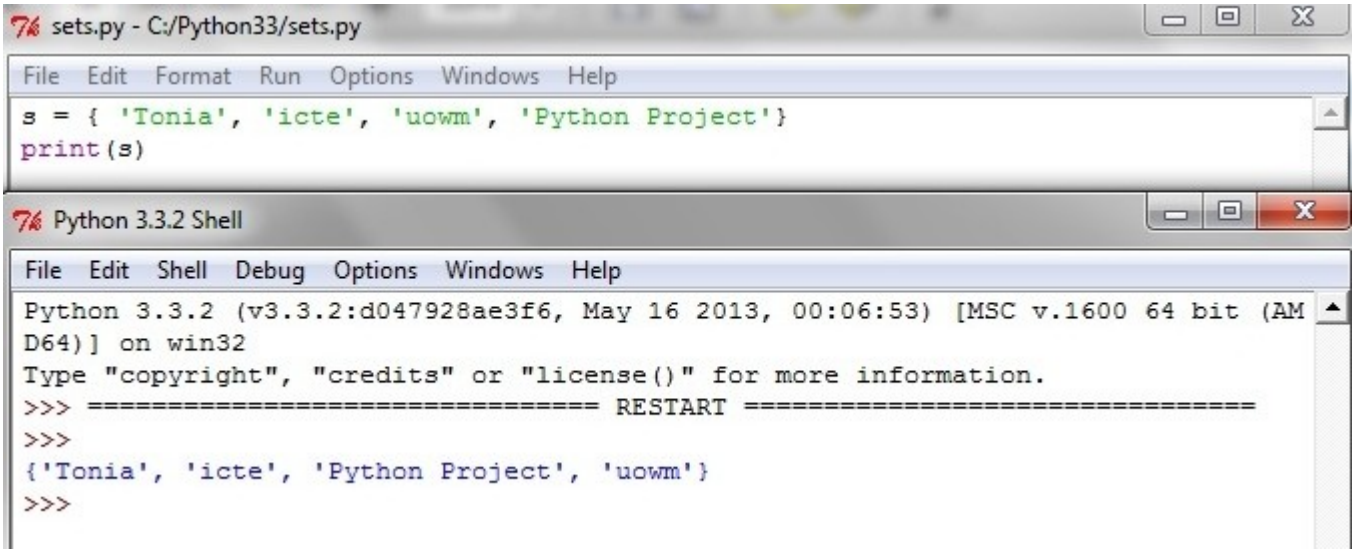
Χρησιμοποιώντας τα σύνολα, μπορείτε να ελέγξετε για ένταξη (membership), εάν είναι ένα υποσύνολο (subset) ενός άλλου συνόλου, να βρείτε την τομή (intersection) ανάμεσα σε δύο σύνολα και ούτω καθεξής.

Τα σύνολα μας διευκολύνουν στην ομαδοποίηση πολλών αντικειμένων και στην εφαρμογή στην συνέχεια πράξεων όπως η ενωσή τους με αποδοτικό τρόπο, εξασφαλίζοντας πως κάθε στοιχείο , αν περιέχεται σε πάνω από ένα σύνολα, τελικά θα βρεθεί μόνο μια φορά στο τελικό αποτέλεσμα. Πρόκειται για μια δομή δεδομένων που αναπαρίσταται ψς ένα μη διατεταγμένο σύνολο μοναδικών στοιχείων.

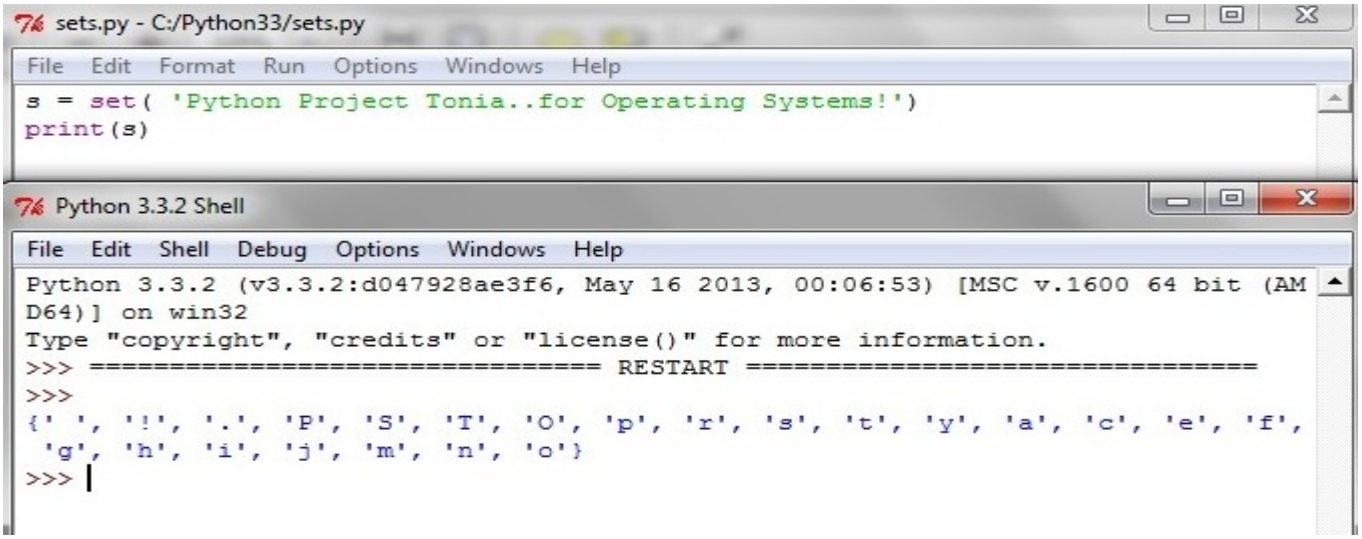
Δημιουργία συνόλου

Για την απευθείας δημιουργία ενός συνόλου χρησιμοποιούμε αγκύλες. Η Python καταλαβαίνει πως δεν είναι λεξικό αλλά αναφερόμαστε σε σύνολο, επειδή δεν

υπάρχει η ανω κάτω τελεία η οποία υποδηλώνει την τιμή που αντιστοιχίζεται σε κάθε κλειδί. Ας δούμε ένα παράδειγμα :



Μπορούμε επίσης να δημιουργήσουμε ένα λεξικό από οποιοδήποτε τύπο δεδομένων φθάνει να μπορούμε να διατρέξουμε τα στοιχεία αυτού του τύπου (iterable).



Βασικές πράξεις συνόλων

Όπως γνωρίζουμε από τα μαθηματικά, οι βασικές πράξεις συνόλων είναι :

- ✓ Πρόσθεση στοιχείου σε ένα σύνολο
- ✓ Αφαίρεση στοιχείου από ένα σύνολο
- ✓ Ένωση συνόλων
- ✓ Τομή συνόλων
- ✓ Διαφορά συνόλων
- ✓ Συμμετρική διαφορά συνόλων

➔ Ακολουθούν μερικά παραδείγματα για την καλύτερη κατανόηση των συνόλων:

1ο παράδειγμα :

examplewithsets.py - C:/Python33/examplewithsets.py

File Edit Format Run Options Windows Help

```
a = set('ToniaTerzuowm')
b = set('PythonTericte')

print('A = ', a)
print('B = ',b)
a.add('c')       # προσθέτω στοιχειο
b.remove('c')    # αφαιρώ στοιχειο
print('A = ',a)
print('B = ',b)
print('A - B = ', a - b)  # διαφορά συνόλων
print('A | B = ', a | b)  # ένωση συνόλων
print('A & B = ', a & b)  # τομή συνόλων
print('A ^ B = ', a ^ b)  # συμμετρική διαφορά συνόλων
```

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
A =  {'e', 'a', 'm', 'o', 'n', 'i', 'u', 'T', 'w', 'r', 'z'}
B =  {'e', 'c', 'T', 'o', 'n', 'i', 'h', 't', 'P', 'r', 'y'}
A =  {'e', 'a', 'c', 'm', 'o', 'n', 'i', 'u', 'T', 'w', 'r', 'z'}
B =  {'e', 'T', 'o', 'n', 'i', 'h', 't', 'P', 'r', 'y'}
A - B =  {'a', 'c', 'm', 'u', 'w', 'z'}
A | B =  {'e', 'a', 'c', 'm', 'o', 'n', 'i', 'h', 'u', 't', 'w', 'r', 'y', 'z',
'T', 'P'}
A & B =  {'e', 'o', 'n', 'i', 'T', 'r'}
A ^ B =  {'a', 'c', 't', 'm', 'h', 'u', 'w', 'P', 'y', 'z'}
>>>
```

2ο παράδειγμα :

2ndexamplewithsets.py - C:/Python33/2ndexamplewithsets.py

File Edit Format Run Options Windows Help

```
a = ['mathematics1', 'mathematics2', 'discrete mathematics', 'applied mathematics1']
set1 = set(a)
print(set1)
print('mathematics1' in set1)
set2 = set1.copy()
set2.add('systems and signals')
print(set1 < set1)
set1.remove('discrete mathematics')
print(set1 & set2)
print(set1 | set2)
```

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

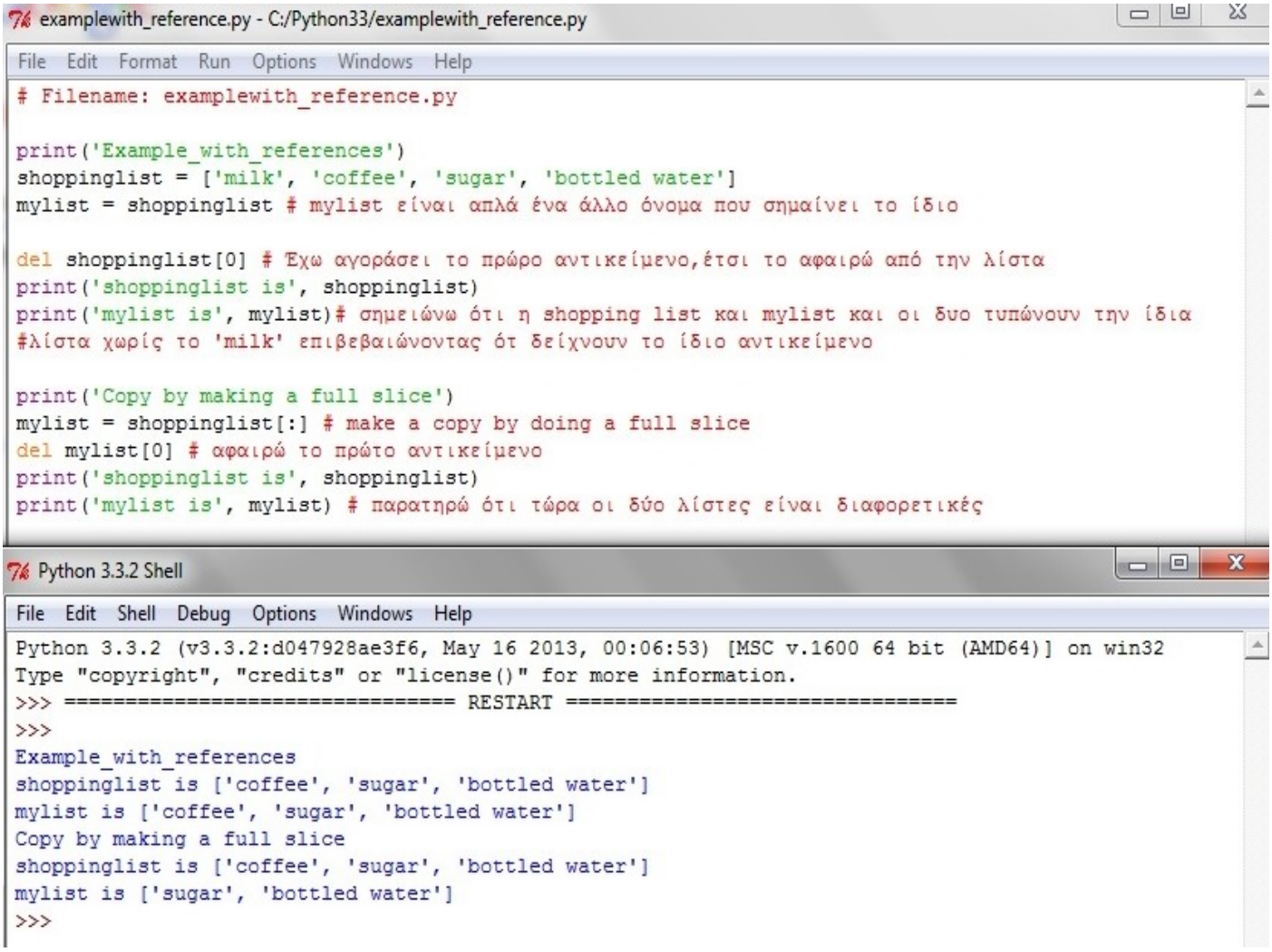
```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
{'discrete mathematics', 'applied mathematics1', 'mathematics2', 'mathematics1'}
True
False
{'applied mathematics1', 'mathematics2', 'mathematics1'}
{'mathematics2', 'mathematics1', 'systems and signals', 'discrete mathematics', 'applied mathematics1'
}
>>>
```

Παραπομπές (References)

Όταν δημιουργείτε ένα αντικείμενο και το εκχωρείτε σε μια μεταβλητή, η μεταβλητή απλά παραπέμπει στο αντικείμενο και δεν αντιπροσωπεύει καθ' αυτό το αντικείμενο. Η ονομασία της μεταβλητής δείχνει σε εκείνο το σημείο της μνήμης του υπολογιστή, όπου αποθηκεύεται το αντικείμενο. Αυτό ονομάζεται *συσχέτιση (binding) της ονομασίας με το αντικείμενο*.

Γενικά δεν πρέπει να ανησυχείτε για αυτό, αλλά υπάρχει μια μικρή επίδραση εξαιτίας των παραπομπών την οποία πρέπει να αντιληφθείτε.

➔ Ακολουθεί σχετικό παράδειγμα :



Τα αναλυτικά σχόλια του παραπάνω παραδείγματος εξηγούν τι ακριβώς κάνει!

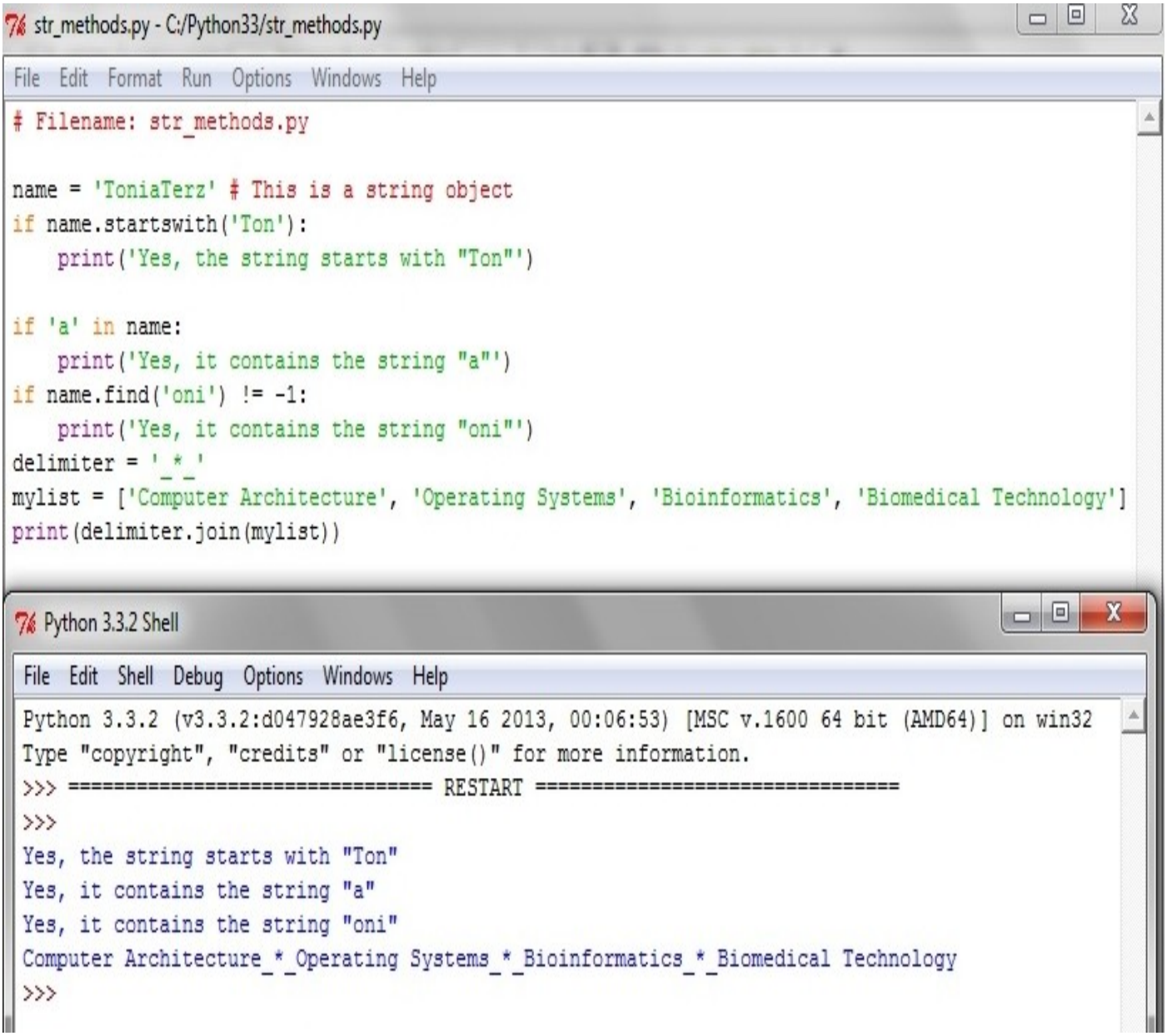
Εάν θέλουμε να φτιάξουμε ένα αντίγραφο μιας λίστας, ή τέτοιου είδους ακολουθίες, ή σύμπλοκα αντικείμενα (όχι απλά αντικείμενα όπως ακέραιους αριθμούς), τότε πρέπει να χρησιμοποιήσουμε τη λειτουργία τεμαχισμού για να φτιάξουμε αντίγραφο. Εάν εκχωρήσουμε την ονομασία της μεταβλητής σε μια άλλη ονομασία, τότε και οι δυο τους θα παραπέμπουν στο ίδιο αντικείμενο και αυτό θα μπορούσε να είναι πρόβλημα για μας, εάν δεν είμαστε προσεκτικοί.

Μια εντολή εκχώρησης για λίστες δε δημιουργεί αντίγραφο. Πρέπει να χρησιμοποιήσετε τη λειτουργία τεμαχισμού για να φτιάξετε αντίγραφο της ακολουθίας.

Κάτι παραπάνω για τις συμβολοσειρές…

Οι συμβολοσειρές είναι επίσης αντικείμενα και έχουν μεθόδους που κάνουν τα πάντα, από τον έλεγχο του τμήματος μιας συμβολοσειράς μέχρι αποκοπή των διαστημάτων! Οι συμβολοσειρές που χρησιμοποιείτε στο πρόγραμμα είναι όλες αντικείμενα της κλάσης str. Μερικές χρήσιμες μεθόδοι της κλάσης παρουσιάζονται στο επόμενο παράδειγμα. Για μια ολοκληρωμένη λίστα τέτοιων μεθόδων (βλέπε την help(str)).

➔ Ακολουθεί ένα κατατοπιστικότατο παράδειγμα :



Πως λειτουργεί το παραπάνω πρόγραμμα :

Εδώ βλέπουμε πολλές μεθόδους της συμβολοσειράς σε ενέργεια. Η **μέθοδος startswith** χρησιμοποιείται για να ανακαλύψουμε αν η συμβολοσειρά αρχίζει με τη δοθείσα συμβολοσειρά. Ο **τελεστής in** χρησιμοποιείται για να ελέγξει αν η δοθείσα συμβολοσειρά είναι μέρος της συμβολοσειράς.

Η **μέθοδος find** χρησιμοποιείται για να ανακαλύψει τη θέση της δοθείσας συμβολοσειράς στη συμβολοσειρά ή επιστρέφει -1 εάν δεν επιτύχει την ανακάλυψη της υποσυμβολοσειράς (substring). Η **κλάση str** επίσης έχει την μέθοδο join για να ενώνει τα στοιχεία μιας ακολουθίας, με τη συμβολοσειρά να ενεργεί σα **διαχωριστικό (delimiter)** ανάμεσα σε κάθε στοιχείο της ακολουθίας, και επιστρέφει μια μεγαλύτερη συμβολοσειρά γεννημένη από αυτό.

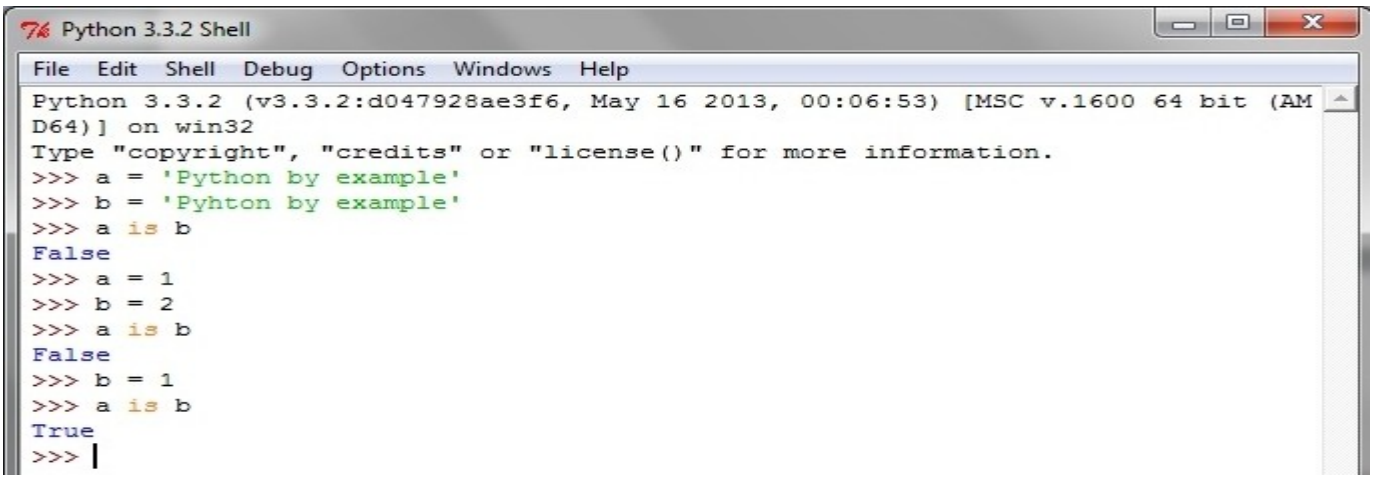
Σύνθετα αντικείμενα

Εδώ θα αναφερθούμε στα υπόλοιπα αντικείμενα στην Python που περιέχουν άλλα αντικείμενα (container types). Μερικά από αυτά είναι :

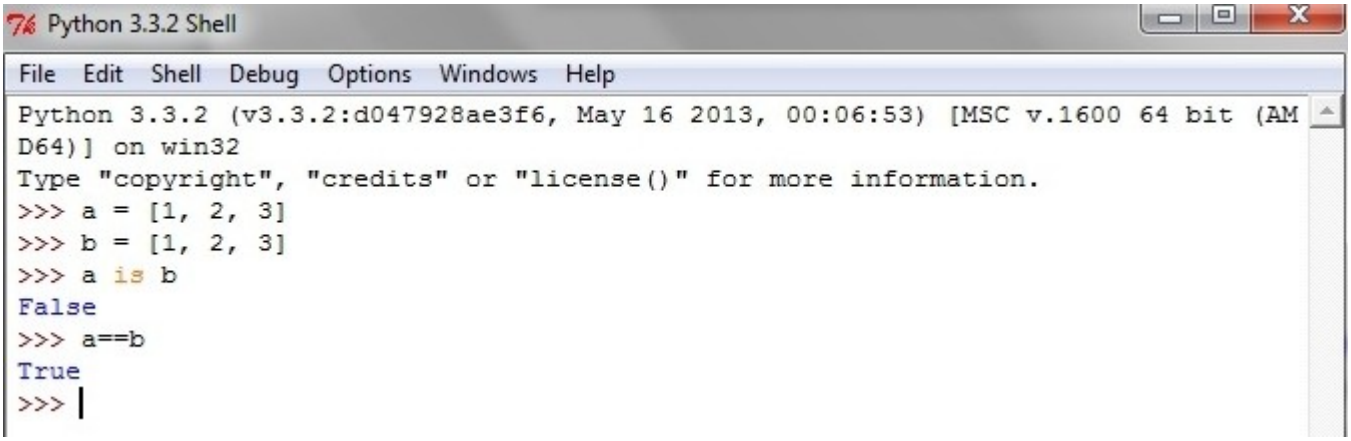
- ✓ Πλειάδες (tuple)
- ✓ Αλφαριθμητικά (string)
- ✓ Λίστες (list)
- ✓ Σύνολα (set)
- ✓ Λεξικά (dictionary)

Στην ίδια κατηγορία μπορούμε να εντάξουμε αντικείμενα από κλάσεις που φτιάχνουμε εμείς.

Αυτά καταλαμβάνουν διαφορετική θέση μνήμης ακόμα και αν περιέχουν αντικείμενα με τις ίδιες τιμές. Μόνη εξαίρεση είναι αν μια μεταβλητή αποτελεί ψευδώνυμο (alias) μιας άλλης,το οποίο θα αναλύσουμε στην συνέχεια.

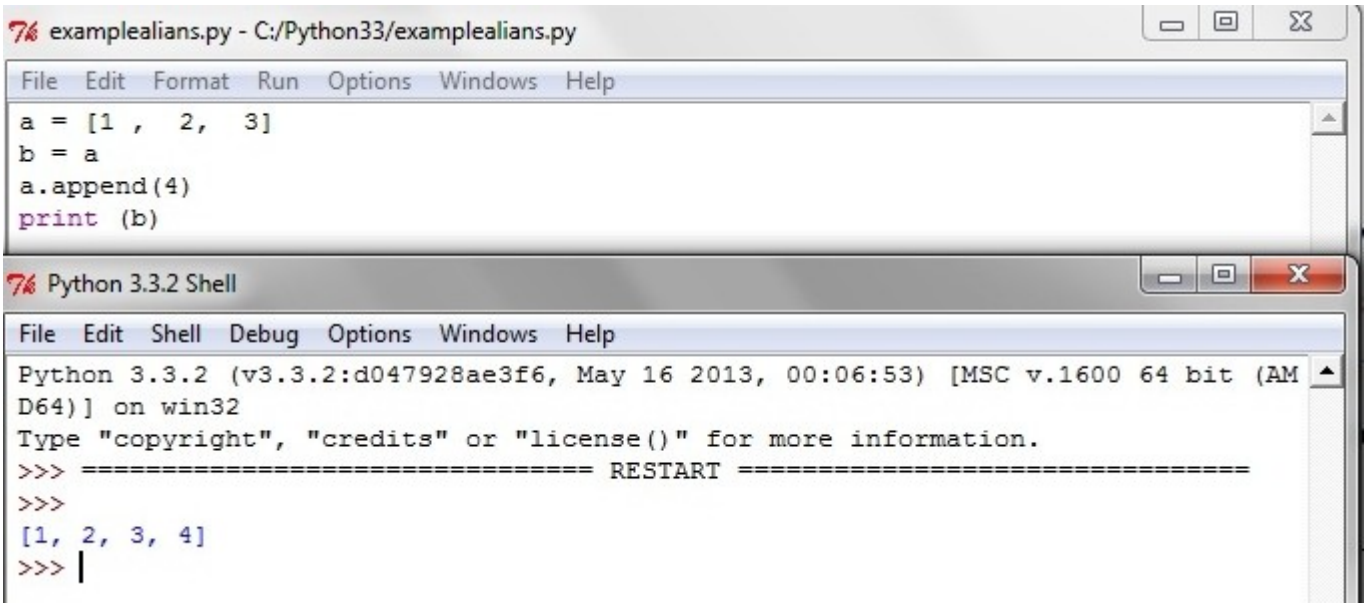


⇨ Εδώ αξίζει να επισημάνουμε και την **διαφορά του τελεστή is από τον τελεστή ==**. Ο πρώτος ελέγχει αν δυο μεταβλητές αναφέρονται σε ακριβώς το ίδιο αντικείμενο, ενώ ο τελεστής == ελέγχει μόνο τα περιεχόμενα των δυο αντικειμένων. Βλέπουμε στο παρακάτω παράδειγμα :



Ψευδώνυμα

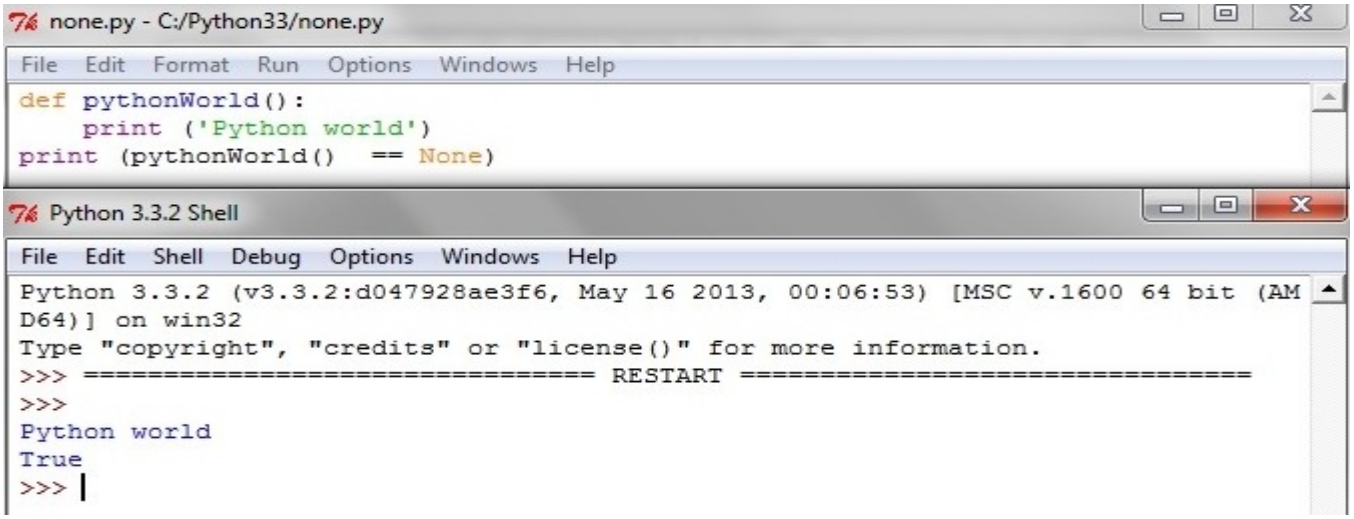
Ψευδώνυμο (alias) είναι όταν αναφερόμαστε σε ένα αντικείμενο με ένα διαφορετικό όνομα, χρησιμοποιώντας μια διαφορετική μεταβλητή για να αναφερθούμε σε αυτό. ΄Οταν γίνεται κάτι τέτοιο, ό,τι αλλαγές γίνουν, ανεξάρτητα από τον τρόπο αναφοράς στο συγκεκριμένο αντικείμενο, θα επηρεάσουν όλες τις μεταβλητές που αναφέρονται σε αυτό.



Παρατηρούμε δηλαδή ότι η προσθήκη του αριθμού 4 στην λίστα που έδειχνε το a, επηρεάζει και την λίστα b, αφού στην ουσία το b είναι ένα ψευδώνυμο του a και αναφέρονται ακριβώς στο ίδιο αντικείμενο.

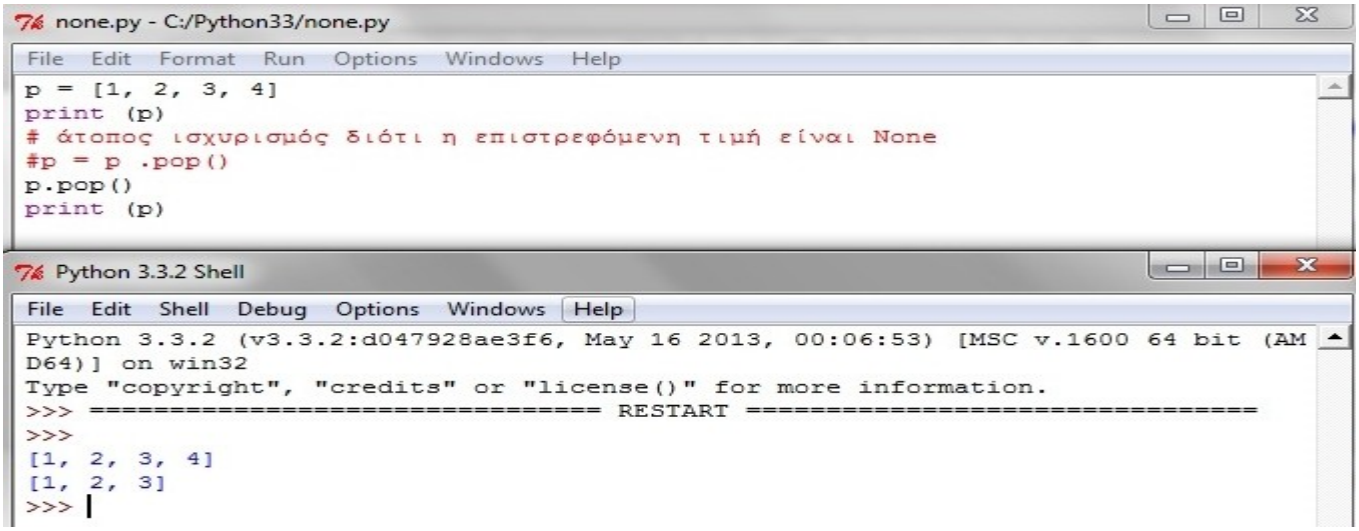
None

Το None είναι ένας ειδικός τύπος ο οποίος αναπαριστά την ιδέα ότι δεν υπάρχει κάτι. Σε άλλες γλώσσες η ίδια έννοια συχνά απαντάται από το Null. Η τιμή Null στην Python δεν ισούται με καμία άλλη τιμή παρά μόνο με τον εαυτό της και ο μόνος τρόπος για να αναφερθούμε σε αυτή είναι μέσω της λέξης κλειδί None.



Η τιμή None είναι η τιμή που μας επιστρέφουν συναρτήσεις που τελικά δεν επιστρέφουν κάτι. Αφού όπως είπαμε η τιμή None αντιπροσωπεύει την έννοια του δεν υπάρχει κάτι, η συγκεκριμένη συμπεριφορά είναι κάτι που θα έπρεπε να περιμένουμε.

Σημαντική σημείωση → Η συνάρτηση θα μπορούσε να επιστρέφει την τιμή None. Τώρα όμως είμαστε στην περίπτωση όπου επειδή δεν επιστρέφεται κάτι, ρίσκουμε αυτή την τιμή. Αυτό οδηγεί σε μια σχεδιαστική αρχή όπου συνήθως, όταν οι συναρτήσεις επηρεάζουν τα ορίσματα τους, επιστρέφουν την τιμή None έτσι ώστε αφού ο χρήστης να λέπει ητά πως τα αποτελέσματα της συνάρτησης αποθηκεύονται μέσα στο αντικείμενο.



Χώρος Ονομάτων

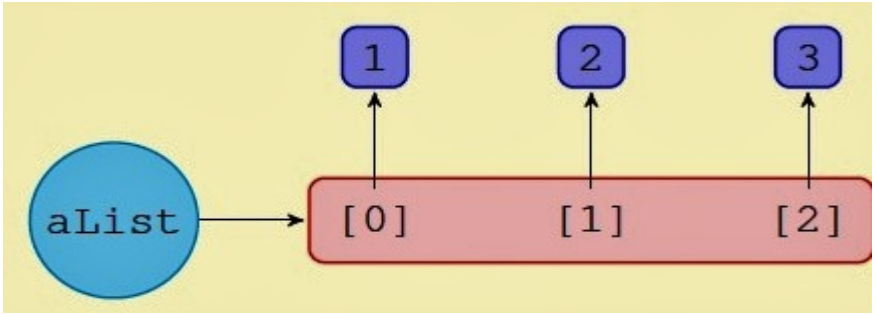
Ο χώρος ονομάτων ουσιαστικά αντιστοιχεί σε ένα ‘περιβάλλον’ για τα αντικείμενα που περιέχει και μας επιτρέπει να αντιληφθούμε (και εμείς και ο υπολογιστής) που αναφέρεται μια μεταβλητή.Στην Python, για να απλοποιήσουμε τα πράγματα, μπορούμε να σκεφθούμε ότι παρέχεται ένας χώρος ονομάτων για κάθε αρχείο. ΄Ετσι, όταν κάνουμε:

`from file import*` εισάγουμε όλες τις μεταβλητές από το αρχείο python.py στον χώρο ονομάτων. Απλοποιώντας κάποια πράγματα, μπορούμε να σκεφθούμε ότι κάθε φορά που κάνουμε στοίχηση σε κώδικα προς τα μέσα, δημιουργείται ένας καινούργιος χώρος ονομάτων. Σε αυτό το χώρο ονομάτων ανήκουν όλες οι μεταβλητές που δηλώνονται για πρώτη φορά εκεί. Αν κάποια μεταβλητή δε ρεθεί εκεί, η αναζήτηση συνεχίζεται στο χώρο ονομάτων που υπάρχει ένα επίπεδο πριν (μικρότερη στοίχηση προς τα μέσα) μέχρι να φθάσουμε στο τέρμα αριστερά επίπεδο.

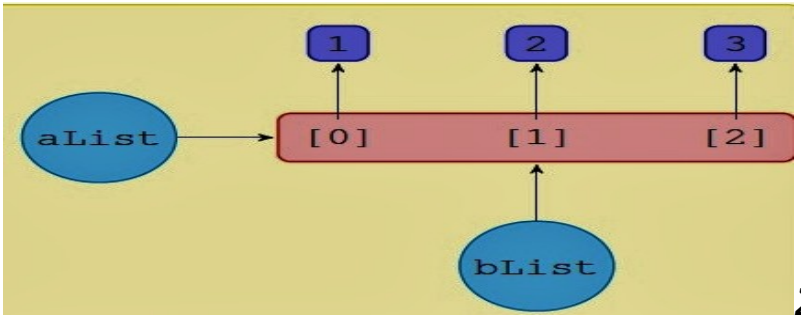
Εμβέλεια

΄Αλλη μια σημαντική παράμετρος ως προς τις μεταβλητές που πάντα πρέπει να έχουμε υπόψη μας είναι η εμβέλεια τους.

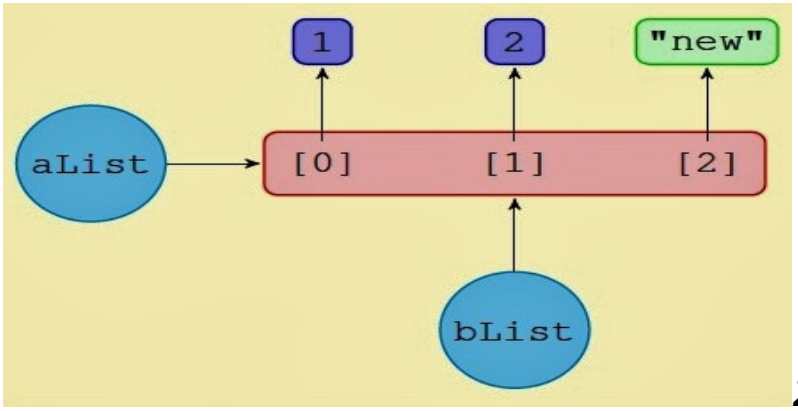
```
def  f(a):

     a = 5
```



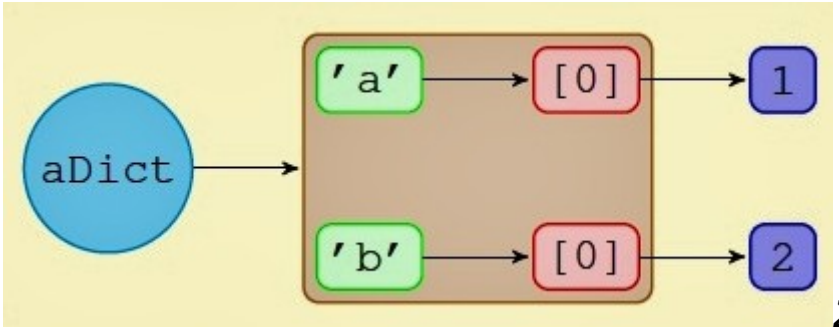
Σχήμα 1 : Ο χώρος ονομάτων (namespace) μετά τη δημιουργία μίας λίστας.



Σχήμα 2 : Ο χώρος ονομάτων μετά τη δημιουργία και μίας δεύτερης λίστας που δείχνει στην πρώτη.

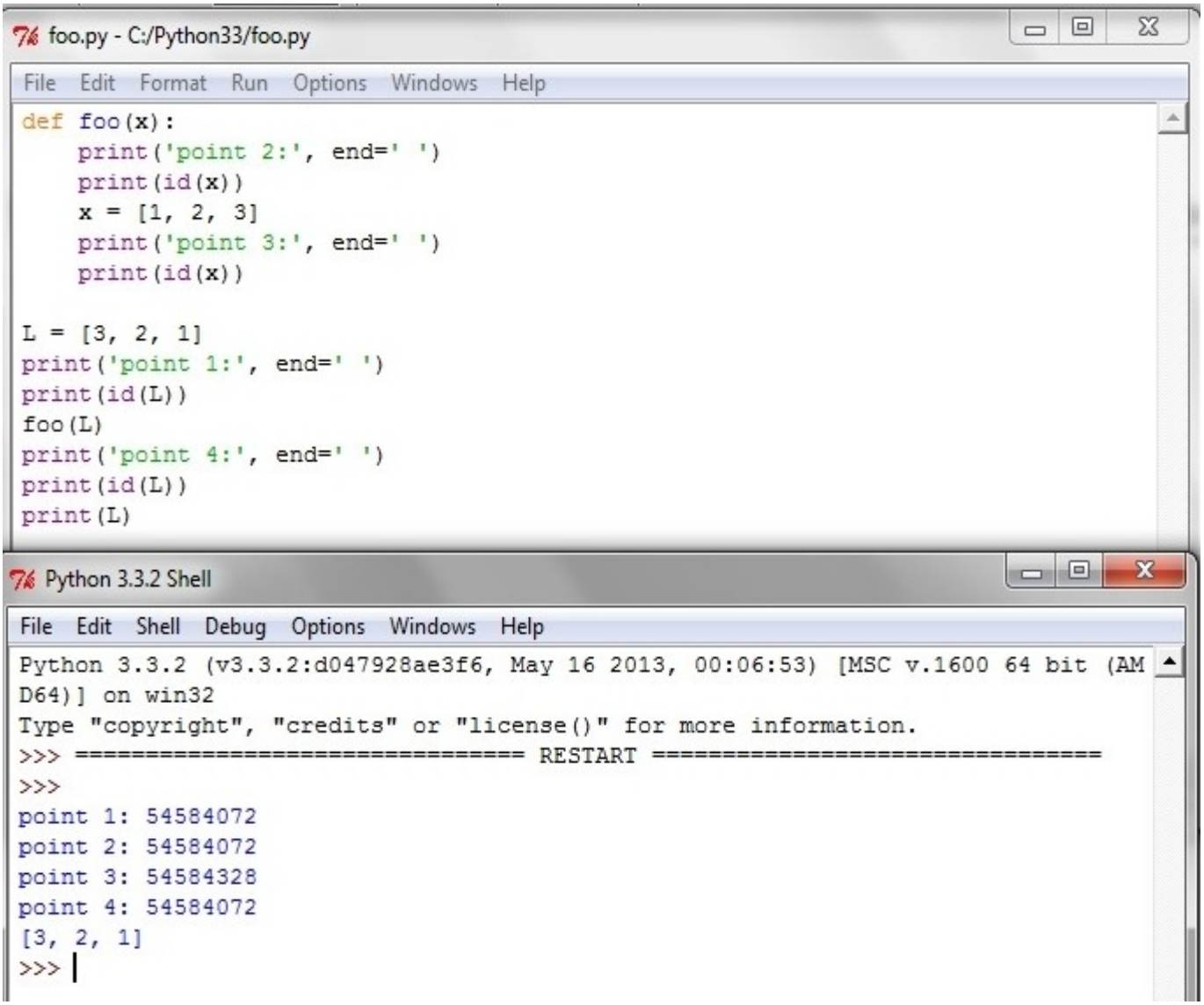


Σχήμα 3 : Η αλλαγή των στοιχείων αλλάζει και τις δύο λίστες.



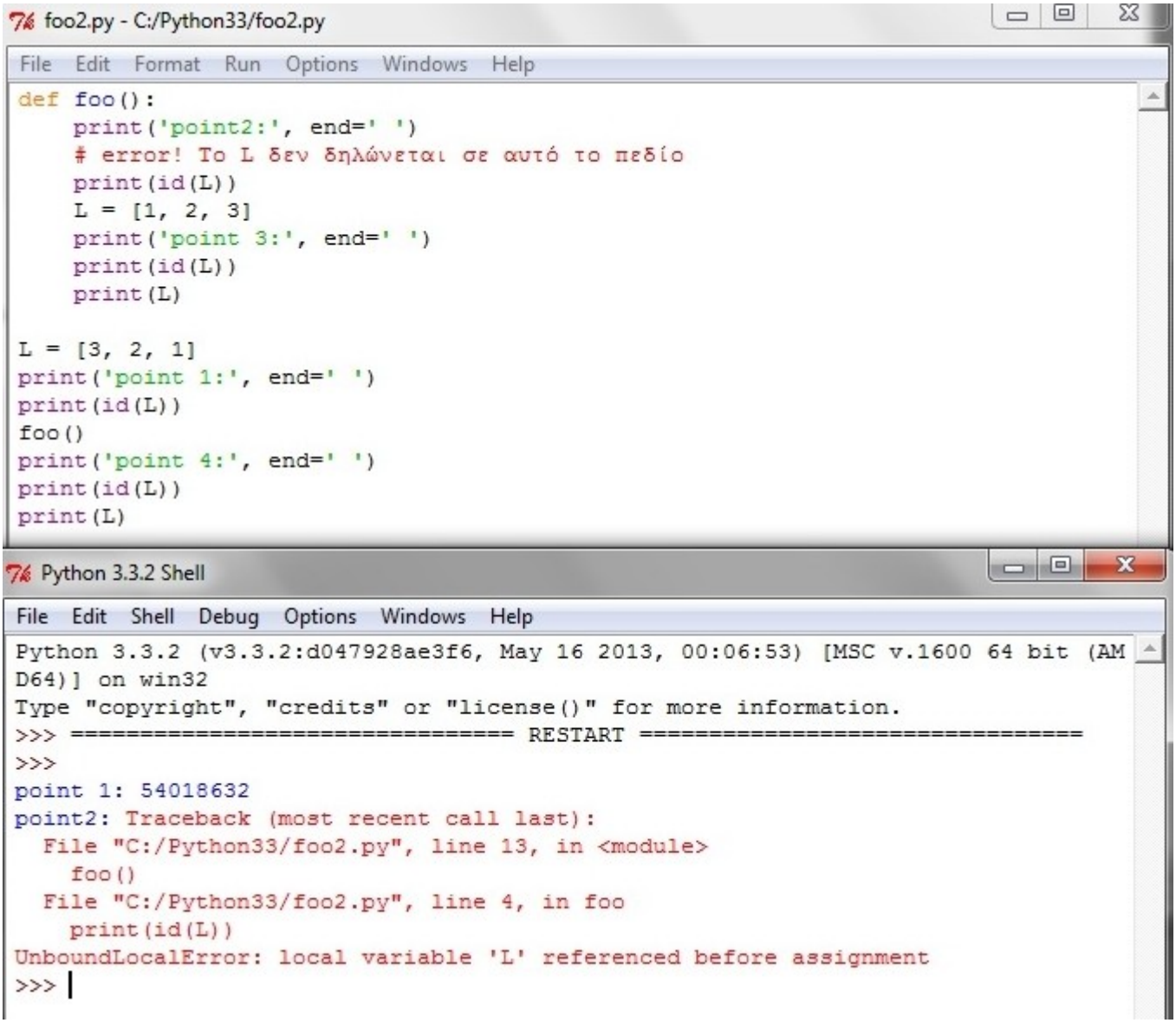
Σχήμα 4 : Ο χώρος ονομάτων μετά τη δημιουργία ενός λεξικού που περιέχει λίστες.

➔ Στο παρακάτω παράδειγμα, το αντικείμενο λίστας L καλείται κατά αναφορά. ΄Ετσι, κατά αρχάς το x δείχνει στην ίδια διεύθυνση που έδειχνε και το L. Στην συνέχεια όμως, δημιουργείται ένα καινούργιο αντικείμενο και το x πλέον δείχνει σε αυτό. Για αυτόν το λόγο και αλλάζει η διεύθυνση όπου δείχνει πια. Τέλος, η συνάρτηση τερματίζει, και το L δείχνει πια στην αρχική του θέση, και έχει τις ίδιες τιμές με πριν κληθεί η συνάρτηση.
Παρατηρούμε, με άλλα λόγια, πως αρχικά περνιέται μια αναφορά του αντικειμένου μέσα στην συνάρτηση. ΄Ομως μέσα στην συνάρτηση δημιουργείται ένα νέο αντικείμενο αφού οι λίστες δεν αναφέρονται με ψευδώνυμα, και το x τώρα πια δείχνει σε αυτό. Μόλις τερματίσει η συνάρτηση, η μεταβλητή L που η τιμή της δεν έχει αλλάξει από την κλήση της συνάρτησης foo() συνεχίζει να δείχνει στο ίδιο αντικείμενο το οποίο και δεν έχει τροποποιηθεί από την συνάρτηση.



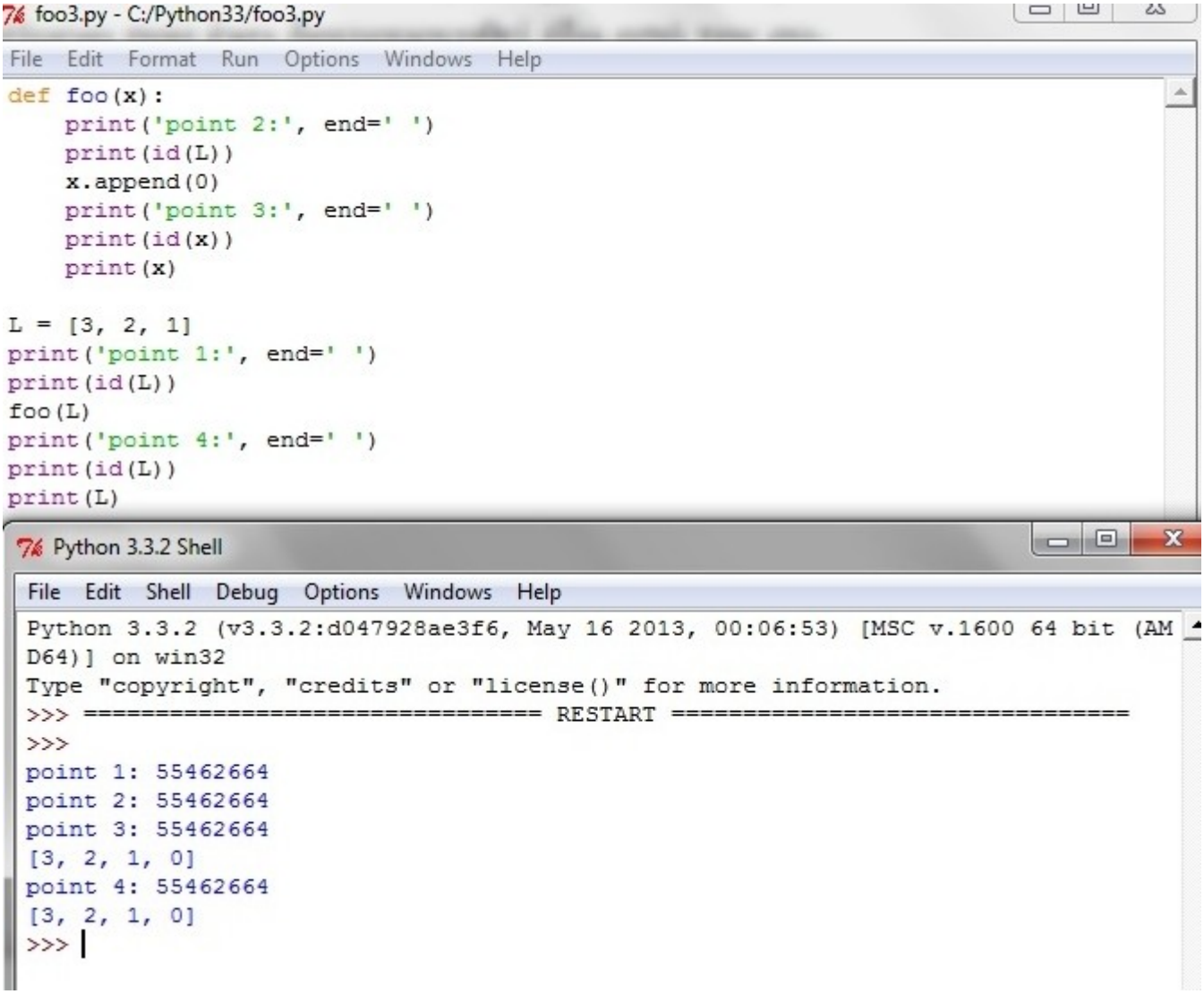
Τώρα γίνεται ακόμα πιο ξεκάθαρο γιατί το όνομα του ορίσματος στην συνάρτηση δεν παίζει κάποιο όλο και γιατί θα μπορούσαμε να το έχουμε αντικαταστήσει με οποιαδήποτε άλλη έγκυρη τιμή, όπως και με το L ακόμα. Ένας πρακτικός κανόνας για να αναγνωρίζουμε σε ποιο L σε αυτή την περίπτωση αναφερόμαστε, είναι στο να έχουμε υπόψη μας αυτό που ρίσκεται πιο κοντά.

Αξίζει να σημειώσουμε πως αν η foo() δεν έπαιρνε όρισμα, και την καλο-ύσαμε χρησιμοποιώντας μέσα στο σώμα της την μεταβλητή L, τότε η python δεν θα έλεγχε αν τυχόν υπάρχει από το εξωτερικό περιβάλλον μια μεταβλητή με το όνομα L αλλά θα έβγαζε error κατευθείαν λέγοντας πως δεν βρέθηκε μεταβλητή με το συγκεκριμένο όνομα μέσα στην εμβέλεια της συνάρτησης. Αυτή η αλλαγή σε σχέση με τις προηγούμενες εκδόσεις της Python συνέβη για την αποφυγή δύσκολων προς εντοπισμό λαθών που θα μπορούσαν να συμβούν με αυτό τον τρόπο.



Συνοψίζοντας λοιπόν μέχρι στιγμής καταλήγουμε στο συμπέρασμα ότι αναθέσεις σε τοπικές μεταβλητές μέσα σε μια συνάρτηση δεν επηρεάζουν τις τιμές των μεταβλητών που είναι δηλωμένες έξω από αυτή την συνάρτηση.

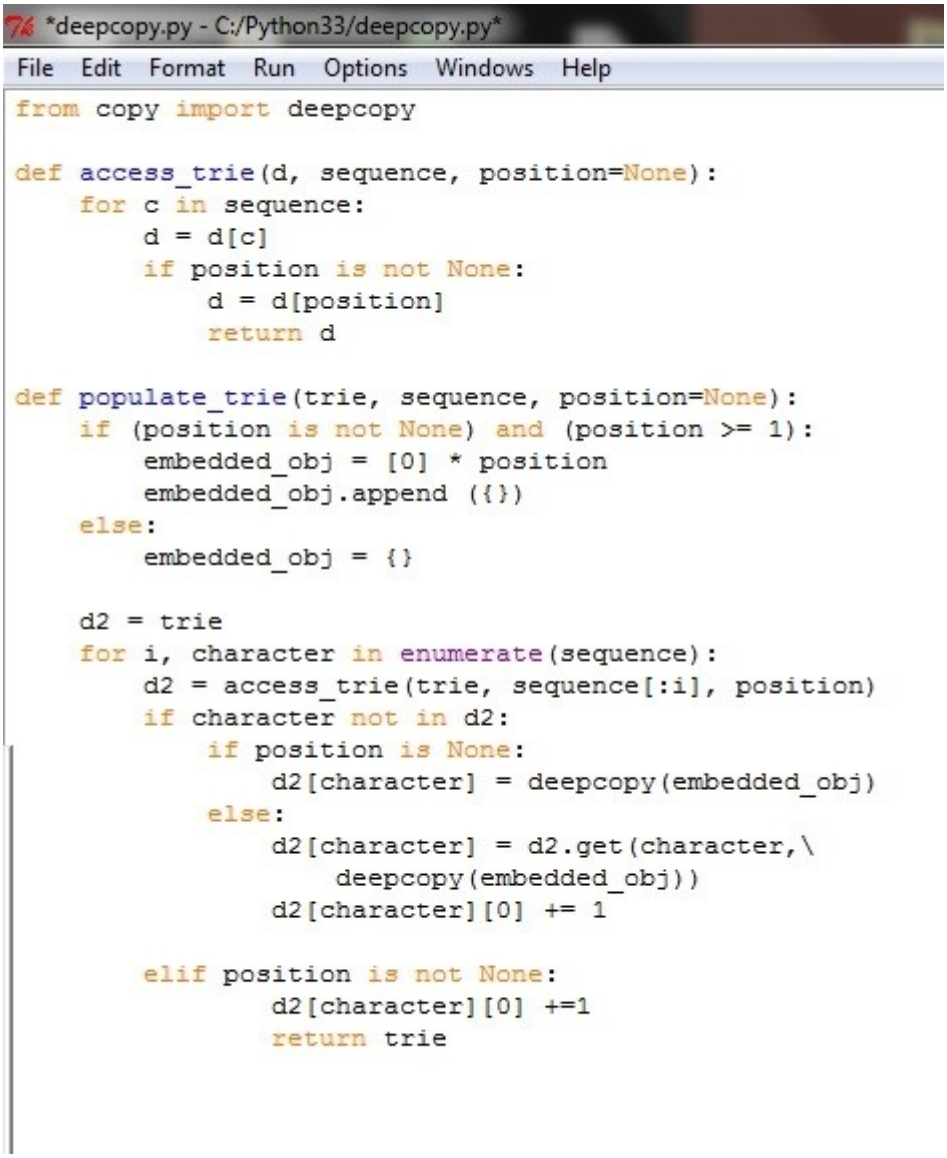
Αντίθετα όμως, όπως λέπουμε στο ακόλουθο παράδειγμα, αν μια μεταβλητή αναφέρεται σε αντικείμενο που έχει δημιουργηθεί έξω από την συνάρτηση που ρισκόμαστε τώρα, και εφόσον αυτή η αναφορά δεν αλλάζει, οποιαδήποτε αλλαγή κάνουμε σε αυτό το αντικείμενο, επηρεάζεται και εξωτερικά της συνάρτησης, αφού αναφερόμαστε στις ακριβώς ίδιες θέσεις μνήμης.



Αντιγραφή αντικειμένων

Ανάλογα με τον τύπο του αντικειμένου ο τελεστής ανάθεσης μπορεί να δημιουργήσει μόνο μια καινούργια αναφορά σε μια μεταβλητή και να μην αντιγράψει τα περιεχόμενα του αντικείμενου. Αν θέλουμε κάτι τέτοιο, πρέπει να χρησιμοποιήσουμε τη συνάρτηση **deepcopy**.

➔ Στο παράδειγμα που ακολουθεί θα δούμε πως μπορούμε να εκμεταλλευτούμε την ευελιξία που μας παρέχει η Python με τις αναφορές αλλά και όταν χρειαστεί να δημιουργούμε καινούργια αντίγραφα των αντικειμένων.



Πως λειτουργεί το παραπάνω πρόγραμμα :

Στην πρώτη συνάρτηση access_trie() βλέπουμε πως χρησιμοποιούμε τις αναφορές που δημιουργούμε με τον τελεστή της ανάθεσης ώστε να μπορέσουμε να προσπελάσουμε κάθε εμφωλευμένο λεξικό αφού έχουμε πρώτα προσπελάσει αυτό που το περιέχει. Για παράδειγμα, ας υποθέσουμε πως το όρισμα position είναι None, που σημαίνει πως ο κώδικας μέσα στο if δεν εκτελείται ποτέ, επομένως μπορούμε να επικεντρωθούμε στις υπόλοιπες δυο γραμμές του κώδικα.

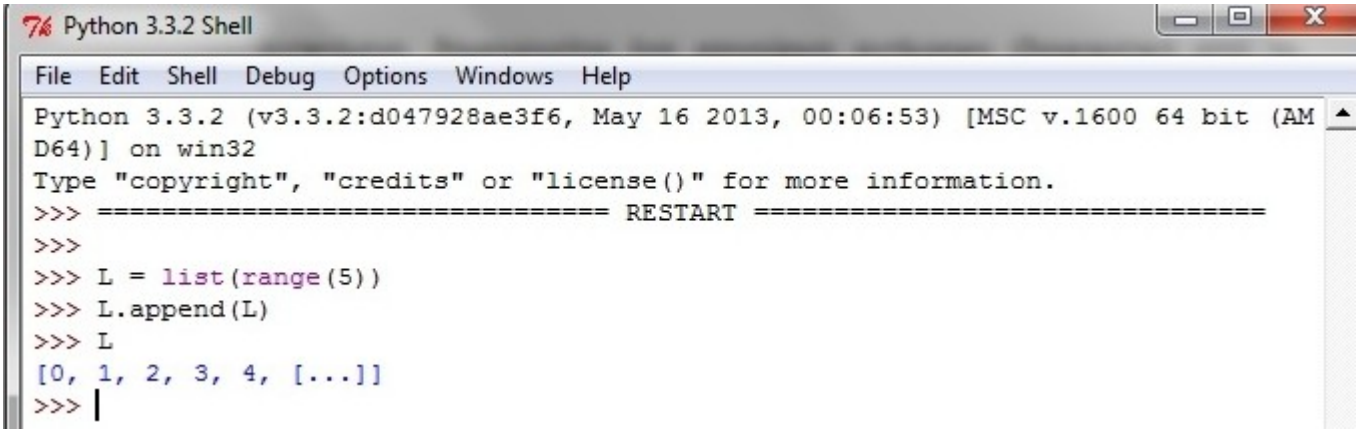
Προσπελαύνουμε ακολουθιακά κάθε στοιχείο της ακολουθίας sequence και με βάση αυτό το στοιχείο δεικτοδοτούμε ένα λεξικό. Αρχικά, το λεξικό που δεικτοδοτούμε είναι αυτό το οποίο έχουμε πάρει ως όρισμα από το περιβάλλον της συνάρτησης. Το αποτέλεσμα που μας επιστρέφεται είναι μια αναφορά σε ένα άλλο λεξικό που περιέχεται στο αρχικό. Αυτή η αναφορά αποθηκεύεται στη τοπική μεταβλητή d η οποία και κρύβει (hides) πλέον το όρισμα που είχαμε δεχθεί αφού έχουν το ίδιο όνομα. ΄Ετσι, οποιαδήποτε τροποποίηση του d από εδώ και πέρα θα έχει ισχύ μόνο στο τοπικό περιβάλλον της συνάρτησης και δεν θα επηρεάσει την τιμή των

ορισμάτων της. Συνεχίζοντας την ίδια διαδικασία ρίσκουμε το επιθυμητό αποτέλεσμα το οποίο και μπορούμε να επιστρέψουμε στο περιβάλλον εκτέλεσης της συνάρτησης.

΄Οσον αφορά τη δεύτερη συνάρτηση populate_trie η λογική που ακολουθείται είναι παρόμοια, όμως επειδή θέλουμε να τροποποιήσουμε το αντικείμενο και να του προσθέτουμε κάποια άλλα προκαθορισμένα αντικείμενα μας δίνεται η ευκαιρία να μελετήσουμε την συνάρτηση deepcopy.

Η λογική σε αυτό τη συνάρτηση είναι παρόμοια με την προηγούμενη. Το ενδιαφέρον κομμάτι αρχίζει μόλις ρούμε το μέρος του λεξικού όπου θέλουμε να εμφωλεύσουμε το καινούργιο αντικείμενο. Αν δεν αντιγράφαμε το embedded_obj εκεί, θα είχαμε εισάγει μια αναφορά στο ίδιο αντικείμενο σε διαφορετικά σημεία. Με άλλα λόγια θα είχαμε πάει ένα επίπεδο πιο μέσα στη δομή και θα είχαμε εισάγει μέσα στο embedded_obj μια αναφορά στο embedded_obj. Αυτό ονομάζεται **κυκλική δομή.** Αντιγράφοντας λοιπόν το αντικείμενο, δημιουργείται ένα καινούργιο αντίγραφο (διαφορετικό από το προηγούμενο) το οποίο και εισάγουμε στη κατάλληλη θέση.

΄Ενα πιο κλασικό παράδειγμα κυκλικής δομής είναι το παρακάτω :



Αντικείμενα και κλάσεις-Αντικειμενοστραφής προγραμματισμός

Όλα τα προγράμματα που γράψαμε μέχρι στιγμής, τα σχεδιάσαμε με τη χρήση συναρτήσεων, δηλαδή μπλοκ εντολών που χειρίζονται δεδομένα. Αυτή η μέθοδος αποκαλείται **διαδικασιοστρεφής προγραμματισμός**. Υπάρχει και άλλη μέθοδος συγγραφής προγραμμάτων, η οποία συνδυάζει δεδομένα και λειτουργικότητα εμπερικλείοντας τα σε ένα αντικείμενο. Η μέθοδος αυτή ονομάζεται **αντικειμενοστρεφής προγραμματισμός**.

Συνήθως η χρήση διαδικασιοστρεφούς προγραμματισμού αρκεί, αλλά αν γράφετε μεγάλα προγράμματα ή αντιμετωπίζετε ένα πρόβλημα που εξυπηρετείται καλύτερα

από αυτή τη μέθοδο, μπορείτε να χρησιμοποιήσετε τεχνικές αντικειμενοστρεφούς προγραμματισμού.

Οι κλάσεις και τα αντικείμενα είναι οι δύο κύριες πτυχές του αντικειμενοστρεφούς προγραμματισμού.

Μια κλάση δημιουργεί ένα νέο τύπο, όπου τα αντικείμενα είναι υποστάσεις (instances) της κλάσης. Μια αναλογία θα ήταν με μεταβλητές του τύπου int και μεταφράζεται αντίστοιχα λέγοντας ότι οι μεταβλητές που αποθηκεύουν ακέραιους αριθμούς είναι υποστάσεις (αντικείμενα) της κλάσης int.

Ακόμη και οι ακέραιοι αριθμοί αντιμετωπίζονται ως αντικείμενα (της κλάσης int). Σε αντίθεση με τη C++ και Java (πριν την έκδοση 1.5), όπου οι ακέραιοι είναι αρχέγονοι ατόφιοι τύποι δεδομένων (primitive native types)→ εντολή help(int) για περισσότερες λεπτομέρειες

Τα αντικείμενα αποθηκεύουν δεδομένα με τη χρήση κοινών μεταβλητών που ανήκουν στο αντικείμενο. Οι μεταβλητές που ανήκουν σε ένα αντικείμενο ή κλάση αναφέρονται ως **πεδία (fields)**. Τα αντικείμενα μπορούν επίσης να είναι λειτουργικά με τη χρήση συναρτήσεων που ανήκουν σε μια κλάση. Τέτοιες συναρτήσεις αποκαλούνται **μέθοδοι της κλάσης**. Η ορολογία αυτή είναι σημαντική διότι μας επιτρέπει να ξεχωρίσουμε ανεξάρτητες συναρτήσεις και μεταβλητές από αυτές που ανήκουν σε ένα αντικείμενο ή κλάση. Καθολικά, τα πεδία και οι μέθοδοι αναφέρονται ως **ιδιοχαρακτηριστικά (attributes)** αυτής της κλάσης.

Τα πεδία είναι δύο τύπων μπορεί να ανήκουν στην υπόσταση/αντικείμενο της κλάσης ή στην ίδια την κλάση. Αποκαλούνται **μεταβλητές υπόστασης** (instance variables) και **μεταβλητές κλάσης** αντίστοιχα.

Μια κλάση δημιουργείται με τη λέξη-κλειδί class. Τα πεδία και οι μέθοδοι της κλάσης απαριθμούνται σε μια πλοκάδα κώδικα σε εσοχή.

Τα βασικά χαρακτηριστικά που παρατηρούμε κατά τον σχεδιασμό και την υλοποίηση ενός τυπικού προγράμματος μέσω αντικειμενοστρέφειας είναι:

- ✓ **Κλάση (Class):** Καθορίζει τα ασικά χαρακτηριστικά (attributes) και συμπεριφορές (methods) των αντικειμένων που περιγράφονται από αυτή την κλάση. ΄Ενα παράδειγμα θα μπορούσε να είναι η κλάση «σχολή». Γενικότερα

μπορούμε να σκεφτόμαστε ότι αντιπροσωπεύουν τα αδρά χαρακτηριστικά που ενυπάρχουν σε όλα τα στιγμιότυπα μιας κλάσης

- ✓ **Αντικείμενο (Object):** Μια συγκεκριμένη μορφή μιας κλάσης, δηλαδή ένα από τα αντικείμενα που περιγράφει η κλάση που το χαρακτηρίζει. Για παράδειγμα «Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών».
- ✓ **Στιγμιότυπο (Instance)**: Η κατάσταση ενός αντικείμενο κατά την εκτέλεση του προγράμματος μέσα από την συγκεκριμενοποίηση του.
- ✓ **Μέθοδος (Method)**: Η συμπεριφορά και οι δυνατότητες ενός αντικειμένου.
- ✓ **Κληρονομικότητα (Inheritance):** ΄Οταν μια κλάση αποτελεί εξειδίκευση μιας άλλης, και έτσι έχει όλα τα χαρακτηριστικά που περιγράφουν την κλάση από την οποία κληρονομεί, καθώς και κάποια ιδιαίτερα που περιγράφουν ειδικά την συγκεκριμένη ομάδα αντικειμένων. Για παράδειγμα ο κλάση «σχολή» μπορεί να κληρονομεί από μια γενικότερη κλάση «ΑΕΙ».

Η μεταβλητή self

Οι μέθοδοι κλάσης έχουν μια συγκεκριμένη διαφορά από τις κοινές συναρτήσεις - πρέπει να έχουν ένα επιπλέον όνομα και πρέπει να προστεθεί στην αρχή της λίστας παραμέτρων, εσείς δε χρειάζεται να ορίσετε μια τιμή σε αυτή την παράμετρο όταν καλείτε τη μέθοδο, σας το παρέχει η Python. Αυτή η συγκεκριμένη μεταβλητή αναφέρεται στο ίδιο το αντικείμενο και κατά συνθήκη, της αποδίδεται το όνομα ***self***.

Παρόλο που μπορείτε να δώσετε οποιοδήποτε όνομα σε αυτή την παράμετρο, συνιστάται το όνομα μεταβλητής self -οποιοδήποτε άλλο όνομα είναι απαξιωτικό. Υπάρχουν πολλά πλεονεκτήματα για τη χρήση ενός πρότυπου ονόματος μεταβλητής -ο κάθε αναγνώστης του κώδικα του προγράμματος σας θα το αναγνωρίσει άμεσα. Ακόμη και γραφικά περιβάλλοντα ανάπτυξης λογισμικού (Integrated Development Environments) θα σας συνδράμουν, αν χρησιμοποιείτε το όνομα μεταβλητής self.

Το όνομα μεταβλητής self στην Python ισούται με το δείκτη this στην C++ και την αναφορά this στην Java και C.

⇨ Πώς η Python ορίζει την τιμή για τη self και γιατί σας απαλλάσσει από τον χειροκίνητο ορισμό της;

΄Ενα παράδειγμα θα το ξεκαθαρίσει. Ας πούμε ότι μια κλάση ονομάζεται MyClass και μια υπόσταση αυτής ονομάζεται myobject. Αν καλέσετε μια μέθοδο αυτού του αντικειμένου myobject.method(arg1, arg2), μετατρέπεται αυτόματα από την Python σε MyClass.method(myobject, arg1, arg2) -αυτή είναι και η ιδιαιτερότητα του ονόματος μεταβλητής self. Αυτό σημαίνει επίσης ότι εάν έχετε μια μέθοδο η

οποία δεν λαμβάνει ορίσματα, τότε θα πρέπει να έχει τουλάχιστον ένα όρισμα, τη μεταβλητή self.

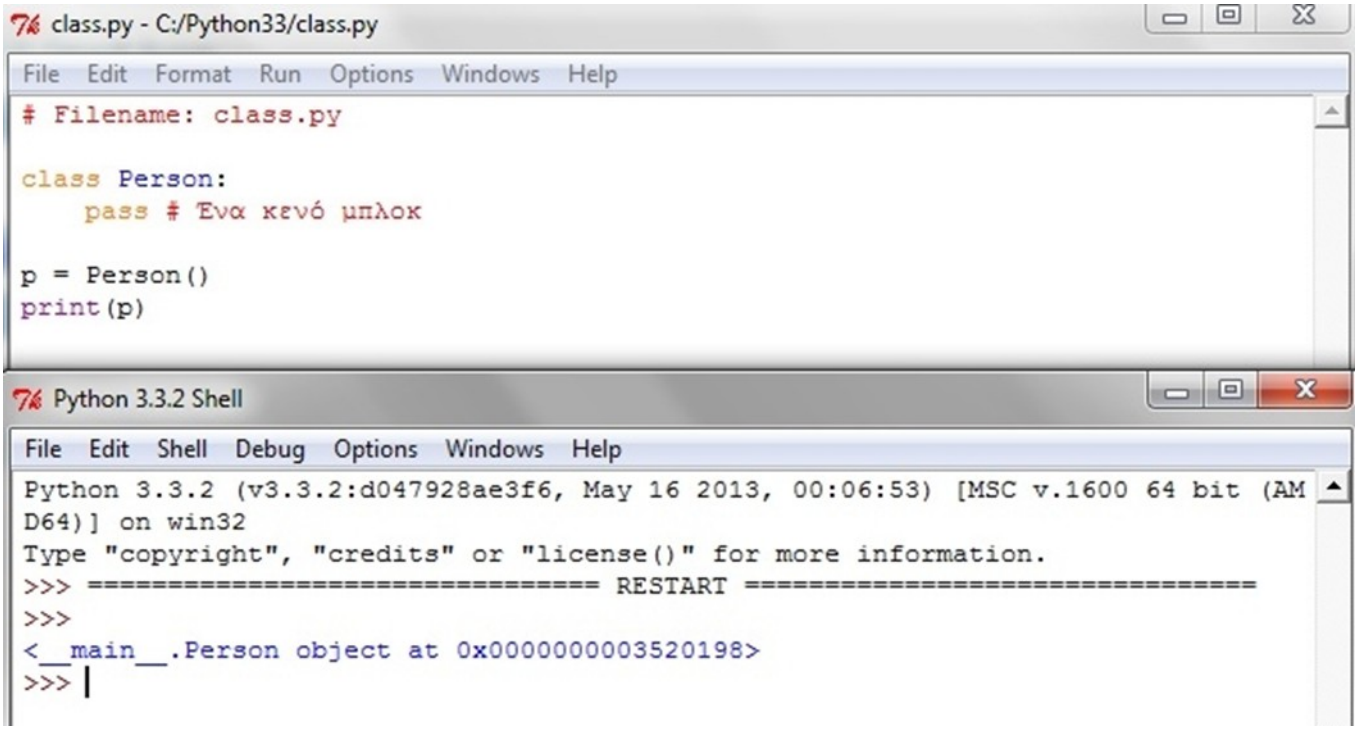
Βασικές ΄Εννοιες

Η πιο απλή κλάση που θα μπορούσαμε να γράψουμε είναι :

```
class icte():

    pass

department_of_uowm = icte()
```

όπου με την δήλωση class icte(): δημιουργούμε μια νέα κλάση icte ενώ στην συνέχεια, με την δήλωση department_of_uowm = icte() δημιουργούμε ένα αντι-κείμενο αυτής της κλάσης.

➲ Ακολουθεί ένα ακόμα παράδειγμα κλάσης :



Πως δουλεύει το παραπάνω πρόγραμμα :

Δημιουργούμε μια νέα κλάση με την εντολή class και το όνομα της κλάσης. Αυτή ακολουθείται από μια πλοκάδα εντολών σε εσοχή που αποτελούν τον κορμό της κλάσης. Στην περίπτωσή μας έχουμε μια κενή πλοκάδα και δηλώνεται με την εντολή pass.

Στη συνέχεια, δημιουργούμε ένα αντικείμενο/υπόσταση αυτής της κλάσης με το όνομα αυτής ακολουθούμενο από ένα ζευγάρι παρενθέσεων. Για επαλήθευση,

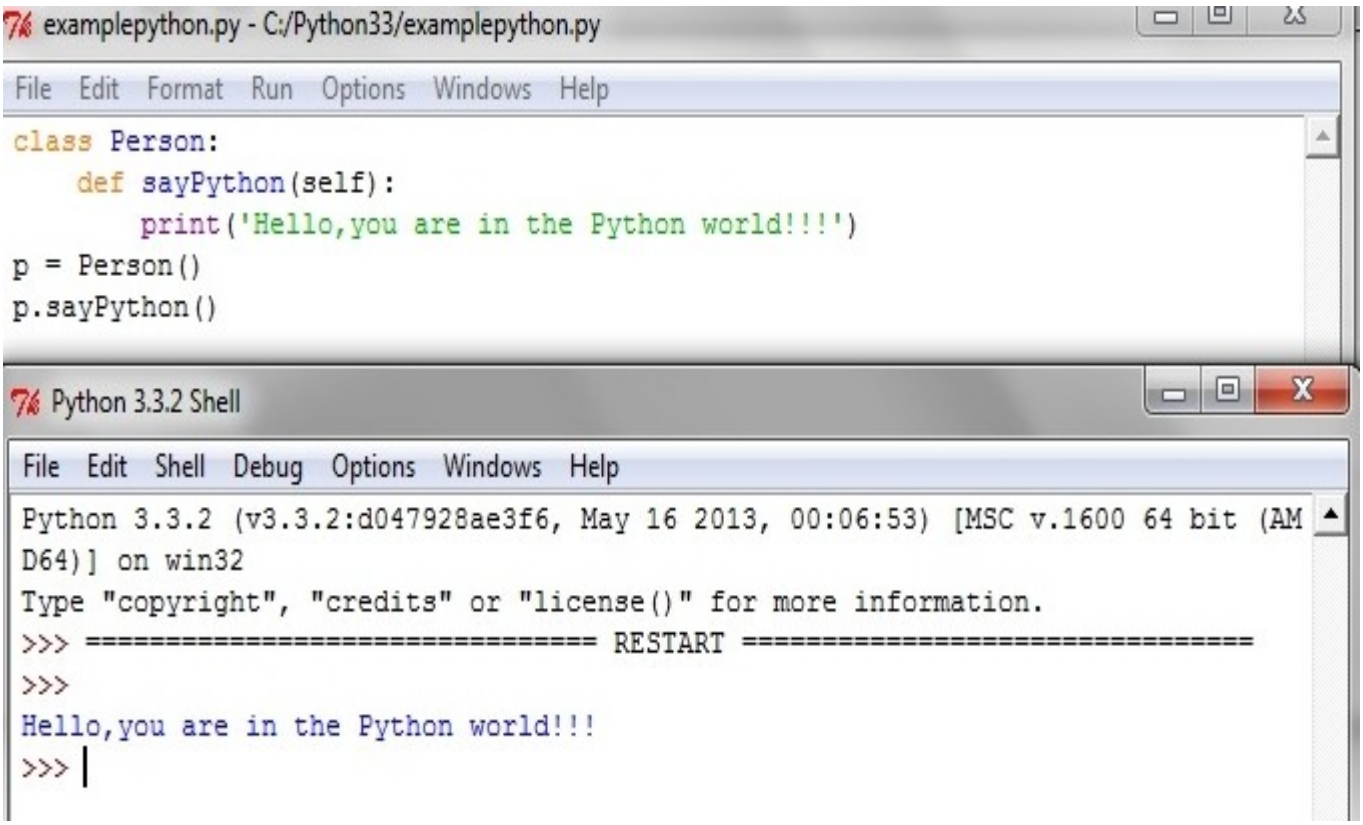
επιβεβαιώνουμε τον τύπο της μεταβλητής με απλή εκτύπωση στην οθόνη. Μας αναφέρει ότι έχουμε μια υπόσταση της κλάσης Person στο άρθρωμα __main__.

Παρατηρήστε ότι η διεύθυνση μνήμης του υπολογιστή όπου αποθηκεύτηκε το αντικείμενο εκτυπώνεται επίσης. Η διεύθυνση θα έχει διαφορετική τιμή στον υπολογιστή σας διότι η Python αποθηκεύει το αντικείμενο όπου βρει διαθέσιμο χώρο.

Μέθοδοι αντικειμένου

Οι κλάσεις/αντικείμενα δέχονται μεθόδους όπως και συναρτήσεις, εκτός της πρόσθετης μεταβλητής self.

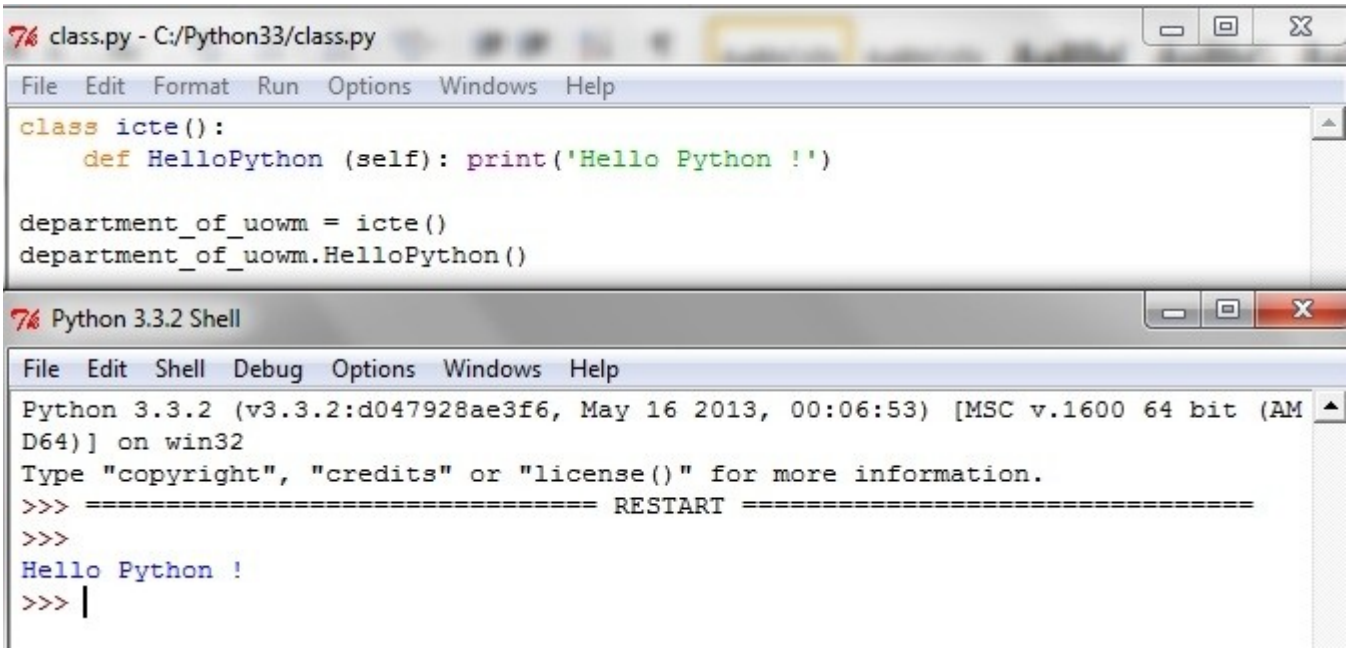
- 1° παράδειγμα :



Πώς δουλεύει το παραπάνω πρόγραμμα :

Εδώ βλέπουμε τη μεταβλητή self σε δράση. Παρατηρήστε πως η μέθοδος sayPython δε δέχεται παραμέτρους, αλλά εμπερικλείει τη μεταβλητή self στον ορισμό της συνάρτησης.

➲ 2ο παράδειγμα :



Αξίζει να παρατηρήσουμε πως, όπως και οι συναρτήσεις, έτσι και οι μέθοδοι μιας κλάσης ορίζονται με την λέξη κλειδί def. Επίσης, κατά την κλήση της μεθόδου ενός αντικειμένου δεν θέτουμε εμείς κάποια τιμή στο όρισμα self αλλά αυτό γίνεται αυτόματα. Κάθε μέθοδος μιας κλάσης πρέπει να έχει ως πρώτο όρισμα το self, έτσι ώστε στην συνέχεια μέσω αυτού του ορίσματος να γίνεται γνωστό πιο αντικείμενο κάλεσε τη μέθοδο.

Με απλά λόγια, μια κλάση είναι ένας τρόπος κατασκευής ενός αντικειμένου. ΄Ενα αντικείμενο είναι ένας τρόπος να χειριζόμαστε πιο εύκολα χαρακτηριστικά και συμπεριφορές ομαδοποιημένα. Μπορούμε να σκεφθούμε τα αντικείμενα σαν κάτι που υπάρχει στον πραγματικό κόσμο, ενώ τις κλάσεις έναν τρόπο κατασκευής τους και προσδιορισμού των ιδιοτήτων τους.

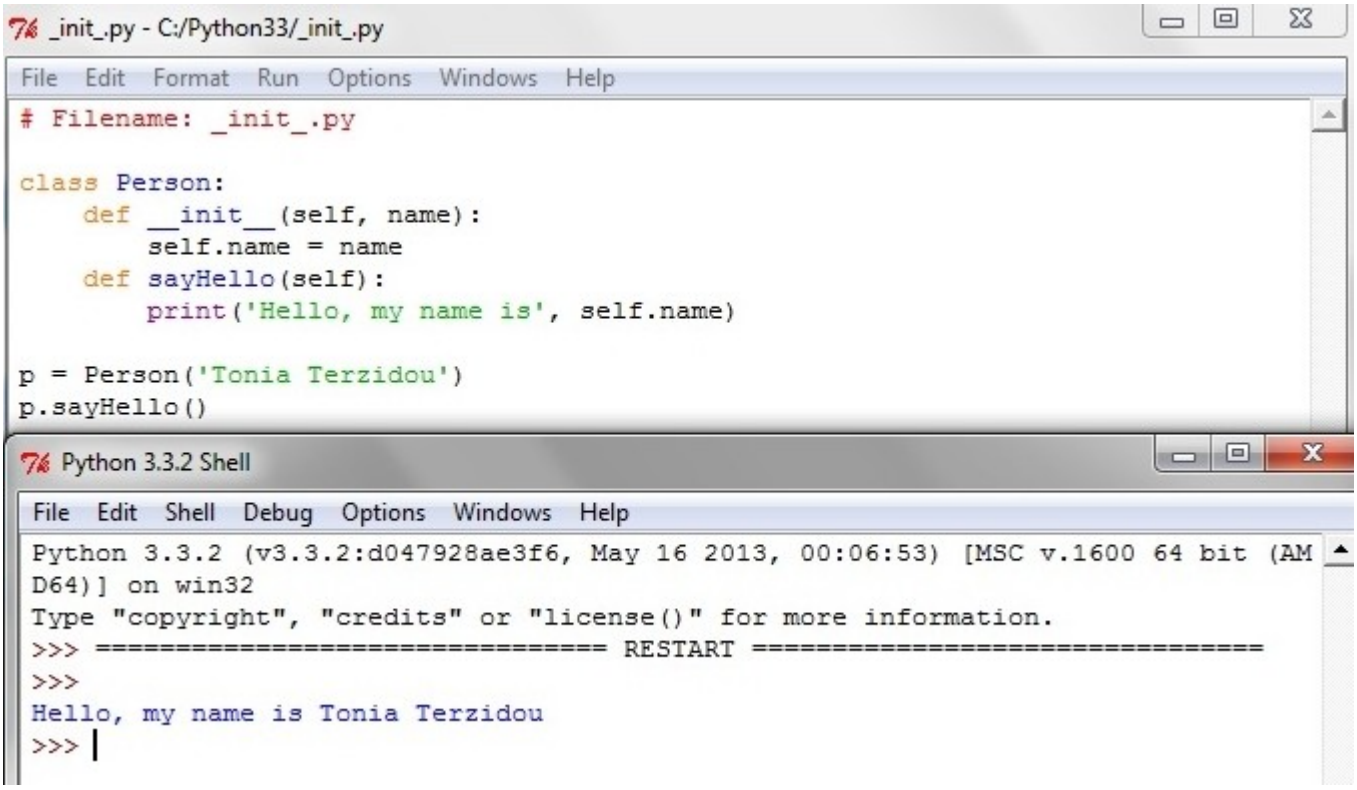
Η μέθοδος __init__

Υπάρχουν πολλά ονόματα μεθόδων που έχουν ειδική σημασία στις κλάσεις Python. Τώρα θα δούμε τη σημασία της μεθόδου __init__.

Η μέθοδος __init__ εκτελείται μόλις ένα αντικείμενο μιας κλάσης αρχικοποιείται. Η μέθοδος αυτή είναι χρήσιμη για την κατάλληλη αρχικοποίηση που επιθυμείτε για το αντικείμενο σας. Παρατηρήστε τις διπλές κάτω παύλες στην αρχή και στο τέλος του ονόματος.

Ακολοθούν παραδείγματα :

➲ 1° παράδειγμα :



Πώς δουλεύει το παραπάνω πρόγραμμα :

Εδώ ορίζουμε τη μέθοδο __init__ να λαμβάνει ως παράμετρο την name (μαζί με τη γνωστή self). Τώρα, δημιουργούμε ένα νέο πεδίο που αποκαλείται επίσης name. Προσέξτε ότι πρόκειται για δύο διαφορετικές μεταβλητές παρότι αναφέρονται και οι δύο ως 'name'. Δεν πρόκειται να υπάρξει σύγχιση εδώ διότι ο διάστικτος συμβολισμός (dotted notation) self.name σημαίνει ότι υπάρχει που λέγεται "name" και είναι μέρος του αντικειμένου που λέγεται "self" ενώ το άλλο name είναι μια τοπική μεταβλητή. Αποφεύγουμε τη σύγχιση μεταξύ των δύο, αφού υποδεικνύουμε ρητά σε ποιό name αναφερόμαστε.

Το σημαντικότερο, προσέξτε ότι δεν καλούμε ρητά τη μέθοδο __init__, αλλά μεταφέρουμε τα ορίσματα εντός των παρενθέσεων μετά το όνομα της κλάσης, όταν δημιουργούμε μια νέα υπόσταση της κλάσης. Αυτή είναι η ιδιαίτερη σημασία της μεθόδου.

Μπορούμε πλέον να χρησιμοποιήσουμε το πεδίο self.name στις μεθόδους μας και παρουσιάζεται στη μέθοδο sayHello.

➲ 2ο παράδειγμα :

examplewithobject.py - C:/Python33/examplewithobject.py

File Edit Format Run Options Windows Help

```
class Person():
    def __init__(self, name, height, university, mood, age):

        self.name = name
        self.height = height
        self.university = university
        self.mood = mood
        self.age = age
        self.tired = True

    def print_attributes(self):

        print('Name is', self.name)
        print('Height is', self.height)
        print('University is',self.university)
        print('Mood is ', self.mood)
        print('Age is ', self.age)
        print('Tired is',self.tired)

tonia = Person('Tonia',1.68,'Uowm','good',22)
tonia.print_attributes()
```

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Name is Tonia
Height is 1.68
University is Uowm
Mood is  good
Age is  22
Tired is True
>>>
```

Πως λειτουργεί το παραπάνω πρόγραμμα :

Οπως βλέπουμε παραπάνω, για να κατασκευάσουμε ένα αντικείμενο τύπου Person() πρέπει να του περάσουμε τα κατάλληλα ορίσματα που θα προσδιορίσουν τις ιδιότητες του.

Η ειδική μέθοδος που καλείται μόλις κατασκευάζουμε το αντικείμενο είναι η __init__(). Το όνομα της σε κάθε κλάση πρέπει να είναι το συγκεκριμένο.

Η __init__() παίρνει πάντα ως πρώτο όρισμα την λέξη self που σημαίνει ότι η συμπεριφορά που προσδιορίζεται αφορά το αντικείμενο από το οποίο καλείται ή στην προκειμένη περίπτωση, το αντικείμενο που πάει να δημιουργηθεί. Τα επόμενα ορίσματα είναι για να προσδιορίσουμε τις ιδιότητες του αντικειμένου.

Μεταβλητές κλάσης και αντικειμένου

Έχουμε ήδη μελετήσει τη λειτουργικότητα των κλάσεων και αντικειμένων (δηλαδή μεθόδων), ας εξετάσουμε τώρα τα δεδομένα. Τα δεδομένα, δηλαδή τα πεδία, δεν είναι παρά συνηθισμένες μεταβλητές και οριοθετούνται από τους χώρους ονομάτων (namespaces) των κλάσεων και αντικειμένων. Αυτό σημαίνει πρακτικά ότι τα ονόματα αυτά είναι έγκυρα μόνο εντός αυτών των κλάσεων και αντικειμένων. Γι' αυτό αποκαλούνται **name spaces**.

Υπάρχουν δύο τύποι πεδίων (fields) -μεταβλητές κλάσης και μεταβλητές αντικειμένων και οι οποίες ταξινομούνται ανάλογα με την κλάση ή αντικείμενο που τις κατέχει αντίστοιχα.

Οι **μεταβλητές κλάσης** είναι κοινόχρηστες -μπορούν να προσπελαστούν από όλες τις υποστάσεις αυτής της κλάσης. Υπάρχει μόνο ένα αντίγραφο της μεταβλητής κλάσης και όταν κάποιο αντικείμενο τροποποιήσει τη μεταβλητή αυτή, η τροποποίηση θα είναι εμφανής σε όλες τις υποστάσεις.

Οι **μεταβλητές αντικειμένου** ανήκουν μεμονωμένα σε κάθε αντικείμενο/υπόσταση της κλάσης. Στην περίπτωση αυτή, κάθε αντικείμενο έχει το δικό του αντίγραφο του πεδίου, δηλαδή δεν είναι κοινόχρηστες και δεν σχετίζονται σε καμία περίπτωση με το συνονόματο πεδίο σε μια διαφορετική υπόσταση.

- ➲ Ένα παράδειγμα θα μας βοηθήσει στην κατανόησή του είναι το παρακάτω :

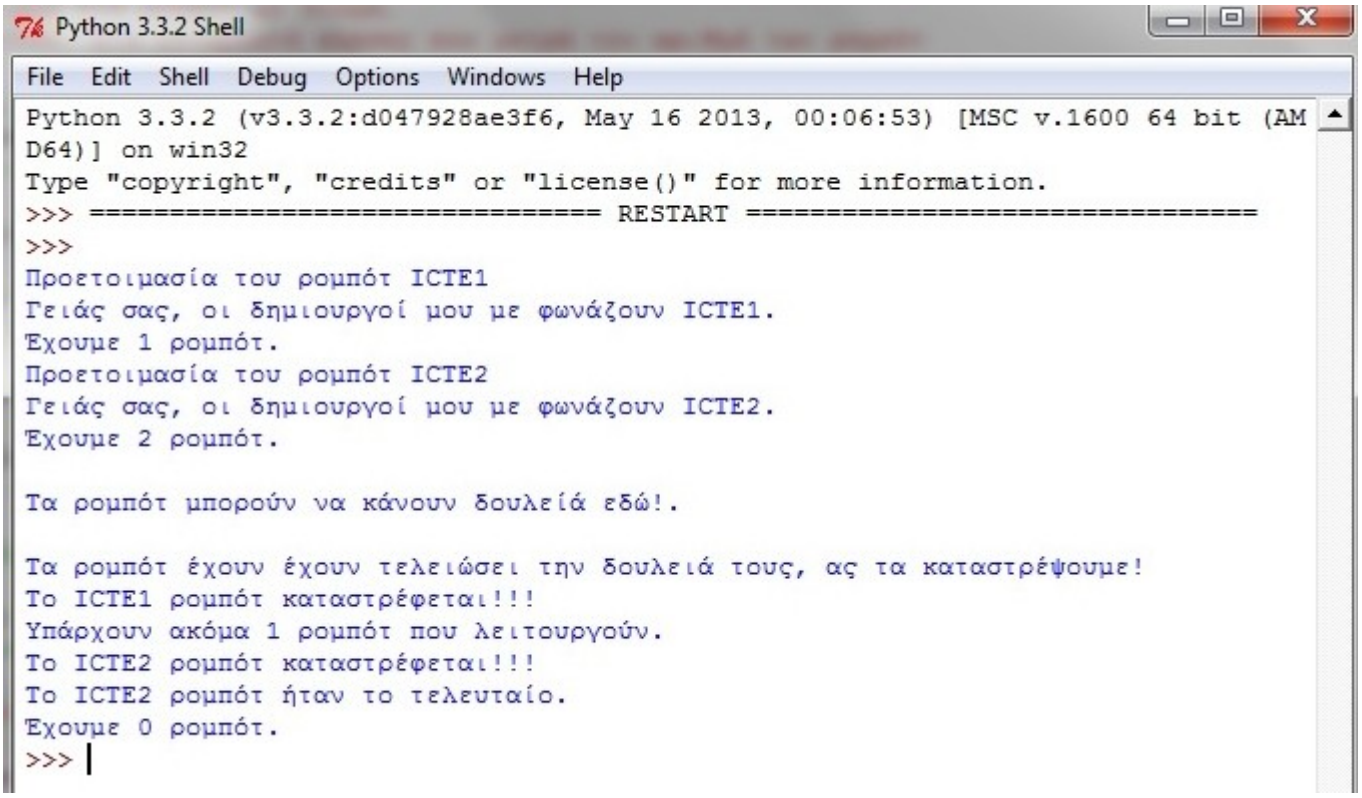
Είναι αρκετά μεγάλο το παράδειγμα αλλά βοηθά στην κατανόηση της φύσης των μεταβλητών κλάσεων και αντικειμένων.

robot.py - C:/Python33/robot.py

File Edit Format Run Options Windows Help

```
class Robot: # αναπαριστά ένα ρομποτ με όνομα.
    population = 0 # είναι μια μεταβλητή κλάσης που μετρά τον αριθμό των ρομπότ

    def __init__(self, name):  # προετοιμάζει τα δεδομένα
        self.name = name
        print('Προετοιμασία του ρομπότ {0}'.format(self.name))

        Robot.population += 1    #Όταν δημιουργείται το άτομο, προστίθεται το ρομπότ στον πληθυσμό

    def __del__(self): #καταστρέφεται
        print('Το {0} ρομπότ καταστρέφεται!!!'.format(self.name))

        Robot.population -= 1

        if Robot.population == 0:
            print('Το {0} ρομπότ ήταν το τελευταίο.'.format(self.name))
        else:
            print('Υπάρχουν ακόμα {0:d} ρομπότ που λειτουργούν.'.format(Robot.population))

    def sayHello(self): # χαιρετισμός από το ρομποτ
        print('Γειάς σας, οι δημιουργοί μου με φωνάζουν {0}.'.format(self.name))

    def howMany(): # τυπώνει τον τρέχων πληθυσμό.
        print('Έχουμε {0:d} ρομπότ.'.format(Robot.population))
    howMany = staticmethod(howMany)

droid1 = Robot('ICTE1')
droid1.sayHello()
Robot.howMany()

droid2 = Robot('ICTE2')
droid2.sayHello()
Robot.howMany()

print("\nΤα ρομπότ μπορούν να κάνουν δουλειά εδώ!.\n")

print("Τα ρομπότ έχουν έχουν τελειώσει την δουλειά τους, ας τα καταστρέψουμε! ")
del droid1
del droid2

Robot.howMany()
```

Τρέχοντας το παραπάνω πρόγραμμα, προκύπτει :



Πως δουλεύει το παραπάνω πρόγραμμα :

Η μεταβλητή population ανήκει στην κλάση Robot και συνεπώς είναι μια μεταβλητή κλάσης. Η μεταβλητή name ανήκει στο αντικείμενο (έχει ανατεθεί με τη χρήση της self) και είναι μια μεταβλητή αντικειμένου. Άρα, αναφερόμαστε στη μεταβλητή κλάσης population ως Robot.population και όχι ως self.population.

Αναφερόμαστε στη μεταβλητή αντικειμένου name με το συμβολισμό self.name των μεθόδων αυτού του αντικειμένου. ΠΡΟΣΟΧΗ σε αυτή την απλή διαφορά μεταξύ των μεταβλητών κλάσεων και αντικειμένων. Επίσης, μια μεταβλητή αντικειμένου με το ίδιο όνομα μιας μεταβλητής κλάσης θα αποκρύψει τη μεταβλητή κλάσης!

Το πεδίο howMany είναι πράγματι μια μέθοδος που ανήκει στην κλάση και όχι στο αντικείμενο. Αυτό σημαίνει ότι μπορούμε να οριστεί είτε ως classmethod ή ως staticmethod, ανάλογα με το αν θα πρέπει να γνωρίζουμε σε ποια κλάση ανήκει. Δεδομένου ότι δε χρειαζόμαστε τέτοιες πληροφορίες, θα επιλέξουμε τη staticmethod.

- Θα είχαμε επίσης το ίδιο αποτέλεσμα με τη χρήση διακοσμητών (decorators):

```
@staticmethod

def howMany(): # τυπώνει τον τρέχων πληθυσμό

print('Έχουμε {0:d} ρομπότ!'.format(Robot.population))
```

Μπορούμε να φανταστούμε τους διακοσμητές ως συντομεύσεις για την κλήση μιας ρητής εντολής (explicit statement), όπως είδαμε σε αυτό το παράδειγμα.

Παρατηρήστε ότι η μέθοδος __init__ χρησιμοποιείται για την αρχικοποίηση της υπόστασης Robot με ένα όνομα. Σε αυτή τη μέθοδο, ο αριθμός του πληθυσμού population αυξάνεται κατά 1, προσθέτοντας ένα ακόμη ρομπότ. Παρατηρήστε επίσης ότι οι τιμές της κλάσης self.name είναι ειδικές για κάθε αντικείμενο, γεγονός που υποδεικνύει τη φύση των μεταβλητών αντικειμένου.

Θυμηθείτε ότι οφείλετε να αναφέρεστε στις μεταβλητές και μεθόδους του ιδίου αντικειμένου αποκλειστικά με τη χρήση της μεθόδου self . Αυτό αποκαλείται ως αναφορά ιδιοχαρακτηριστικού (attribute reference).

Σε αυτό το πρόγραμμα, βλέπουμε επίσης τη **χρήση docstrings** για κλάσεις αλλά και για μεθόδους. Μπορούμε να προσπελάσουμε τις docstring της κλάσης ενόσω τρέχει το πρόγραμμα χρησιμοποιώντας το Robot.__doc__ και τη μέθοδο docstring ως Robot.sayHi.__doc__ Ακριβώς όπως η μέθοδος __init__, υπάρχει η ειδική μέθοδος __del__ και καλείται όταν τερματίζεται ένα αντικείμενο, δηλαδή όταν αποδεσμεύεται και επιστρέφεται στο σύστημα για επαναχρησιμοποίηση αυτή η περιοχή μνήμης. Σε αυτή τη μέθοδο, μειώνουμε τον αριθμό των ρομπότ Robot.population κατά 1.

Η μέθοδος __del__ εκτελείται όταν το αντικείμενο δεν είναι σε χρήση πλέον και δεν εγγυάται πότε θα εκτελεστεί αυτή. Αν θέλετε να το δείτε σε δράση, πρέπει να χρησιμοποιήσετε την εντολή del και το έχουμε σε αυτό το πρόγραμμα.

Όλα τα μέλη κλάσης (class members) (συμπεριλαμβανομένων και των μελών δεδομένων (data members))

είναι δημόσια (public) και όλες οι μέθοδοι είναι εικονικές (virtual) στην Python.

Μια εξαίρεση: Για την ονομασία μελών δεδομένων με τη χρήση του διπλού προθέματος κάτω παύλας (double underscore prefix), π.χ __privatevar, η Python χρησιμοποιεί κατάτμηση ονόματος (name-mangling) για να οριστεί η μεταβλητή ως ιδιωτική αποτελεσματικά.

Άρα, η σύμβαση που ακολουθείται είναι ότι η χρήση μιας μεταβλητής εντός αποκλειστικά μιας κλάσης ή ενός αντικειμένου, οφείλει να ξεκινά με μια κάτω παύλα και ότι όλα τα υπόλοιπα ονόματα είναι δημόσια (public) και μπορούν να χρησιμοποιηθούν από άλλες κλάσεις/αντικείμενα. Να θυμάστε ότι μιλάμε μόνο για μια σύμβαση και δεν επιβάλλεται από την Python (με εξαίρεση το διπλό πρόθεμα κάτω παύλας).

Στατικές Μέθοδοι

Μια μέθοδος που έχει δηλωθεί ως **στατική**, λειτουργεί στο επίπεδο της κλάσης και όχι στο επίπεδο του στιγμιότυπου της. Επομένως, μια στατική μέθοδος δεν μπορεί να αναφέρεται σε συγκεκριμένο στιγμιότυπο της κλάσης (δηλαδή δεν μπορεί να αναφέρεται στο self).

Για να δηλώσουμε μια στατική μέθοδο σε μια κλάση, χρησιμοποιούμε τον διακοσμητή (decorator) `@staticmethod` όπως είδαμε και στο παραπάνω παράδειγμα. Στην συνέχεια, για να χρησιμοποιήσουμε μια στατική μέθοδο, αρκεί να αναφερθούμε σε ποια κλάση αυτή ανήκει, ή αν χρησιμοποιήσουμε στιγμιότυπο κάποιας κλάσης, τότε αυτό αγνοείται εκτός από την κλάση στην οποία αυτό ανήκει.

Κληρονομικότητα (Inheritance)

Ένα από τα σημαντικά οφέλη του αντικειμενοστρεφούς προγραμματισμού είναι η επαναχρησιμοποίηση (reuse) του κώδικα και ένας από τους τρόπους που επιτυγχάνεται είναι μέσω του μηχανισμού της κληρονομικότητας.Την υλοποίηση της κληρονομικότητας μπορούμε να τη φανταστούμε καλύτερα ως μια συγγένεια τύπου και υποτύπου (type and subtype) μεταξύ των κλάσεων. Ας υποθέσουμε ότι θέλετε να γράψετε ένα πρόγραμμα που καταγράφει τους καθηγητές και τους μαθητές ενός κολλεγίου. Έχουν κάποια κοινά χαρακτηριστικά - ονοματεπώνυμο, ηλικία και διεύθυνση κατοικίας. Επίσης, έχουν ιδιαίτερα χαρακτηριστικά - μισθοδοσία, μαθήματα και άδειες για τους καθηγητές, βαθμολογία και δίδακτρα για τους μαθητές.

Μπορείτε να δημιουργήσετε δύο ανεξάρτητες κλάσεις για κάθε τύπο και να τις επεξεργαστείτε, αλλά η προσθήκη ενός νέου κοινού χαρακτηριστικού θα απαιτούσε την προσθήκη του και στις δύο αυτές ανεξάρτητες κλάσεις. Καταντά δύσχρηστη αυτή η προσέγγιση.

Η κληρονομικότητα μοιάζει με το να χρησιμοποιούμε ως μέλος μιας κλάσης A ένα αντικείμενο b1 μιας άλλης κλάσης B. ΄Ετσι, θα μπορούσαμε να καλέσουμε ορισμένες συναρτήσεις της B στα αντικείμενα της A, μέσω του αντικειμένου b1 που υπάρχει σε αυτά. Η κληρονομικότητα μας επιτρέπει να εφαρμόζουμε απευθείας αυτές τις συναρτήσεις στα αντικείμενα της κλάσης B2. Οι άνθρωποι διακρίνουν τις αφηρημένες έννοιες σε δυο διαστάσεις: μέρος-από (part-of) ή σχέση έχει (has-a) και εξειδίκευση του ή είναι (is-a). Για παράδειγμα η Python είναι γλώσσα προγραμματισμού και έχει δυναμικούς τύπους. Η σχέση έχει συνήθως σημαίνει ότι πρέπει να χρησιμοποιήσουμε ένα αντικείμενο ως μέρος μιας κλάσης ενώ η σχέση είναι συνήθως επιτάσσει την χρήση κληρονομικότητας.

Μια καλύτερη λύση θα ήταν να δημιουργηθεί μια κοινή κλάση με το όνομα UniversityMember και κατόπιν οι κλάσεις καθηγητές και φοιτητές να κληρονομήσουν από αυτή, δηλαδή να γίνουν υποτύποι αυτού του τύπου (κλάση) και στη συνέχεια να προσθέσουμε ειδικά χαρακτηριστικά σε αυτούς τους υποτύπους.

Υπάρχουν πολλά πλεονεκτήματα στην προσέγγιση αυτή. Εάν προσθέσουμε/αλλάξουμε κάποια λειτουργία στην κλάση UniversityMember, αυτομάτως ενημερώνονται και οι υποτύποι. Για παράδειγμα, ένα πεδίο αριθμού ταυτότητας για τους φοιτητές και τους καθηγητές είναι εφικτό εύκολα με την προσθήκη του στην κλάση UniversityMember. Ωστόσο, οι αλλαγές σε έναν υποτύπο δεν επηρεάζουν τους άλλους υποτύπους.

Ένα άλλο πλεονέκτημα είναι ότι εάν μπορούμε να αναφερθούμε σε ένα αντικείμενο καθηγητή ή φοιτητή ως αντικείμενο UniversityMember, θα μας φανεί χρήσιμο σε καταστάσεις για τον υπολογισμό του συνολικού αριθμού των μελών του τμήματος του πανεπιστημίου. Η ιδιότητα αυτή αποκαλείται **πολυμορφισμός (polymorphism)** όπου ένας υποτύπος μπορεί να αντικατασταθεί σε κάθε περίπτωση που περιμένουμε ένα γονικό τύπο, δηλαδή το αντικείμενο μπορούμε να το χειριστούμε σαν μια υπόσταση της γονικής κλάσης.

Παρατηρήστε επίσης πώς επαναχρησιμοποιούμε τον κώδικα της γονικής κλάσης και δε χρειάζεται να τον επαναλαμβάνουμε, σε αντίθεση με την περίπτωση που θα χρησιμοποιούσαμε ανεξάρτητες κλάσεις.

- ✓ Η κλάση UniversityMember σε αυτή την περίπτωση ονομάζεται **βασική κλάση** (base class) ή υπερκλάση (superclass). Οι κλάσεις Professor και Student αποκαλούνται **παράγωγες κλάσεις** (derived classes) ή υποκλάσεις (subclasses).

➲ Ακολουθεί παράδειγμα :

inherit.py - C:/Python33/inherit.py

File Edit Format Run Options Windows Help

```
class UniversityMember:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def who(self):
        print('Name:{} Age:{}' .format(self.name, self.age))

class Professor(UniversityMember):
    def __init__(self, name, age, salary):
        UniversityMember. __init__(self, name, age)
        self.salary = salary

    def who(self):
        UniversityMember.who(self)
        print('Salary: {}'.format(self.salary))

class Student(UniversityMember):
    def __init__(self, name, age, marks):
        UniversityMember. __init__(self, name, age)
        self.marks = marks

    def who(self):
        UniversityMember.who(self)
        print('Marks:{}'.format(self.marks))

p = Professor ('Dr. M. Dasygenis', 35, 1500)
s = Student('Tonia Terzidou', 22, 8)

for member in (p,s):
    print('_____________')
    member.who()
```

Και τρέχοντάς το, προκύπτει :

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>

_____________
Name:Dr. M. Dasygenis Age:35
Salary: 1500

_____________
Name:Tonia Terzidou Age:22
Marks:8
>>>
```

Πως δουλεύει το παραπάνω πρόγραμμα :

Για να χρησιμοποιήσουμε κληρονομικότητα, μετά τον ορισμό του ονόματος της υπερκλάσης, προσδιορίζουμε τα ονόματα της σε μια πλειάδα. Στη συνέχεια, παρατηρούμε ότι η μέθοδος __init__ της υπερκλάσης καλείται ρητά με τη χρήση της μεταβλητής self για την αρχικοποίηση του τμήματος της υπερκλάσης που αντιστοιχεί στο αντικείμενο. Είναι σημαντικό να θυμάστε ***ότι η Python δεν καλεί αυτόματα τον κατασκευαστή της υπερκλάσης, πρέπει να κληθεί από εμάς.***

Παρατηρούμε επίσης ότι μπορούμε να καλέσουμε μεθόδους της υπερκλάσης με πρόθεμα στο όνομα της κλάσης, όταν καλούμε τη μέθοδο και στη συνέχεια περνά η μεταβλητή self μαζί με τυχόν εντολές. Προσέξτε πώς μπορούμε να μεταχειριστούμε τις υποστάσεις Professor ή Student ως απλές υποστάσεις της κλάσης UniversityMember, όταν χρησιμοποιούμε τη μέθοδο tell της κλάσης UniversityMember.

Επίσης, παρατηρήστε ότι καλείται η μέθοδος tell του υποτύπου και όχι η μέθοδος tell της κλάσης UniversityMember. Ένας τρόπος για να γίνει κατανοητό είναι ότι η Python αρχίζει να ψάχνει πάντα για μεθόδους στον πραγματικό τύπο, όπως και κάνει σε αυτή την περίπτωση. Αν δεν βρει τη μέθοδο, αρχίζει να αναζητά στις μεθόδους που ανήκουν στις υπερκλάσεις με τη σειρά που έχουν οριστεί στην πλειάδα.

Μια σημείωση σχετικά με την ορολογία - αν περισσότερες από μία κλάσεις παρατίθενται στην πλειάδα κληρονομικότητας, αποκαλείται τότε πολλαπλή κληρονομικότητα.

Στο συγκεκριμένο παράδειγμα, η κλάση Professor κληρονομεί από την υπερκλάση της UniversityMember. Για να προσδιορίσουμε μια κλάση από ποια κλάση κληρονομεί, βάζουμε ως όρισμα της κλάσης Professor τις κλάσεις από τις οποίες θέλουμε να κληρονομεί (UniversityMember). Στην συνέχεια, μπορούμε να καλέσουμε συναρτήσεις από την υπερκλάση της Professor και να συμπληρώσουμε στα αποτελέσματα τους την επιπρόσθετη λειτουργία που απαιτείται.

Με αυτό τον τρόπο λέπουμε πως οι υπερκλάσεις αποτελούν ένα επίπεδο αφαίρεσης ενώ αυτές που κληρονομούν από αυτές αφορούν πιο συγκεκριμένες έννοιες. Έτσι, μπορούμε να χρησιμοποιούμε υπερκλάσεις όταν θέλουμε να ομαδοποιούμε κοινή συμπεριφορά ανάμεσα σε κλάσεις και στην συνέχεια σε αυτές να υλοποιούμε μόνο ότι διαφέρει, επαναχρησιμοποιώντας έτσι όσο το δυνατόν περισσότερο κώδικα μπορούμε και επιμένοντας στην **αρχή DRY – Don't Repeat Yourself** που σημαίνει να μην γράφουμε τον ίδιο κώδικα ξανά και ξανά, πολλαπλασιάζοντας έτσι τα πιθανά λάθη που αυτός μπορεί να φέρει και κάνοντας πιο δύσκολη την συντήρησή τους (μια

διόρθωση σε ένα σημείο πρέπει να γίνει ξανά και σε όλα τα σημεία που πιθανώς υπάρχει ο ίδιος κώδικας).

Ειδικές Μέθοδοι

Μέσω κλάσεων, μπορούμε να υλοποιήσουμε μεθόδους που να αντιστοιχούν σε ειδικές περιπτώσεις (όπως αριθμητικούς τελεστές, σύγκριση αντικειμένων, διαγραφή τους κ.α.). Αυτές οι μέθοδοι έχουν ειδικά ονόματα που αρχίζουν και τελειώνουν σε __. Στην συνέχεια, παραθέτω ορισμένες βασικές τέτοιες μεθόδους. Αν κάποιος είναι εξοικειωμένος με άλλες γλώσσες προγραμματισμού, αυτός είναι ο τρόπος της Python για υπερφόρτωση τελεστών operator overloading.

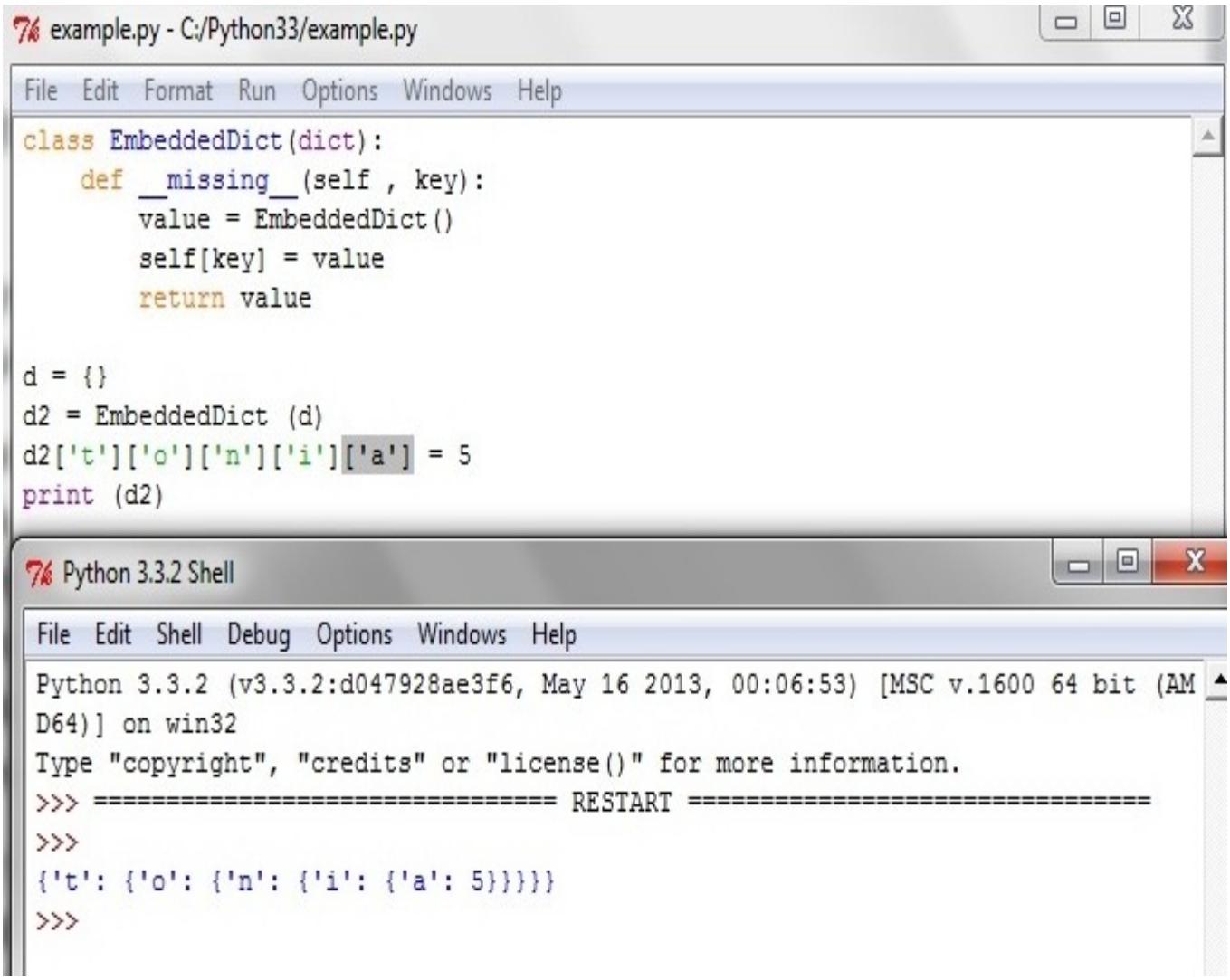
- **__init__(self, . . .)**: Αποτελεί μια μέθοδο που ήδη έχουμε δει. Είναι ο δημιουργός μιας κλάσης και καλείται όταν δημιουργούμε ένα καινούργιο αντικείμενο. Αν μια υπερκλάση (base class) έχει την μέθοδο __init__(), τότε αυτή πρέπει να υλοποιείται ητά και από όλες τις υποκλάσεις της. Επιπλέον, δεν μπορεί η μέθοδος __init__() να επιστρέφει κάποια τιμή
- **__del__(self):** Καλείται όταν ένα αντικείμενο καταστρέφεται. Ονομάζεται καταστροφέας (destructor). Αντίστοιχα με τον δημιουργό, και αυτή πρέπει να καλείται ητά αν μια υπερκλάση την υλοποιεί.
- **__repr__(self):** Καλείται μέσω της repr() συνάρτησης έτσι ώστε να δημιουργήσει την ‘επίσημη’ αναπαράσταση σε αλφαριθμητικό του αντικειμένου. Αν είναι δυνατό, πρέπει να επιστρέφεται μια έκφραση σε Python που να δημιουργεί ένα αντικείμενο με την ίδια τιμή. Συνήθως χρησιμοποιείται για λόγους debugging.
- **__str__(self):** Καλείται μέσω της str() συνάρτησης ή όποτε προσπαθούμε να τυπώσουμε απευθείας το αντικείμενο μέσω της print(). Διαφέρει από την __repr__() στο ότι δεν χρειάζεται να είναι έκφραση. Πάντρα πρέπει να είναι ένα αλφαριθμητικό.
- **__lt__(self, other), __le__(self, other), __eq__(self, other), __ne__(self, other), __ne__(self, other), __gt__(self, other), __ge__(self, other):** Αποτελούν τελεστές συγκρίσεις και αντιστοιχούν στο μικρότερο από (<), μικρότερο ή ίσο (<=), ίσο με (==), όχι ίσο με (! =), μεγαλύτερο από (>), μεγαλύτερο ή ίσο από (>=). Οι παραπάνω συναρτήσεις καλούν-ται μέσω των τελεστών που λέπουμε στις παρενθέσεις και είναι αρκετά χρήσιμες σε περιπτώσεις που για παράδειγμα μπορεί να θέλουμε να κάνουμε ταξινόμηση σε αντικείμενα που δημιουργήσαμε.

Επίσης, μπορούμε να υλοποιήσουμε μεθόδους που αντιστοιχούν σε μαθηματικές πράξεις, μερικές από τις οποίες βλέπουμε παρακάτω :

→ __add__(self, other): Πρόσθεση (+)
→ __sub__(self, other): Αφαίρεση ( )
→ __mul__(self, other): Πολλαπλασιασμός ( )
→ __truediv__(self, other): Διαίρεση (=)
→ __floordiv__(self, other): Διαίρεση ακεραίων (==)
→ __mod__(self, other): Υπόλοιπο (%)

Προσοχή πρέπει να δοθεί στο ότι πρέπει να αποφεύγουμε να δημιουργούμε δικές μας συναρτήσεις που αρχίζουν και τελειώνουν με __, παρά μόνο να υλοποιούμε ήδη υπάρχουσες, όπως αυτές που φαίνονται παραπάνω.

➔ Στο παράδειγμα που ακολουθεί, προσπαθούμε να προσπελαύσουμε ένα στοιχείο του λεξικού το οποίο αν δεν υπάρχει, το δημιουργούμε εκείνη τη στιγμή.



Αρχεία

Ήρθε η ώρα να μελετήσουμε ορισμένες βασικές λειτουργίες για τη διαχείριση του συστήματος αρχείων στον υπολογιστή μας.

Προσπέλαση

Προτού μπορέσουμε να επιτελέσουμε οποιαδήποτε λειτουργία πάνω σε ένα αρχείο, πρέπει να το ανοίξουμε, έτσι ώστε να ενημερώσουμε το λειτουργικό ότι πρόκειται να το χρησιμοποιήσουμε και να αναλάβει αυτό να το ανασύρει από τον σκληρό δίσκο. Για να ανοίξουμε ένα αρχείο, χρησιμοποιούμε την συνάρτηση open() με όρισμα το όνομα του αρχείου. Αυτή, στην συνέχεια μας επιστρέφει έναν περιγραφέα για το αρχείο, τον οποίο και χρησιμοποιούμε ώστε να το προσπελάσουμε. ΄Επειτα μπορούμε να εφαρμόσουμε, χρησιμοποιώντας αυτόν τον περιγραφέα, διάφορες λειτουργίες πάνω στο αρχείο.

Οι τρόποι αλληλεπίδρασης με αρχεία, δηλαδή οι τρόποι που μπορούμε να ανοίξουμε ένα αρχείο μέσω της open() φαίνονται στον πίνακα :

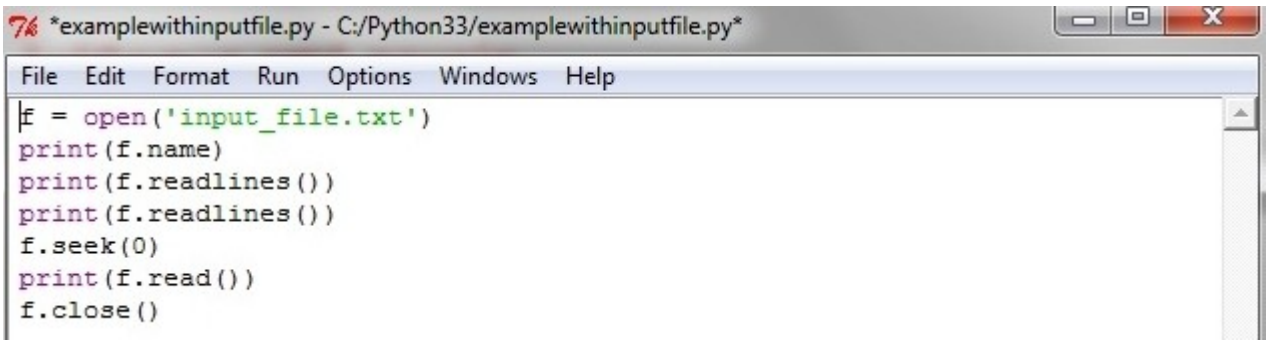
Τρόπος ανοίγματος	Χρήση
“r”	Ανάγνωση
“w”	Εγγραφή (Διαγραφή τυχών προηγούμενων περιεχομένων)
“a”	Προσθήκη (Διατήρηση τυχών προηγούμενων περιεχομένων)
“b”	Αρχείο δυαδικής μορφής
“+”	Προσθήκη δεδομένων στο τέλος του αρχείου ‘‘r+’’ είναι άνοιγμα αρχείου και για ανάγνωση/εγγραφή

Από τα ‘‘r’’, ‘‘w’’, ‘‘a’’ μόνο μια σημαία το πολύ μπορεί να επιλεχθεί. Αν δεν επιλεχθεί καμία, το αρχείο ανοίγει ως προεπιλογή μόνο για ανάγνωση (δηλαδή υπονοείται η σημαία ‘‘r’’.

Βασικές συναρτήσεις

Διάβασμα από αρχείο

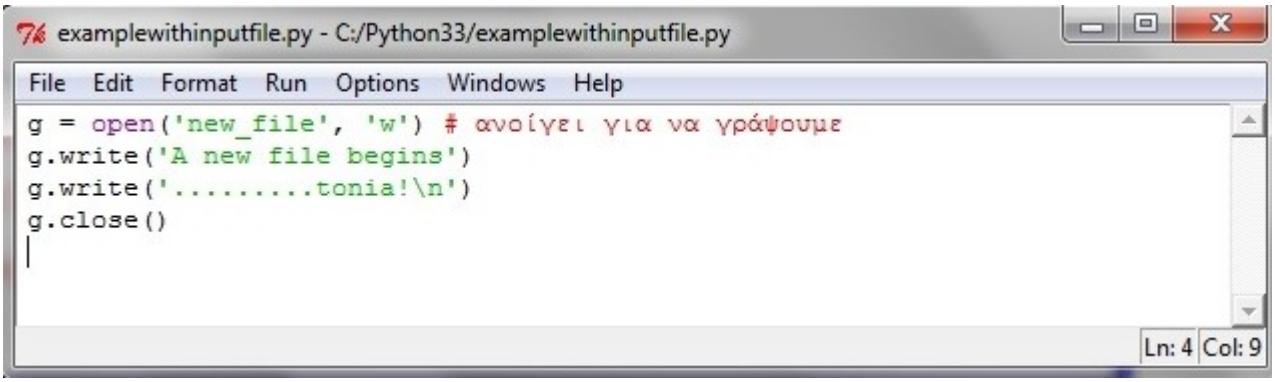
Η κύρια συνάρτηση για διάβασμα από αρχείο είναι η **read()**. Χωρίς όρισμα θα διαβάσει μέχρι το τέλος του αρχείου, αλλιώς για τον αριθμό bytes όπως καθορίζεται από το όρισμα της.



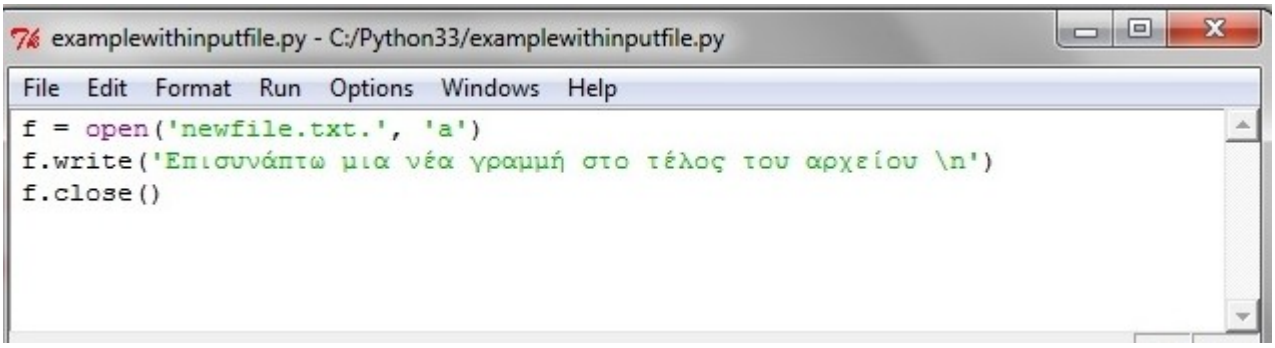
Η διαφορά που υπάρχει ανάμεσα στην readlines και στην read, είναι ότι η πρώτη διαβάζει μια μια τις γραμμές και τις επιστρέφει σε μια λίστα, ενώ η δεύτερη διαβάζει όλο το αρχείο και επιστρέφει ό,τι διαβάσει μέσα σε μια μεταβλητή. Τέλος, η συνάρτηση close() αναλαμβάνει να κλείσει το αρχείο και να απελευθερώσει έτσι πόρους του συστήματος.

Εγγραφή σε αρχείο

Αντίστοιχα με το διάβασμα από αρχείο, μπορούμε να κάνουμε την εγγραφή, μόνο που τώρα πρέπει να το καθορίσουμε, άζοντας το κατάλληλο όρισμα στην συνάρτηση open(). Προσοχή πρέπει να δοθεί μόνο στο γεγονός ότι με τον τρόπο που ανοίγουμε τώρα το αρχείο, οποιαδήποτε άλλα περιεχόμενα του διαγράφονται.

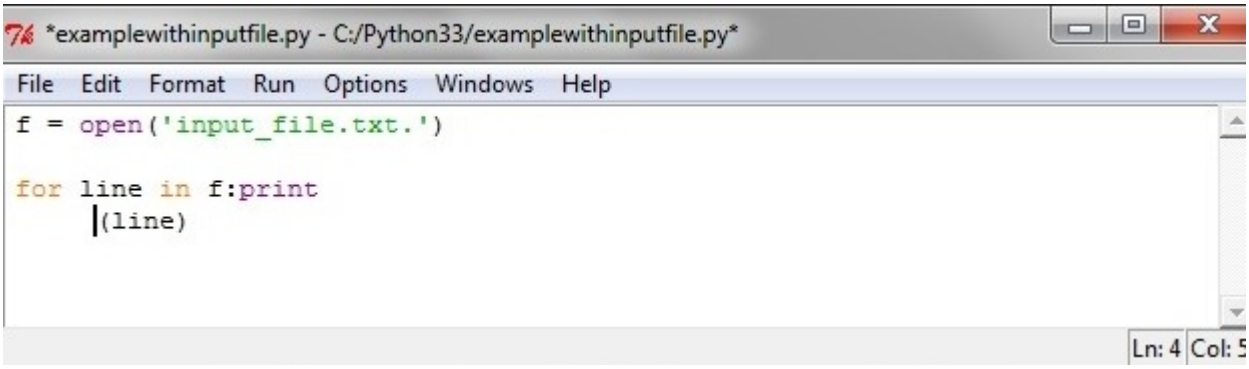


Αν θέλουμε να κρατήσουμε τα περιεχόμενα ενός αρχείου, και να προσθέσουμε κάποια άλλα, τότε πρέπει να ανοίξουμε το αρχείο με την ειδική σημαία για προσθήκη.



Διάτρεξη σε αρχεία

Μπορούμε να χρησιμοποιήσουμε βρόγχους for για να διατρέξουμε πάνω σε αρχεία (iterate), παίρνοντας για παράδειγμα μια μια τις γραμμές.



Αν δοκιμάσουμε το παραπάνω παράδειγμα, θα διαπιστώσουμε πως εκτυπώνεται μια κενή γραμμή ανάμεσα σε κάθε γραμμή του αρχείου. Αυτό συμβαίνει, γιατί στο τέλος της γραμμής κάθε αρχείου, υπονοείται ο χαρακτήρας αλλαγής γραμμής nn. ΄Ετσι, όταν την εκτυπώνουμε, εκτυπώνεται και αυτός ο χαρακτήρας. Ο επιπλέον λοιπόν χαρακτήρας αλλαγής γραμμής προέρχεται από την print καθώς όταν εκτυπώνουμε το αλφαριθμητικό, στο τέλος τυπώνουμε και έναν επιπρόσθετο χαρακτήρα αλλαγής γραμμής.

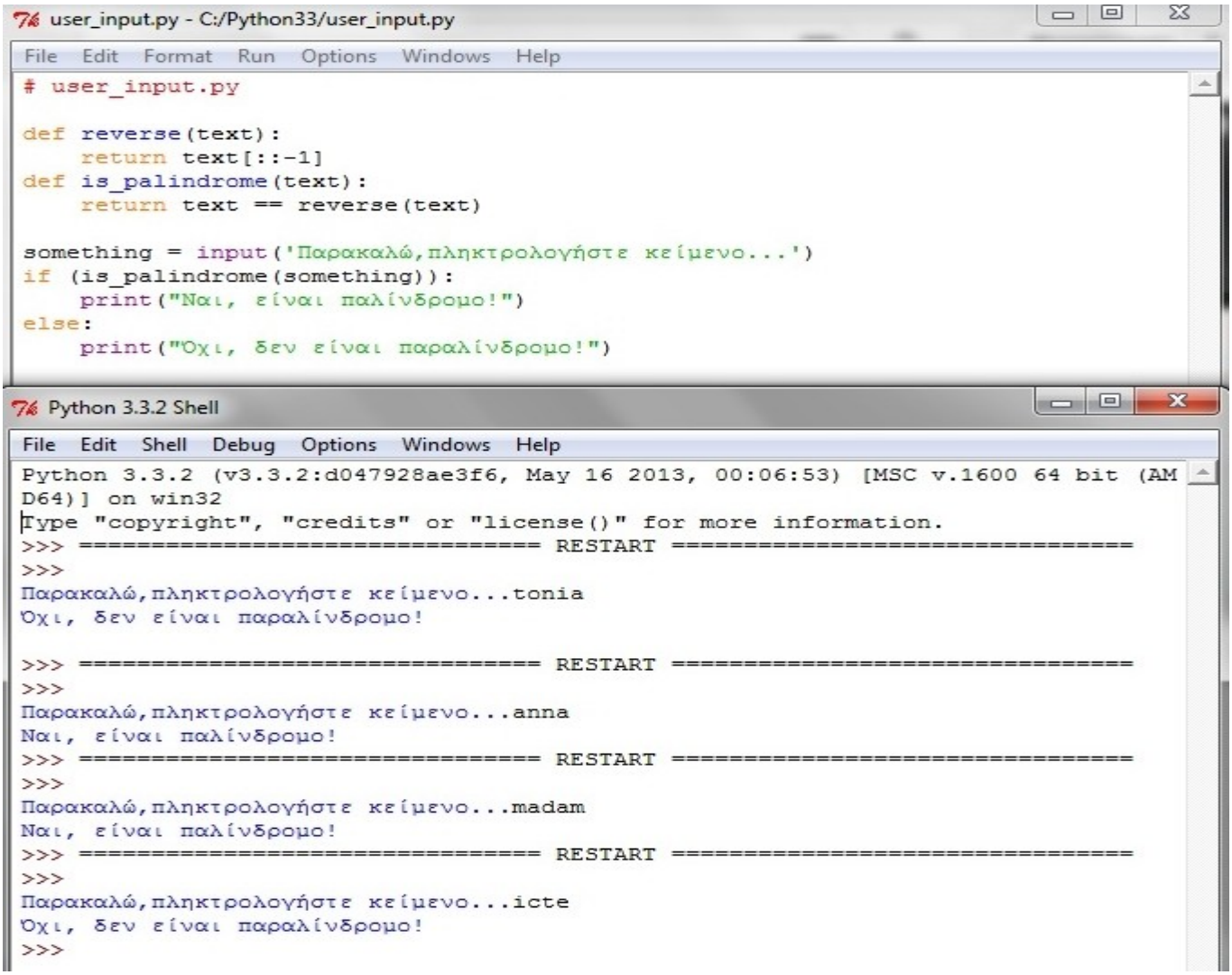
Είσοδος – έξοδος

Θα υπάρξουν καταστάσεις όπου το πρόγραμμά μας πρέπει να αλληλεπιδράσει με το χρήστη. Για παράδειγμα, θα θέλαμε να πάρουμε είσοδο από το χρήστη και μετά να τυπώσουμε πίσω μερικά αποτελέσματα. Μπορούμε να το επιτύχουμε αυτό χρησιμοποιώντας αντίστοιχα τις ***συναρτήσεις input() και print().***

Για έξοδο μπορούμε να χρησιμοποιήσουμε τις διάφορες μεθόδους της κλάσης str (string). Για παράδειγμα, μπορούμε να χρησιμοποιήσουμε τη μέθοδο rjust για να πάρουμε μια συμβολοσειρά, η οποία είναι δεξιά στοιχισμένη (right justified) σε ένα καθορισμένο εύρος. [help(str) για περισσότερες λεπτομέρειες].

Ένας ακόμα συνηθισμένος τύπος εισόδου/εξόδου είναι ο χειρισμός των αρχείων (files). Η ικανότητα να δημιουργείτε, διαβάζετε και να γράφετε αρχεία είναι βασική σε πολλά προγράμματα και θα την αναλύσουμε παρακάτω.

Είσοδος από το χρήστη



Πως δουλεύει το παραπάνω πρόγραμμα :

Χρησιμοποιούμε τον τεμαχισμό (κομμάτιασμα) για να αναστρέψουμε το κείμενο. Έχουμε ήδη δει πώς μπορούμε να κάνουμε κομμάτια από ακολουθίες, χρησιμοποιώντας τον κώδικα seq[a:b], αρχίζοντας από τη θέση a μέχρι τη θέση b. Μπορούμε επίσης να δώσουμε ένα τρίτο όρισμα το οποίο προσδιορίζει το βήμα με το οποίο γίνεται το κομμάτιασμα. Το προκαθορισμένο βήμα είναι το 1 εξαιτίας του οποίου επιστρέφει ένα συνεχές τμήμα του κειμένου. Δίνοντας ένα αρνητικό βήμα δηλ. -1, θα επιστρέψει το κείμενο ανάστροφα.

Η συνάρτηση input() παίρνει μια συμβολοσειρά σαν όρισμα και το παρουσιάζει στον χρήστη.Τότε περιμένει το χρήστη να τυπώσει κάτι και να πιέσει το return. Άπαξ και ο χρήστης έχει εισάγει κάτι, η συνάρτηση input() τότε θα επιστρέψει αυτό το κείμενο.

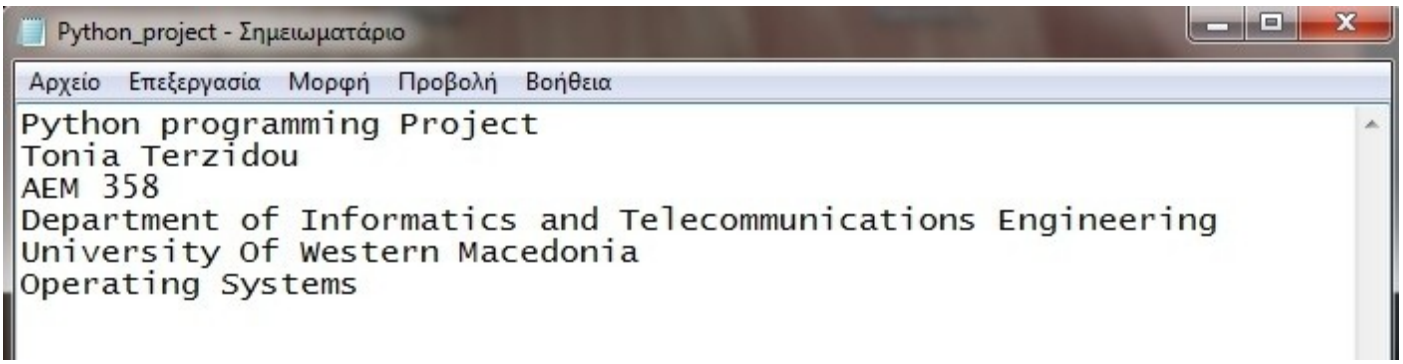
Παίρνουμε αυτό το κείμενο και το αναστρέφουμε. Εάν το αυθεντικό κείμενο και το ανεστραμμένο είναι ίσα, τότε το ***κείμενο είναι ένα παλίνδρομο.***

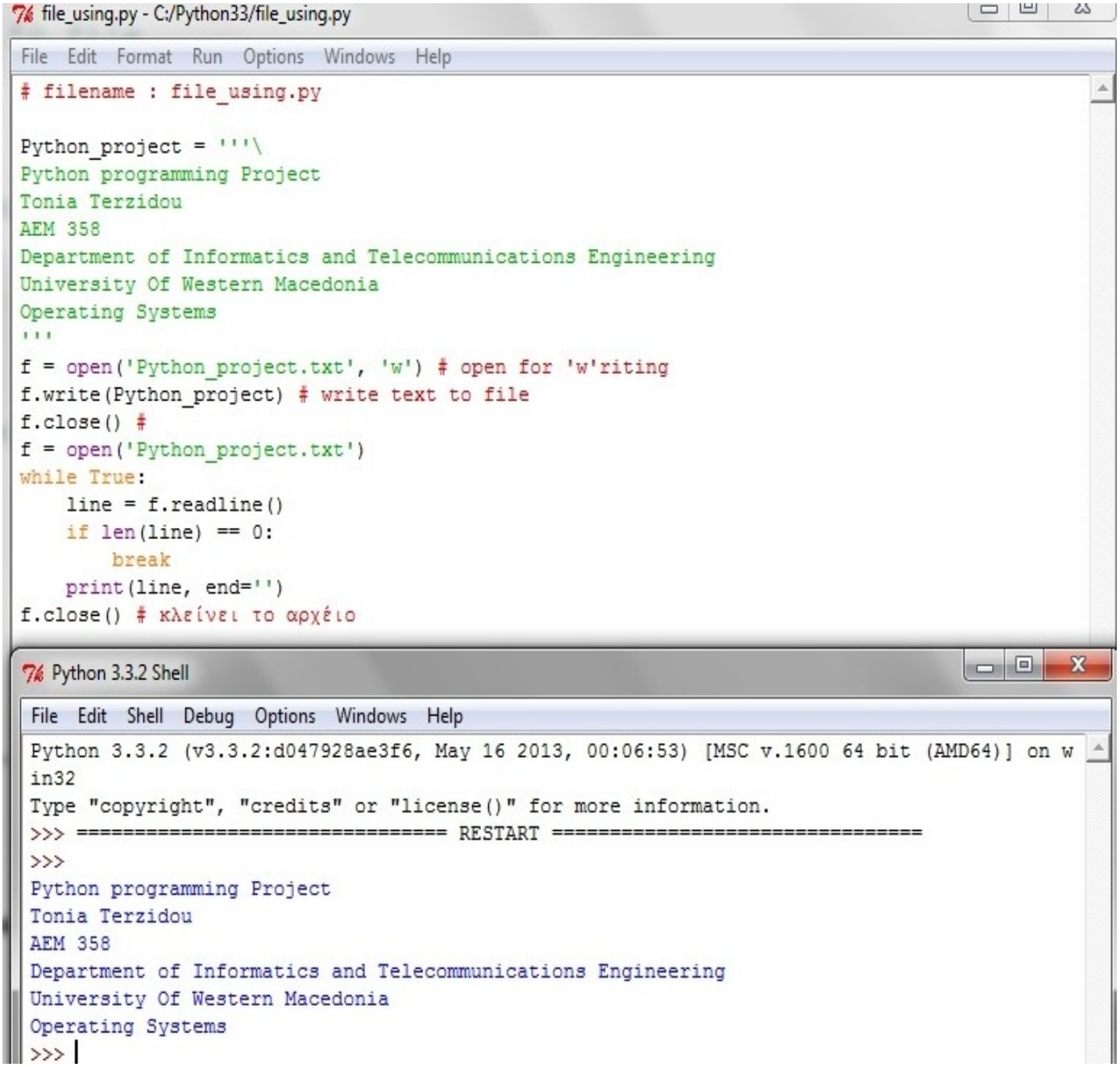
Αρχεία

Μπορείτε να ανοίξετε και να χρησιμοποιήσετε αρχεία για διάβασμα ή γράψιμο, δημιουργώντας ένα αντικείμενο της κλάσης file και χρησιμοποιώντας τις μεθόδους της, read, readline ή write κατάλληλα για να διαβάσει από ή να γράψει στο αρχείο. Η ικανότητα να διαβάζει ή να γράφει στο αρχείο εξαρτάται από τον τρόπο (mode) που έχετε καθορίσει για το άνοιγμα του αρχείου. Τότε τελικά, όταν τελειώσετε με το αρχείο, καλείτε τη μέθοδο close, να πει στην Python ότι τελειώσαμε με τη χρήση του αρχείου.

➲ Ακολουθεί παράδειγμα :

Αρχείο Python_project.txt





Πως δουλεύει το παραπάνω πρόγραμμα :

Αρχικά ανοίγετε ένα αρχείο χρησιμοποιώντας την ενσωματωμένη συνάρτηση open καθορίζοντας την ονομασία του αρχείου και τον τρόπο με τον οποίο θέλουμε να ανοίγει το αρχείο. Ο τρόπος μπορεί να είναι με διάβασμα ('r', read mode), με γράψιμο ('w', write mode) ή με πρόσθεση ('a', append mode). Μπορούμε επίσης να χειριστούμε ένα αρχείο κειμένου ('t', text file) ή ένα δυαδικό αρχείο ('b', binary file). Στην πραγματικότητα υπάρχουν πάρα πολλοί τρόποι διαθέσιμοι και η help(open) θα σας δώσει περισσότερες λεπτομέρειες γι αυτούς. Από προεπιλογή η open() θεωρεί το αρχείο ως αρχείο κειμένου ('t'ext file) και το ανοίγει με τον τύπο 'r'ead.

Σε αυτό το παράδειγμα, αρχικά ανοίγουμε το αρχείο σε εγγραφή και χρησιμοποιούμε τη μέθοδο write του αντικειμένου του αρχείου, για να γράψουμε στο

αρχείο και τότε τελικά κλείνουμε (close) το αρχείο. Κατόπιν ανοίγουμε το ίδιο αρχείο πάλι για ανάγνωση. Δε χρειάζεται να καθορίσουμε έναν τύπο, γιατί η 'ανάγνωση' είναι ο προκαθορισμένος τρόπος. Διαβάζουμε κάθε γραμμή του αρχείου χρησιμοποιώντας τη μέθοδο readline σε βρόχο. Αυτή η μέθοδος επιστρέφει μια ολοκληρωμένη γραμμή περιλαμβάνοντας το χαρακτήρα νέας γραμμής (newline character) στο τέλος της γραμμής. Όταν μια άδεια συμβολοσειρά επιστρέφεται, σημαίνει ότι έχουμε φθάσει στο τέλος του αρχείου και 'ξεφεύγουμε' (break) από το βρόχο.

Από προεπιλογή η συνάρτηση print() τυπώνει το κείμενο καθώς και μια αυτόματη νέα γραμμή (newline) στην οθόνη. Καταστέλλουμε τη νέα γραμμή καθορίζοντας end='', διότι η γραμμή που διαβάζεται από το αρχείο ήδη τελειώνει με ένα χαρακτήρα νέας γραμμής. Τότε, τελικά κλείνουμε (close) το αρχείο.Ελέγχωντας τα περιεχόμενα του αρχείου Python_Project.txt (το οποίο έχω παραθέσει παραπάνω),επιβεβαιώνεται ότι το πρόγραμμα έχει πραγματικά γραφτεί και διαβαστεί από αυτό το αρχείο.

Εγγραφή αντικειμένων σε αρχεία (σειριοποίηση)

Pickle

Η Python παρέχει ένα πρότυπο άρθρωμα που ονομάζεται ***pickle*** και χρησιμοποιώντας το μπορείτε να αποθηκεύετε οποιοδήποτε αντικείμενο της Python σε ένα αρχείο και αργότερα να το παίρνετε πίσω. Αυτό ονομάζεται **επίμονη αποθήκευση του αντικειμένου (persistently).**

Η **σειριοποίηση (serialization) ενός αντικειμένου**, είναι η διαδικασία μετατροπής του αντικειμένου σε μια σειρά από bits με σκοπό την εγγραφή του σε κάποιο μέσο αποθήκευση (συνήθως αρχεία) ή την μεταφορά του μέσω δικτύου. Η σειριοποίηση πρέπει να γίνεται με τέτοιο τρόπο ώστε να καθίσταται δυνατή η ανάκτηση του αντικειμένου στην αρχική του μορφή και με τις ίδιες ιδιότητες που το χαρακτήριζαν. Η σειριοποίηση θα σας χρειαστεί εφόσον θέλετε ένα πρόγραμμα που έχετε φτιάξει να αποθηκεύσει ορισμένα δεδομένα από τη μνήμη στο σκληρό (σε συνήθως ακαταλαβίστικη μορφή από τον άνθρωπο) και στην συνέχεια θέλετε να τα ανακτήσετε.

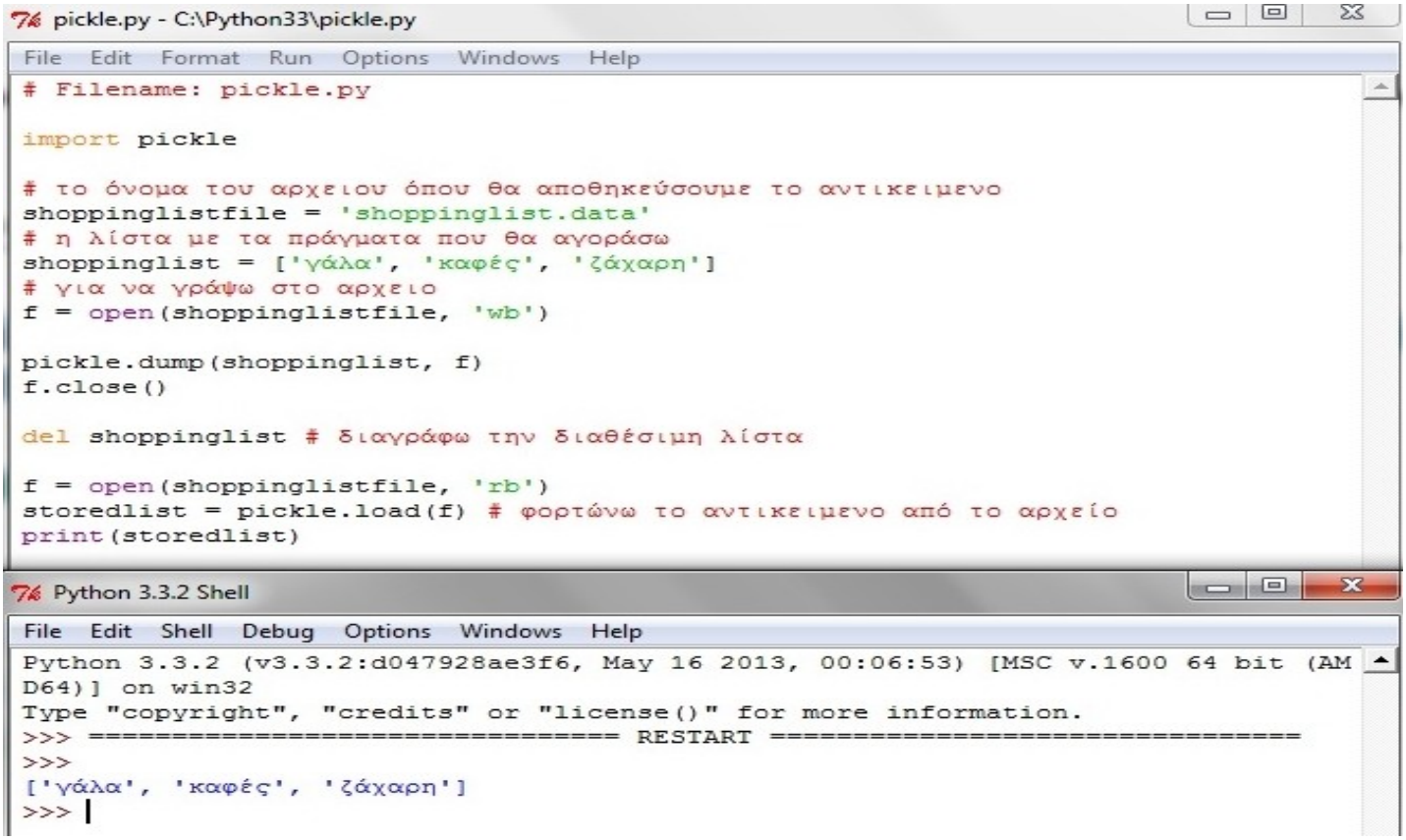
Για τον σκοπό αυτό, η Python χρησιμοποιεί το άρθρωμα (module) pickle. Το pickle μπορεί να χρησιμοποιήσει διάφορους τρόπους για να αποθηκεύσει τα δεδομένα σε αρχείο.

Το pickle μπορεί να χρησιμοποιήσει διάφορους τρόπους για να αποθηκεύσει τα δεδομένα σε αρχείο.

- Πρωτόκολλο έκδοσης 0: η αρχική έκδοση του πρωτοκόλλου που τα δεδομένα αποθηκεύονται σε αναγνώσιμη μορφή από τον άνθρωπο
- Πρωτόκολλο έκδοσης 1: η παλιά έκδοση που αποθήκευε τα δεδομένα σε δυαδική μορφή και δεν χρησιμοποιείται πια
- Πρωτόκολλο έκδοσης 2: η έκδοση που χρησιμοποιόταν στην έκδοση 2 της Python
- Πρωτόκολλο έκδοσης 3: η τρέχουσα έκδοση

Συνήθως χρησιμοποιείται η πιο καινούργια έκδοση, παραλείποντας το αντίστοιχο όρισμα. ΄Οταν ανακτάται ένα αντικείμενο από κάποιο αρχείο, επιλέγεται αυτόματα η σωστή έκδοση του πρωτοκόλλου.

➔ Ακολουθεί ένα παράδειγμα με pickle :



Πώς δουλεύει το παραπάνω πρόγραμμα :

Για να αποθηκεύσουμε ένα αντικείμενο σε ένα αρχείο, πρέπει αρχικά να ανοίξουμε (open) το αρχείο με τρόπο 'w'rite 'b'inary και μετά να καλέσουμε τη συνάρτηση dump του αρθρώματος pickle. Αυτή η διαδικασία ονομάζεται **pickling.**

Κατόπιν, ανακτούμε το αντικείμενο χρησιμοποιώντας τη συνάρτηση load του αρθρώματος pickle, η οποία επιστρέφει το αντικείμενο. Αυτή η διαδικασία ονομάζεται **unpickling**.

Φάκελοι

Πολλές φορές, κάνουμε διάφορα πράγματα για τα οποία χρειαζόμαστε γνώση για κάποιους φακέλους του συστήματος. Η python μπορεί να μας βοηθήσει πολύ εύκολα σε αυτό, αναλαμβάνοντας από μόνη της ο κώδικας της να τρέχει ανεξαρτήτως πλατφόρμας. Το μόνο που έχουμε να κάνουμε εμείς είναι να εισάγουμε την ***βιβλιοθήκη os*** που περιέχει τις συγκεκριμένες συναρτήσεις (που προφανώς υλοποιούνται διαφορετικά ανάλογα το σύστημα) και στην συνέχεια με απλές συναρτήσεις να πάρουμε τις απαραίτητες πληροφορίες.

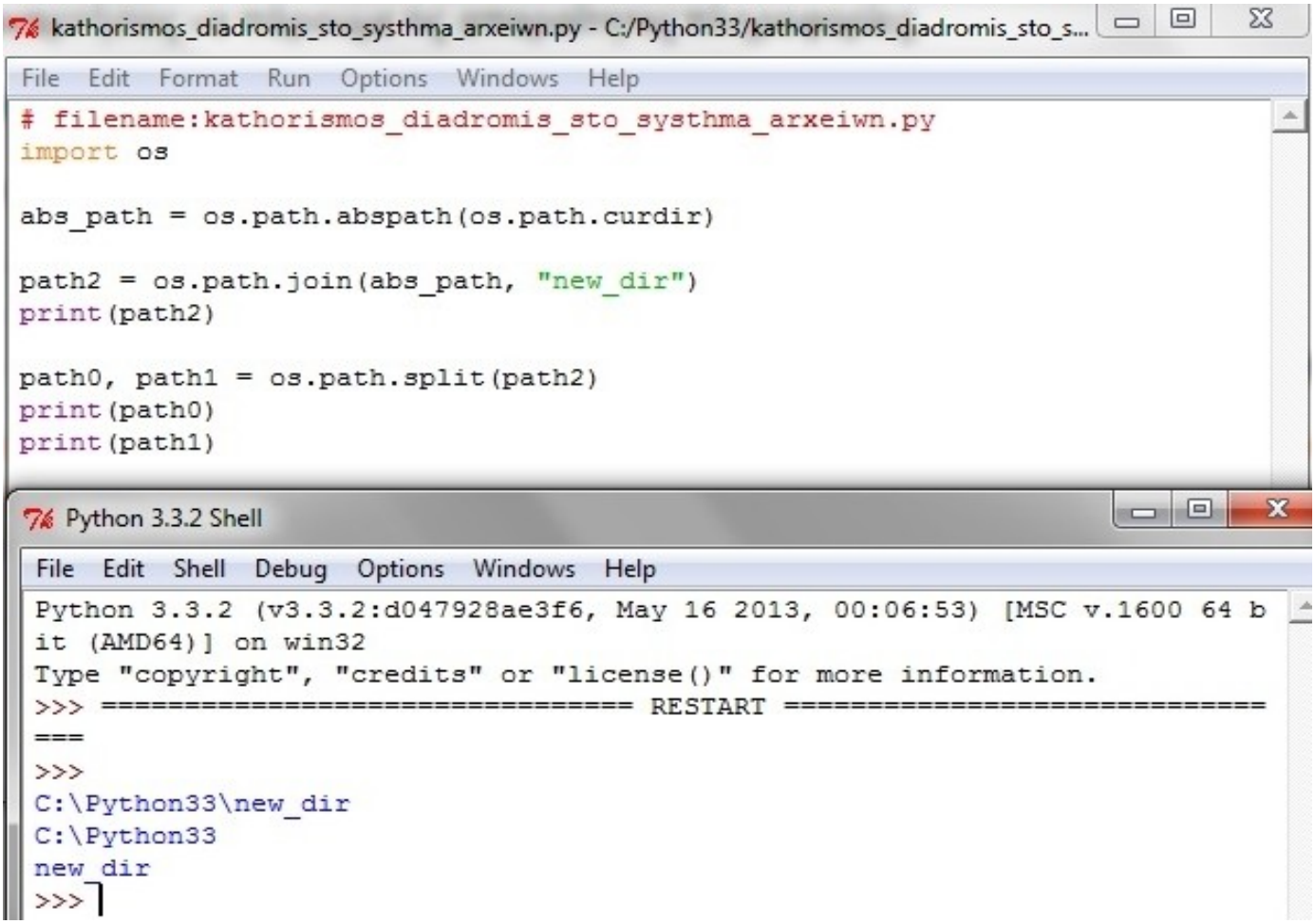
Ανάκτηση Πληροφοριών

Οι κύριες συναρτήσεις που χρειάζονται για αυτόν τον σκοπό είναι:

- ✓ `getcwd()`: Επιστρέφει το όνομα του τρέχοντος φακέλου.
- ✓ `listdir()`: Παραθέτει τα αρχεία του τρέχοντος φακέλου.
- ✓ `chdir(path)`: Αλλαγή φακέλου σε αυτόν που καταδεικνύεται από το path.

Το άρθρωμα (module) που χρησιμοποιούμε για την διαχείριση αρχείων και φακέλων είναι κυρίως το os. Οι συναρτήσεις που μας παρέχει είναι στην συντριπτική τους πλειοψηφία cross-platform, με εξαίρεση ορισμένες εξιδικευμένες περιπτώσεις.

➔ Στο παράδειγμα που ακολουθεί βλέπουμε βασικές λειτουργίες για τον καθορισμό μιας διαδρομής στο σύστημα αρχείων. Οι συναρτήσεις `os.path.split()` και `os.path.join()` πρέπει να χρησιμοποιούνται αν θέλουμε να έχουμε cross-platform εφαρμογή, γιατί υπάρχουν διαφορές στο πως αναπαριστώνται οι διαδρομές στις διάφορες πλατφόρμες.



Δημιουργία Φακέλων

Για την δημιουργία φακέλων υπάρχουν δυο βασικές συναρτήσεις :

- ✓ mkdir(path): Δημιουργία φακέλου με το όνομα path.
- ✓ makedirs(path): Δημιουργία του φακέλου που ορίζεται από το path καθώς και όλων των άλλων φακέλων που μπορεί να χρειάζεται να δημιουργηθούν, μέχρι να φθάσουμε σε αυτόν.

➔ Ακολουθεί ένα παράδειγμα :

kathorismos_diadromis_sto_systhma_arxeiwn.py - C:/Python33/kathorismos_diadromis_sto_systhma_...

File Edit Format Run Options Windows Help

```
import os

if os.path.isdir('new_dir'):
    os.rmdir('new_dir')
os.makedirs('new_dir')
print(os.listdir('.'))
os.chdir('new_dir')
```

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
['1stexample.py', '2ndexample.py', '2ndexamplewithsets.py', 'akolouyhiaexample.p
y', 'antistrofi.py', 'antostrofi.py', 'axa.py', 'backup_1ststep.py', 'break.py',
 'class.py', 'continue.py', 'deepcopy.py', 'del.py', 'diakrinousa.py', 'dictiona
ry.py', 'dir.py', 'elj.py', 'example.py', 'example1.py', 'example2.py', 'example
3.py', 'examplealians.py', 'examplelist.py', 'examplelist2.py', 'examplepython.p
y', 'EXAMPLES.py', 'examplewithinputfile.py', 'examplewithobject.py', 'examplewi
thsets.py', 'examplewith_reference.py', 'expression.py', 'fault.py', 'file_using
.py', 'foo.py', 'foo2.py', 'foo3.py', 'forexample.py', 'format.py', 'function.py
', 'function_default.py', 'function_DocStrings.py', 'function_globall.py', 'func
tion_key.py', 'function_local.py', 'function_nonlocal.py', 'function_parameter.p
y', 'function_parameter2.py', 'function_return.py', 'helloworld.py', 'ifelsesynt
hiki.py', 'ifexample.py', 'increasevalue.py', 'inherit.py', 'input_file.py', 'in
put_file.txt.txt', 'iselsesynthiki.py', 'kathorismos_diadromis_sto_systhma_arxei
wn.py', 'kommatiantistrofi.py', 'lambda.py', 'leksiko.py', 'Lexample.py', 'list.
py', 'listexample.py', 'list_.py', 'migadikoi.py', 'module_demo.py', 'myexample.
py', 'myfirstmodule.py', 'myfirstmodule_demo.py', 'mymodule_demo.py', 'mymodule_
demo2.py', 'myprogram.py', 'name.py', 'newfile.txt', 'new_dir', 'new_file', 'non
e.py', 'OKO.py', 'OO.py', 'paradeigma.py', 'pickle.py', 'pleiades.py', 'praxeis.
py', 'print.py', 'program1.py', 'Python_project.txt', 'range.py', 'range2.py', '
range3.py', 'robot.py', 'sequenceexample.py', 'sets.py', 'shoplist.data', 'shopp
inglist.data', 'shoppinglist.py', 'simpleexample.py', 'simpleexample1.py', 'simp
leexamples.py', 'someexamples.py', 'stoiva.py', 'strings.py', 'strings2.py', 'st
r_methods.py', 'synartisi.py', 'tonia python.py', 'trianglearea.py', 'triangle_r
eturn.py', 'tringlearea.py', 'user_input.py', 'using_dictionary.py', 'using_name
,py.py', 'using_sys,py.py', 'VarArgs,py.py', 'variable.py', 'whileexample.py', '
```

Εξαιρέσεις

Οι εξαιρέσεις μας επιτρέπουν να διαχωρίσουμε τις περιπτώσεις όπου καλούμαστε να διαχειριστούμε μια «εξαιρετική» περίπτωση ενός συμβάντος(πχ λείπει κάποιο αρχείο) από την λογική του προγράμματός μας.

Για παράδειγμα, τι συμβαίνει εάν πρόκειται να διαβάσετε ένα αρχείο και το αρχείο δεν υπάρχει; Ή τι συμβαίνει εάν το διαγράψατε κατά λάθος όταν το πρόγραμμα έτρεχε; Τέτοιες καταστάσεις χειρίζονται χρησιμοποιώντας τις εξαιρέσεις.

Παρομοίως, τι συμβαίνει εάν το πρόγραμμά σας είχε μερικές άκυρες εντολές; Αυτό το χειρίζεται η Python η οποία σηκώνει τα χέρια της και σας λέει ότι υπάρχει ένα σφάλμα.

Ο **χειρισμός εξαιρέσεων** είναι μια κατασκευή η οποία μας επιτρέπει να χειριστούμε ειδικές συνθήκες που αλλάζουν την φυσιολογική ροή του προγράμματος.

΄Οπως βλέπουμε και από τον παραπάνω ορισμό, τις εξαιρέσεις τις χρησιμοποιούμε σε ειδικές περιπτώσεις που η ροή του προγράμματος αλλάζει από την ‘φυσιολογική’ λόγω του ότι έχει συμβεί ένα εξαιρετικό συμβάν, που συνήθως δεν γίνεται. ΄Ενα παράδειγμα θα μπορούσε να είναι η περίπτωση όπου δεν μπορέσαμε να ανοίξουμε ένα αρχείο και να διαβάσουμε τα περιεχόμενα του. Γενικά οι εξαιρέσεις δημιουργούνται όταν κάτι πάει στραβά, ή όταν κάτι παρεκκλίνει σπάνια από μια συγκεκριμένη ροή (πχ υπάρχει μια πολύ πολύ μικρή πιθανότητα να συμβεί ένα γεγονός).

Ο γενικότερος μηχανισμός που λειτουργούν οι εξαιρέσεις είναι:

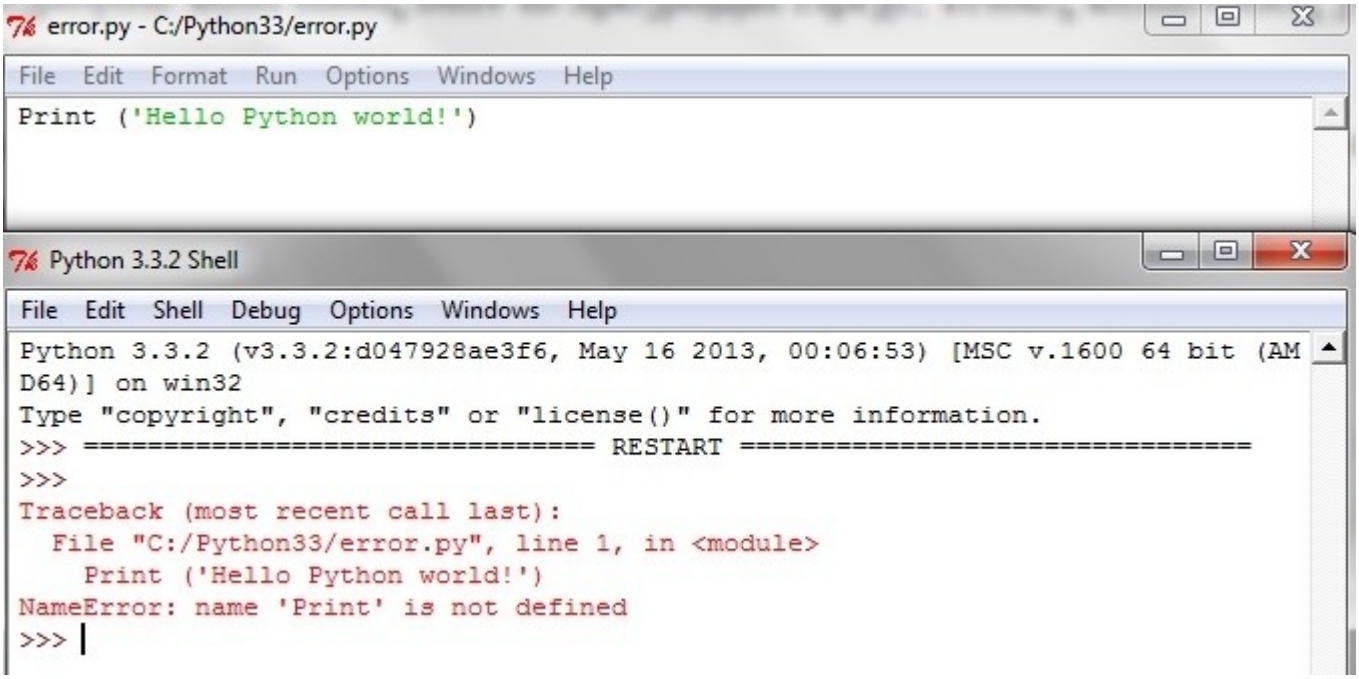
1. Αν κάτι πάει στραβά η Python εγείρει μια εξαίρεση.
2. Αναγνωρίζουμε την κλάση της εξαίρεσης (πχ NameError, TypeError, IndexError).
3. Εκτελούμε ειδικό κώδικα για αυτή την περίπτωση.
4. Αν δεν χειριστούμε μια εξαίρεση που εγείρεται, τότε το πρόγραμμα μας σταματάει την εκτέλεση του.
5. Εξαιρέσεις που δεν έχουν χειριστεί, εμφανίζονται στο traceback, όπου και μας ενημερώνει τι ακριβώς συνέβη.

Αξίζει να σημειώσουμε πως όταν κάποια ενδεχόμενα είναι σχεδόν ισοπίθανο να συμβούν, ή δεν αποτελεί ιδιάζων γεγονός η πραγματοποίηση κάποιου από αυτά, τότε είναι καλύτερα να χρησιμοποιούμε την δομή if. . . else . . . , εφόσον αυτό είναι

δυνατό. Αυτό, γιατί αν συμβεί εξαίρεση, τότε ο χειρισμός της είναι πολύ πιο αργός από το να πηγαίναμε στο κομμάτι του else. Από την άλλη, αν πρόκειται να χειριστούμε σφάλματα τότε και ο κώδικας μας γίνεται πιο ‘καθαρός’ αν χρησιμοποιούμε εξαιρέσεις, και πιο γρήγορος όταν μένει στην προκαθορισμένη του ροή και δεν υπάρχει ανάγκη εκτέλεσης των δηλώσεων στα μέρη του except.

Σφάλματα

Έστω μια απλή κλήση της συνάρτησης print. Τι συμβαίνει αν γράψω Print αντί για το σωστό print; Παρατηρήστε το κεφαλαίο P αντί για p. Σε αυτή την περίπτωση η Python αναδεικνύει (raises) ένα συντακτικό σφάλμα (syntax error).



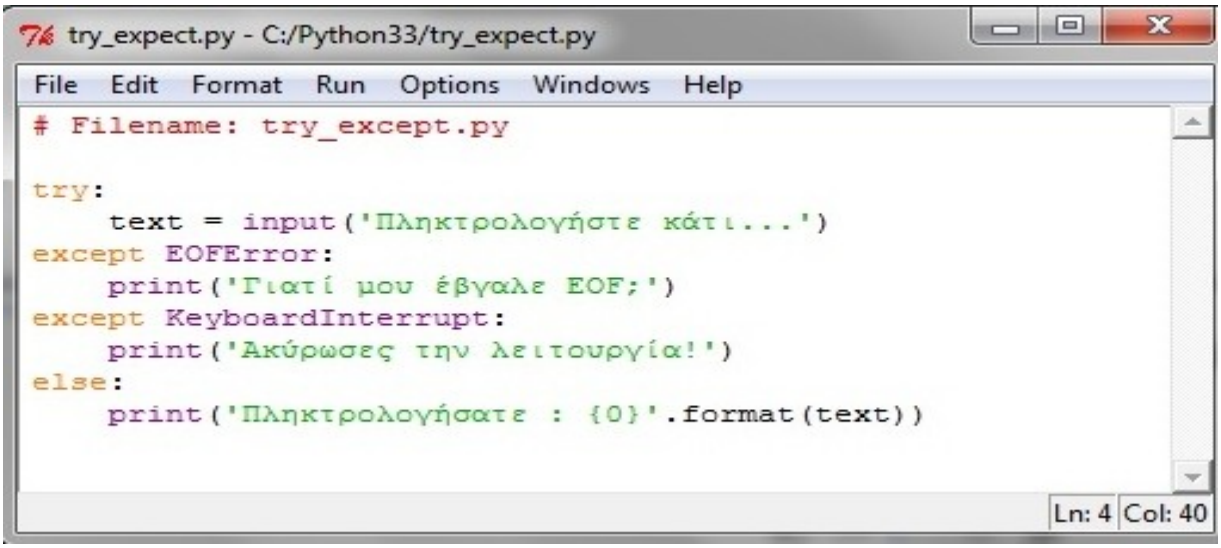
Παρατηρήστε ότι αναδεικνύεται ένα NameError καθώς επίσης τυπώνεται και η θέση όπου ανιχνεύεται το σφάλμα. Αυτό είναι που κάνει ο χειριστής σφάλματος (error handler) για αυτό το σφάλμα.

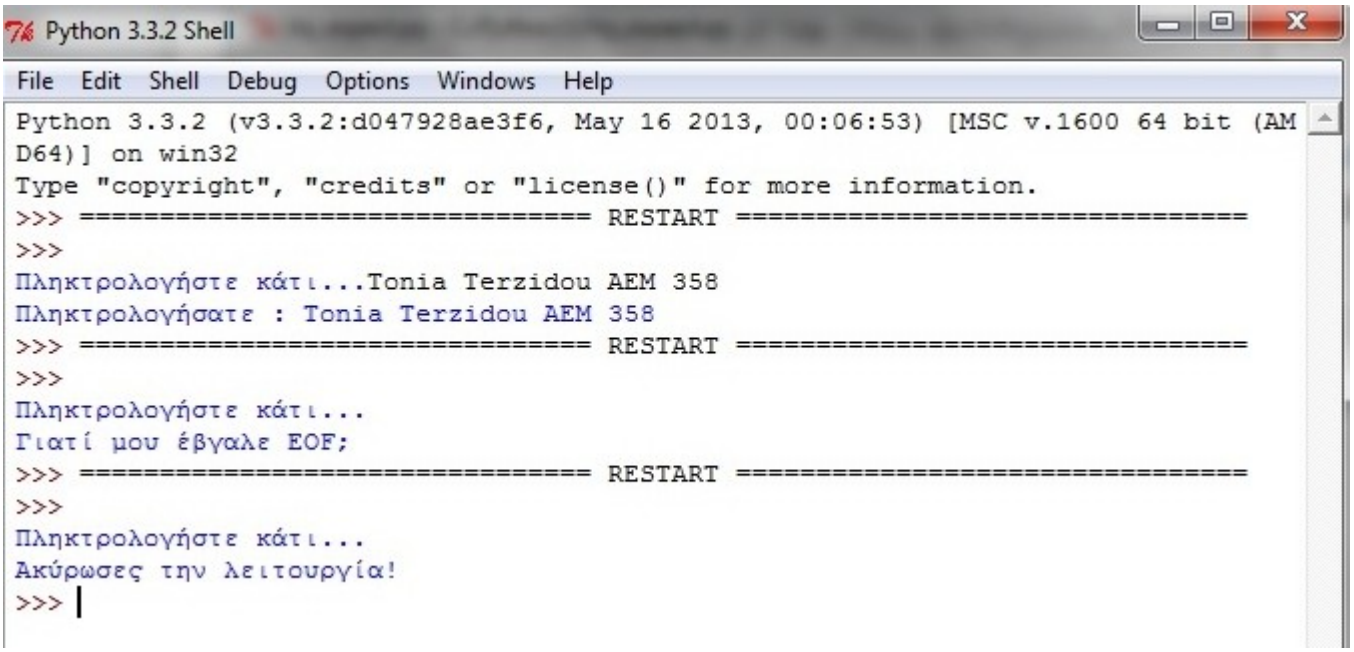
- Θα δοκιμάσουμε να διαβάσουμε είσοδο από το χρήστη. Πιέζοντας ctrl-d και προκύπτει :

```
s = input('Πληκτρολογήστε κάτι… --> ')

Πληκτρολογήστε κάτι -->

  Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
  s = input('Enter something --> ')
  EOFError: EOF when reading a line
```

Η Python αναδεικνύει ένα σφάλμα, που ονομάζεται ***EOFError***, που βασικά σημαίνει ότι βρήκε ένα σύμβολο end of file (που αντιπροσωπεύεται από το ctrl-d), όταν δεν περιμένει να το δει.

Μπορούμε να χειριστούμε τις εξαιρέσεις χρησιμοποιώντας την **εντολή try..except.** Βασικά, τοποθετούμε όλες τις συνήθεις εντολές μέσα στην πλοκάδα try και όλους τους χειριστές σφαλμάτων στην πλοκάδα except.





Παρατήρηση : στην εκτέλεση στην 2η περίπτωση στην προτροπή να πληκτρολογήσω κάτι πάτησα ctrl –d, ενώ στην 3η περίπτωση ctrl –c.

Πώς δουλεύει το παραπάνω πρόγραμμα :

Τοποθετούμε όλες τις εντολές, που ίσως αναδεικνύουν εξαιρέσεις/σφάλματα μέσα στην πλοκάδα try και μετά τοποθετούμε τους χειριστές για τα κατάλληλα σφάλματα/εξαιρέσεις στην πρόταση/πλοκάδα except. Η πρόταση except μπορεί να

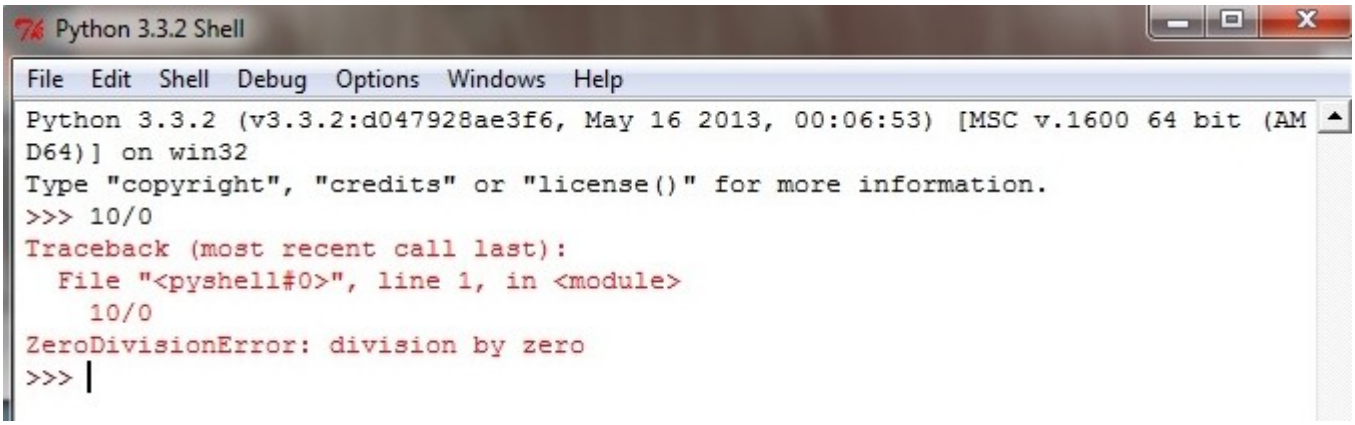
χειριστεί ένα και μόνο καθορισμένο σφάλμα ή εξαίρεση, ή μια λίστα σφαλμάτων/εξαιρέσεων μέσα σε παρένθεση. Εάν δεν παρέχονται καθόλου ονομασίες σφαλμάτων ή εξαιρέσεων, τότε η πρόταση except θα χειριστεί όλες τις εξαιρέσεις και τα σφάλματα.

Σημειώστε ότι πρέπει να υπάρχει τουλάχιστον μια πρόταση except συνδεδεμένη με κάθε πρόταση try. Διαφορετικά για ποιο λόγο να έχετε μια πλοκάδα try; Εάν οποιοδήποτε σφάλμα ή εξαίρεση δεν χειρίζεται, τότε καλείται ο προκαθορισμένος χειριστής της Python, ο οποίος σταματάει την εκτέλεση του προγράμματος και τυπώνει ένα μήνυμα σφάλματος.

Μπορείτε επίσης να έχετε μια πρόταση else συνδεδεμένη με μια πλοκάδα try..except. Η πρόταση else εκτελείται εάν δεν συμβαίνει καμμία εξαίρεση.

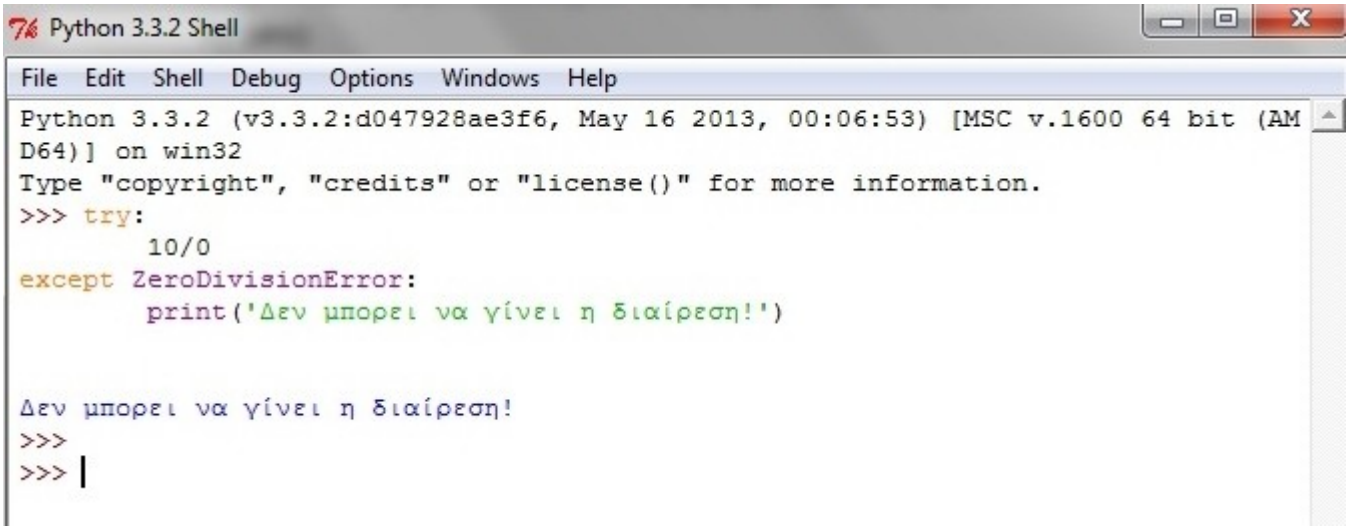
Διάφορα είδη Εξαιρέσεων

Υπάρχουν διάφορα είδη εξαιρέσεων που μπορούμε να συναντήσουμε, καθώς και να φτιάξουμε τα δικά μας. ΄Ενα απλό παράδειγμα εξαίρεσης αποτελεί η διαίρεση με το μηδέν.



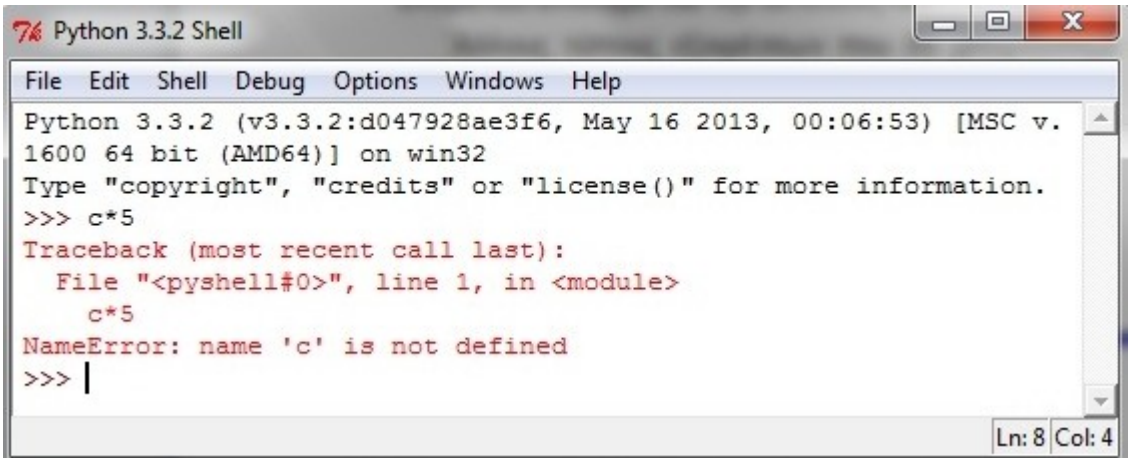
Κάνοντας διαίρεση με το μηδέν στο διαδραστικό κέλυφος (shell) της Python, μπορούμε να δούμε τι ακριβώς συμβαίνει μόλις προκληθεί μια εξαίρεση και δεν την χειριστούμε. Στο παράδειγμα μας, φαίνεται σε ποιο αρχείο συνέβη η εξαίρεση (<pyshell#0>, αφού γράφαμε στο περιβάλλον του διερμηνευτή), σε ποια γραμμή (γραμμή 1), η κλάση της εξαίρεσης (ZeroDivisionError, που σημαίνει διαίρεση με το μηδέν) καθώς και ένα μήνυμα σφάλματος (int division or modulo by zero).

Αφού έχουμε αναγνωρίσει λοιπόν την εξαίρεση που συνέβη, μπορούμε πλέον να την χειριστούμε.



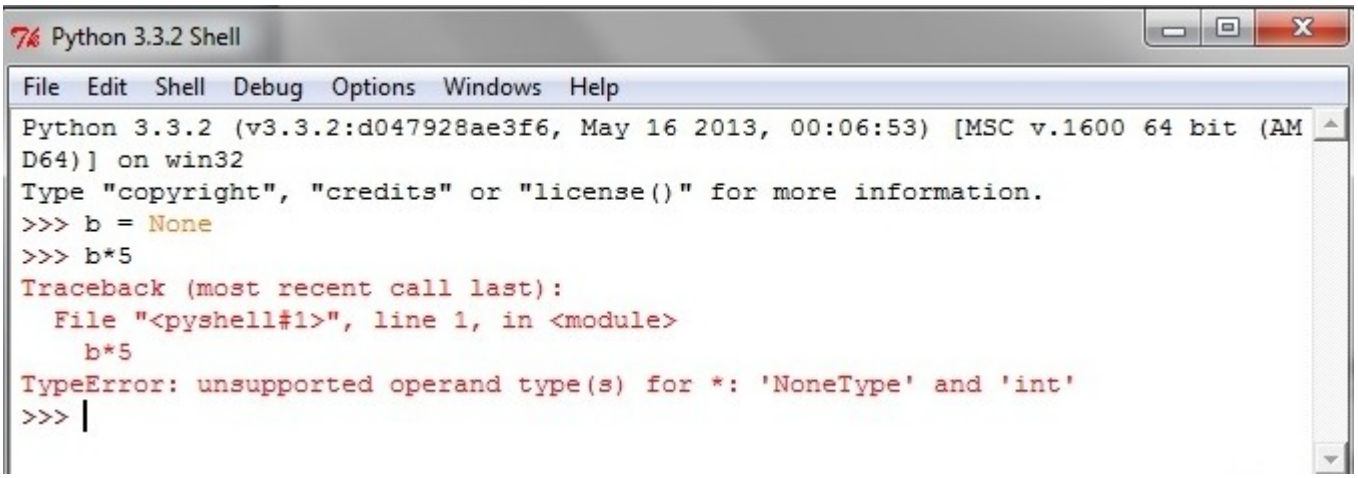
Για να την χειριστούμε, τοποθετούμε τις γραμμές που μπορούν να προκαλέσουν την εξαίρεση που θα χειριστούμε μέσα σε ένα μπλοκ try και εφόσον συμβεί μια εξαίρεση, τότε σταματάει η εκτέλεση των δηλώσεων αυτού του μπλοκ και περνάμε κατευθείαν στο αντίστοιχο μπλοκ except. Αν έχει συμβεί μια εξαίρεση που χειρίζεται το συγκεκριμένο μπλοκ (στην περίπτωση μας ZeroDivisionError), τότε εκτελούμε τις δηλώσεις που αυτό περιέχει και θεωρούμε πως έχουμε χειριστεί την εξαίρεση. ΄Ετσι το πρόγραμμα μας δεν σταματάει απότομα πια την εκτέλεση του!

➔ ΄Αλλοι τύποι εξαιρέσεων που θα μπορούσαμε να συναντήσουμε, φαίνονται στα παραδείγματα που ακολουθούν:

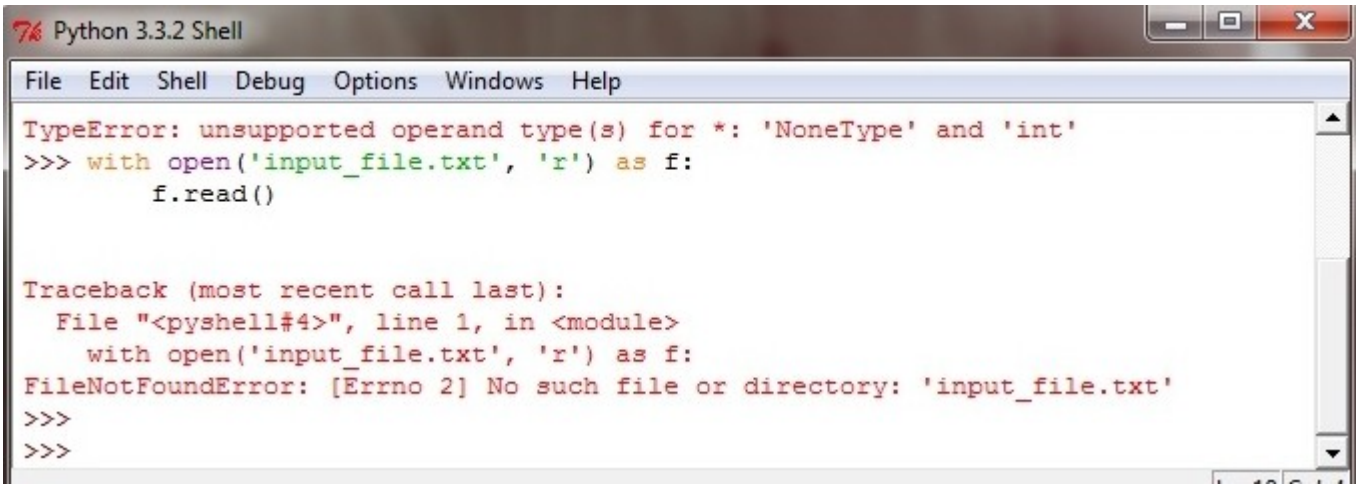


Στο παραπάνω παράδειγμα προσπαθούμε να χρησιμοποιήσουμε τον τελεστή σε μια μεταβλητή a χωρίς όμως πρώτα να την έχουμε ορίσει. Κατά αυτό τον τρόπο προκαλείται μια εξαίρεση κλάσης NameError. Αν αυτή η εξαίρεση έγινε από αβλεψία του προγραμματιστεί, τότε πρέπει να ορίσουμε κατάλληλα την μεταβλητή a, αλλιώς μπορούμε να την χειριστούμε μέσω ενός try. . . except. . . μπλοκ, αφού τώρα πια έχουμε αναγνωρίσει την κλάση της εξαίρεσης (NameError).

Αντίστοιχο σφάλμα μπορούμε να παρατηρήσουμε και στην περίπτωση που προσπαθούμε να εφαρμόσουμε μια συνάρτηση (ή τελεστή) σε τύπους που δεν υποστηρίζουν αυτή τη λειτουργία.



Τέλος, ένα ακόμη παράδειγμα για μια εξαίρεση που τουλάχιστον αρχικά, πολλοί προγραμματιστές συναντούν, αφορά τα σφάλματα εισόδου εξόδου.



Εδώ αξίζει να σημειώσουμε ότι ενώ λόγω της δομής with, αν είχε ανοίξει το αρχείο input?file.txt, τότε βγαίνοντας από τη δήλωση with, θα είχε κλείσει κατάλληλα, αν δεν κάνουμε κατάλληλο χειρισμό του σφάλματος, τότε και πάλι θα προκληθεί μια εξαίρεση την οποία δεν θα έχουμε χειριστεί.

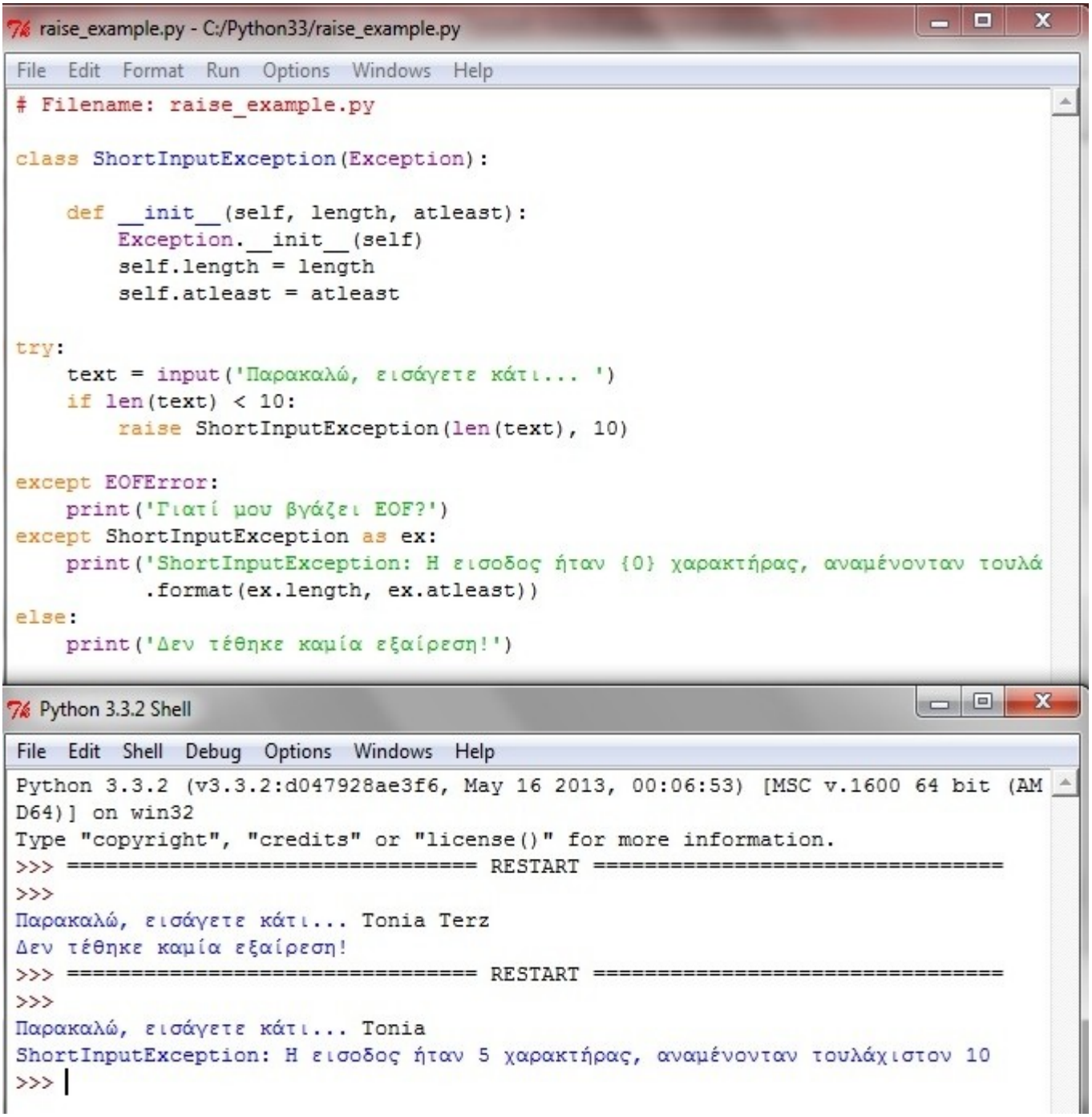
Τέλος, αξίζει να αναφέρουμε πως όπως χειριζόμαστε μια κλάση εξαιρέσεων σε ένα μπλοκ except, θα μπορούσαμε να χειριζόμαστε και πολλές κλάσεις εξαιρέσεων μαζί, χρησιμοποιώντας μια πλειάδα που να περιέχει τις κλάσεις όλων των εξαιρέσεων που θέλουμε να χειριστούμε.

```
except(RuntimeError,  TypeError,  NameError):
      Pass
```

Θα μπορούσαμε να χειριστούμε ακόμα και όλες τις εξαιρέσεις μαζί χρησιμοποιώντας ένα σκέτο except :, αν και δεν συνιστάται ως μέθοδος παρά μόνο σε πολύ συγκεκριμένες περιπτώσεις.

Ανάδειξη των εξαιρέσεων (Raising Exceptions)

Μπορείτε να αναδείξετε εξαιρέσεις χρησιμοποιώντας την **εντολή raise**, παρέχοντας την ονομασία του σφάλματος/εξαίρεσης και το αντικείμενο της εξαίρεσης είναι πρόκειται να συμβεί. Το σφάλμα ή η εξαίρεση που μπορείτε να αναδείξετε, πρέπει να είναι κλάση, η οποία άμεσα ή έμμεσα πρέπει να είναι παράγωγη της κλάσης Exception.



Πώς δουλεύει το παραπάνω πρόγραμμα :

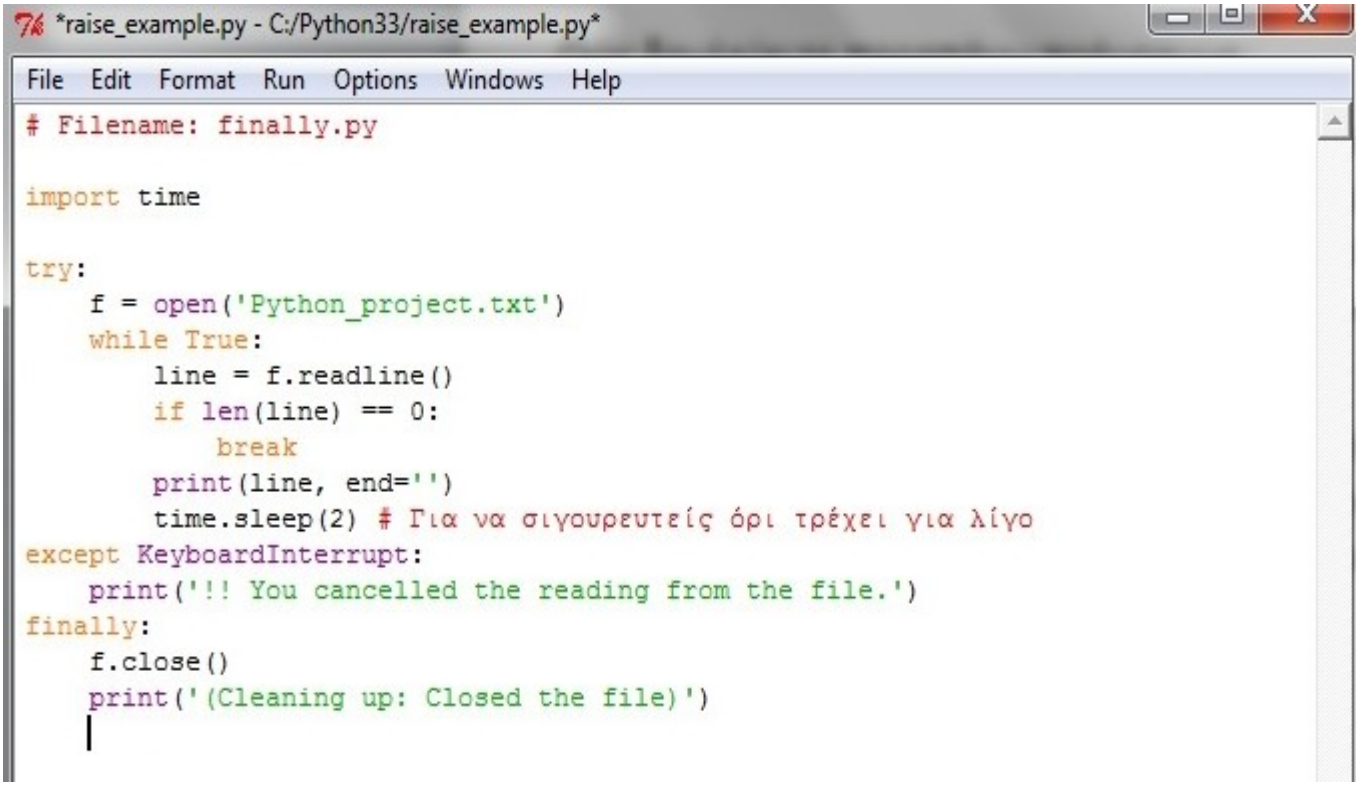
Εδώ δημιουργούμε το δικό μας τύπο εξαίρεσης. Αυτός ο νέος τύπος εξαίρεσης ονομάζεται ShortInputException. Έχει δύο πεδία, το length, το οποίο είναι το μήκος της δοθείσας εισόδου και το atleast που είναι το ελάχιστο μήκος που το πρόγραμμα περίμενε.

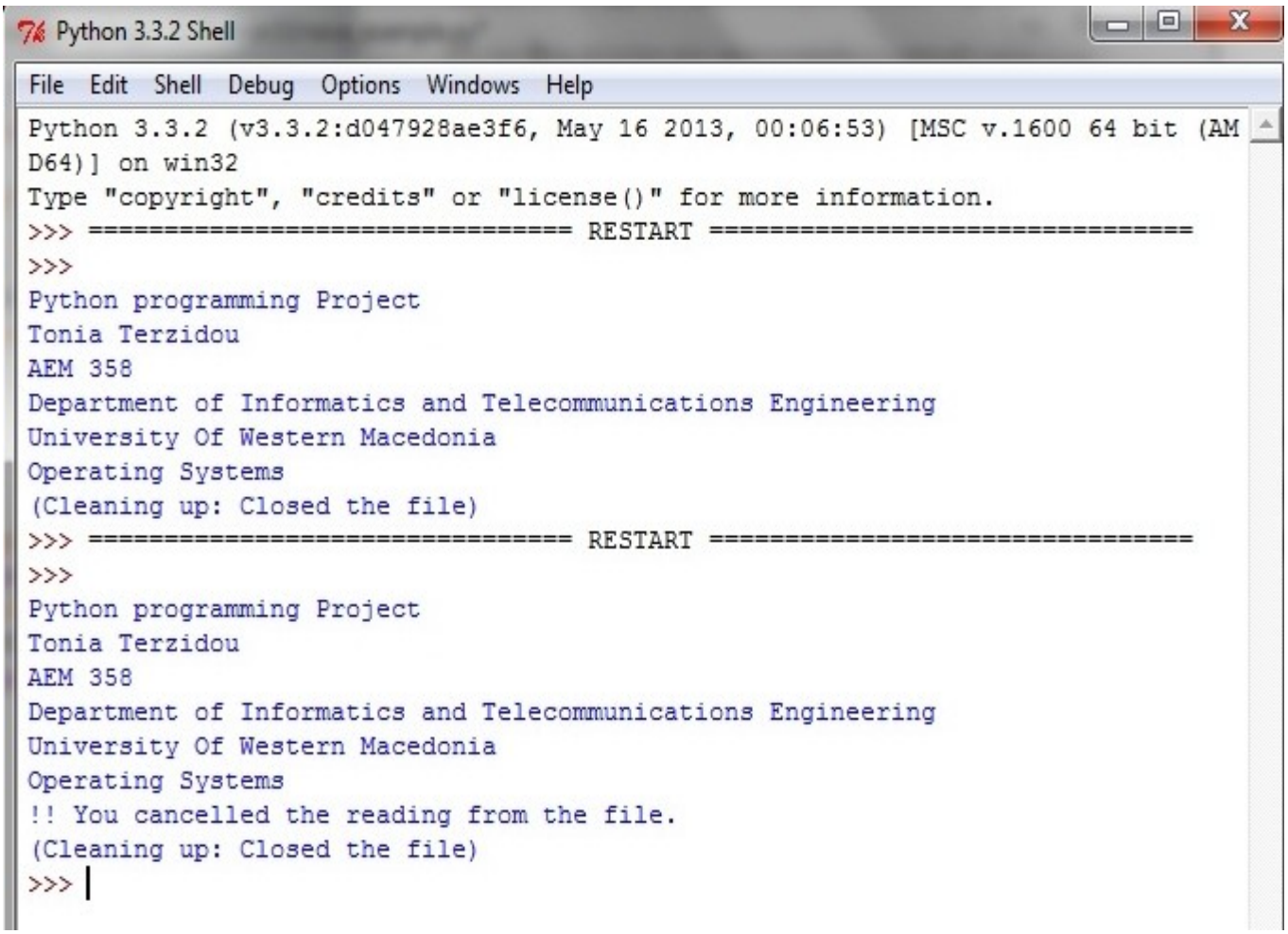
Στην πρόταση except, αναφέρουμε την κλάση του σφάλματος η οποία θα αποθηκεύεται σαν (as) το όνομα μεταβλητής το οποίο συγκρατεί το αντίστοιχο αντικείμενο σφάλματος/εξαίρεσης. Αυτό είναι ανάλογο με τις παραμέτρους και τα ορίσματα σε μια κλήση συνάρτησης.

Μέσα σε αυτή την ειδική πρόταση except, χρησιμοποιούμε τα πεδία length και atleast του αντικειμένου της εξαίρεσης, για να τυπώσουμε ένα κατάλληλο μήνυμα στο χρήστη.

Try .. Finally

Υποθέστε ότι διαβάζετε ένα αρχείο στο πρόγραμμά σας. Πώς επιβεβαιώνετε ότι το αντικείμενο του αρχείου έχει κλειστεί κανονικά, είτε η εξαίρεση αναδείχθηκε είτε όχι; Αυτό μπορεί να γίνει χρησιμοποιώντας την **πλοκάδα finally**. Μπορείτε να χρησιμοποιήσετε μια **πρόταση except μαζί με την πλοκάδα finally** για την ίδια αντιστοιχούσα πλοκάδα try. Πρέπει να ενσωματώσετε τη μια μέσα στην άλλη, εάν θέλετε να χρησιμοποιήσετε και τις δύο.





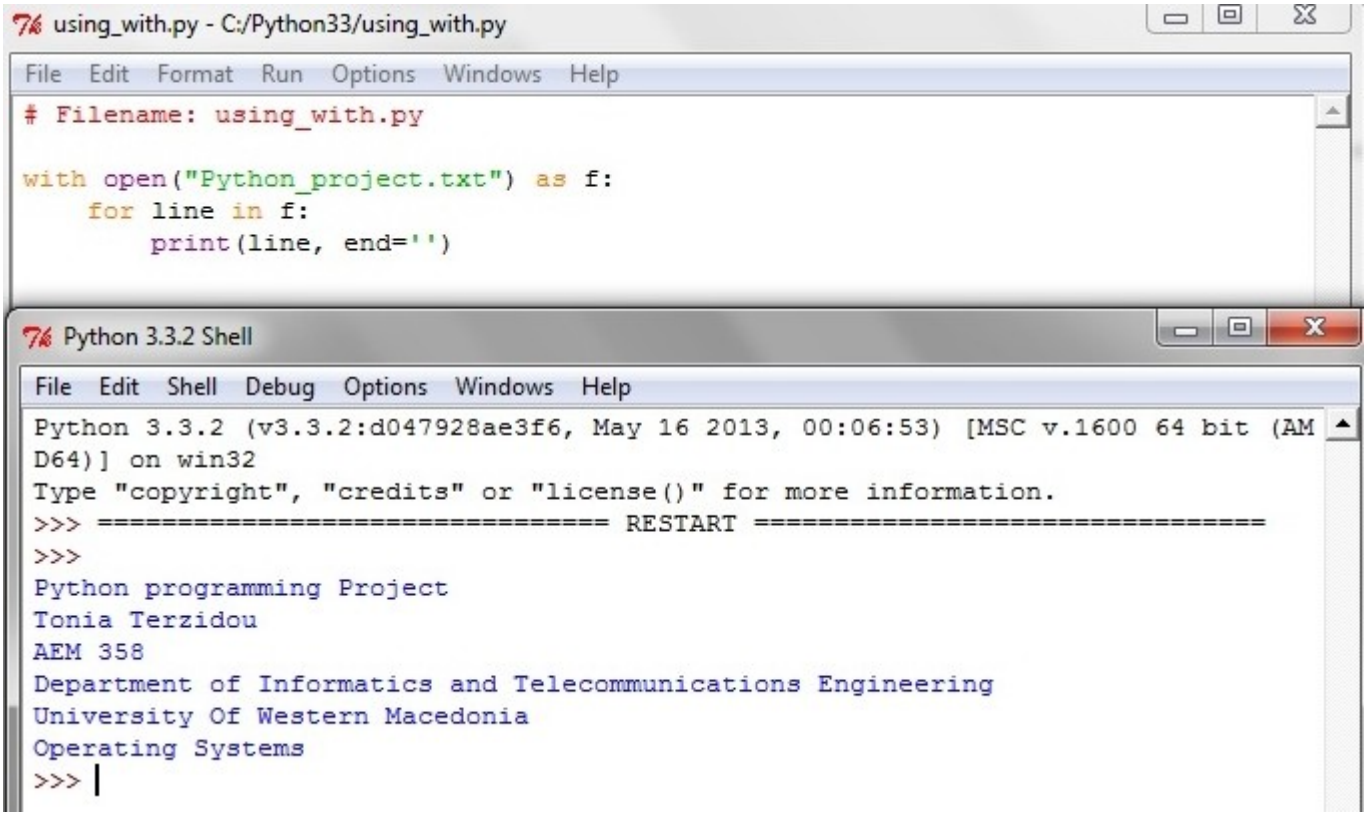
Πώς δουλεύει το παραπάνω πρόγραμμα :

Κάνουμε το σύνηθες διάβασμα αρχείου, αλλά αυθαίρετα έχουμε εισάγει “ύπνωση” για 2 δευτερόλεπτα αφού τυπωθεί η κάθε γραμμή, χρησιμοποιώντας τη συνάρτηση time.sleep έτσι ώστε το πρόγραμμα να τρέχει αργά (η Python από τη φύση της τρέχει πολύ γρήγορα!). Καθώς το πρόγραμμα τρέχει ακόμα, ***πιέστε ctrl-c για να διακόψετε/ακυρώσετε το πρόγραμμα.***

Παρατηρήστε ότι συμβαίνει η εξαίρεση KeyboardInterrupt και το πρόγραμμα εγκαταλείπεται. Πάντως, πριν το πρόγραμμα εγκαταλειφθεί, η πρόταση finally εκτελείται και το αντικείμενο του αρχείου πάντα κλείνεται.

Η εντολή with

Η απόκτηση ενός πόρου στην πλοκάδα try και ακολούθως η απελευθέρωση του πόρου στην πλοκάδα finally είναι ένα συνηθισμένο μοτίβο. Γι' αυτό το λόγο υπάρχει και η εντολή with, η οποία το κάνει ικανό να γίνει με καθαρό τρόπο:



Πως δουλεύει το παραπάνω πρόγραμμα :

Η έξοδος θα έπρεπε να είναι ίδια με το προηγούμενο παράδειγμα. Η διαφορά εδώ είναι ότι χρησιμοποιούμε τη συνάρτηση open με την εντολή with. Αφήνουμε το κλείσιμο του αρχείου να γίνει αυτόματα με το with open.

Αυτό που συμβαίνει παρασκηνιακά είναι ότι υπάρχει ένα πρωτόκολλο που χρησιμοποιείται με την εντολή with. Φέρνει το αντικείμενο που επιστράφηκε από την εντολή open. Ας το ονομάσουμε “ the_file” σε αυτή την περίπτωση.

Πάντα καλεί τη συνάρτηση thefile.__enter__ πριν αρχίσει η πλοκάδα του κώδικα κάτω από αυτό, και πάντα καλεί το thefile.__exit__, αφού τελειώσει η πλοκάδα του κώδικα. Έτσι ο κώδικας που θα είχαμε γράψει σε μια πλοκάδα finally θα έπρεπε να φροντίζεται αυτόματα από τη μέθοδο __exit__. Αυτό είναι που μας βοηθάει να αποφεύγουμε να χρησιμοποιούμε ρητές εντολές try..finally επανειλημμένως.

Γεννήτορες (generators)

Οι γεννήτορες μας δίνουν τη δυνατότητα να κάνουμε τη δουλειά που χρειάζεται, τη στιγμή που χρειάζεται. Μπορούμε να τους σκεφτούμε ως συναρτήσεις, όπου όμως παράγουν τμηματικά την έξοδό τους, ανάλογα με το τι τους ζητάμε. Οφέλη που αυτό μπορεί να έχει : δεν «κολλάει» πια ανεξήγητα ένα πρόγραμμα όταν του ζητάμε κάτι φαινομενικά απλό,χρησιμοποιείται όση μνήμη χρειάζεται.

Επαναλήπτες (Iterators)

Για να καταλάβουμε πως λειτουργούν οι γεννήτορες, πρέπει πρώτα να καταλάβουμε τους *επαναλήπτες (iterators)*. ΄Οταν προσπελαύνουμε τα στοιχεία μιας λίστας ένα ένα, χρησιμοποιούμε επαναλήπτες.

```
mylist =  [1 ,  2,  3,  4,  5]

for  i  in mylist :

print (i)
```

Επαναλήπτης (iterator) είναι ο όρος που χρησιμοποιούμε για να καταδείξουμε ότι ένα αντικείμενο έχει μια next() μέθοδο.

Πώς δουλεύουν οι for βρόγχοι

΄Οποτε γράφουμε :

```
for i in mylist:

    …loop body…
```

Η Python κάνει τα ακόλουθα βήματα:

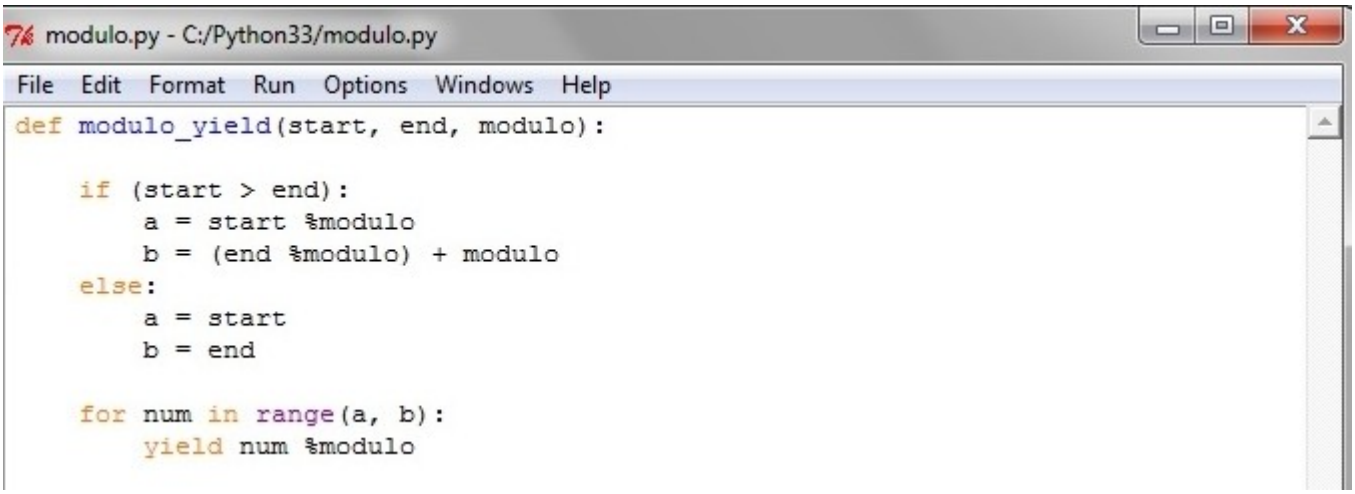
1) Παίρνει ένα επαναλήπτη για την next(). Καλεί την iter(mylist) η οποία επιστρέφει ένα αντικείμενο με την μέθοδο next().
2) Χρησιμοποιεί τον επαναλήπτη για να διατρέξει όλα τα αντικείμενα στην λίστα. ΄Ετσι το i παίρνει επαναληπτικά όλες τις τιμές που ρίσκονται στην λίστα mylist. ια να γίνει αυτό, καλείται συνεχώς η συνάρτηση next() στον επαναλήπτη i που επιστρέφεται από το πρώτο ήμα. Η επιστρεφόμενη τιμή ανατίθεται στο i και το σώμα του βρόγχου εκτελείται. Εάν η εξαίρεση StopIteration προκύψει από την next() σημαίνει πως δεν υπάρχουν άλλες τιμές να ανακτηθούν από την mylist και έτσι ο βρόγχος τερματίζει.

Δημιουργία γεννητόρων

Οι γεννήτορες (generators) δημιουργούνται όταν χρησιμοποιούμε την **λέξη κλειδί yield** αντί της return. Μπορούμε να τις αντιμετωπίσουμε σαν τις κλασικές συναρτήσεις με μόνο μια διαφορά. ΄Ενας γεννήτορας, επιστρέφει μια τιμή με την yield. ΄Οταν ξανακλειθεί, συνεχίζει από την κατάσταση που ήταν μόλις κλήθηκε η yield, μέχρι να φθάσει ξανά σε κάποιο άλλο yield. ΄Ετσι, μπορεί να χρησιμοποιηθεί για να παράγονται δυναμικά τιμές, καταλαμβάνοντας έτσι μικρότερο χώρο στην μνήμη, αφού δεν χρειάζεται να παραχθούν όλες και να επιστραφούν.

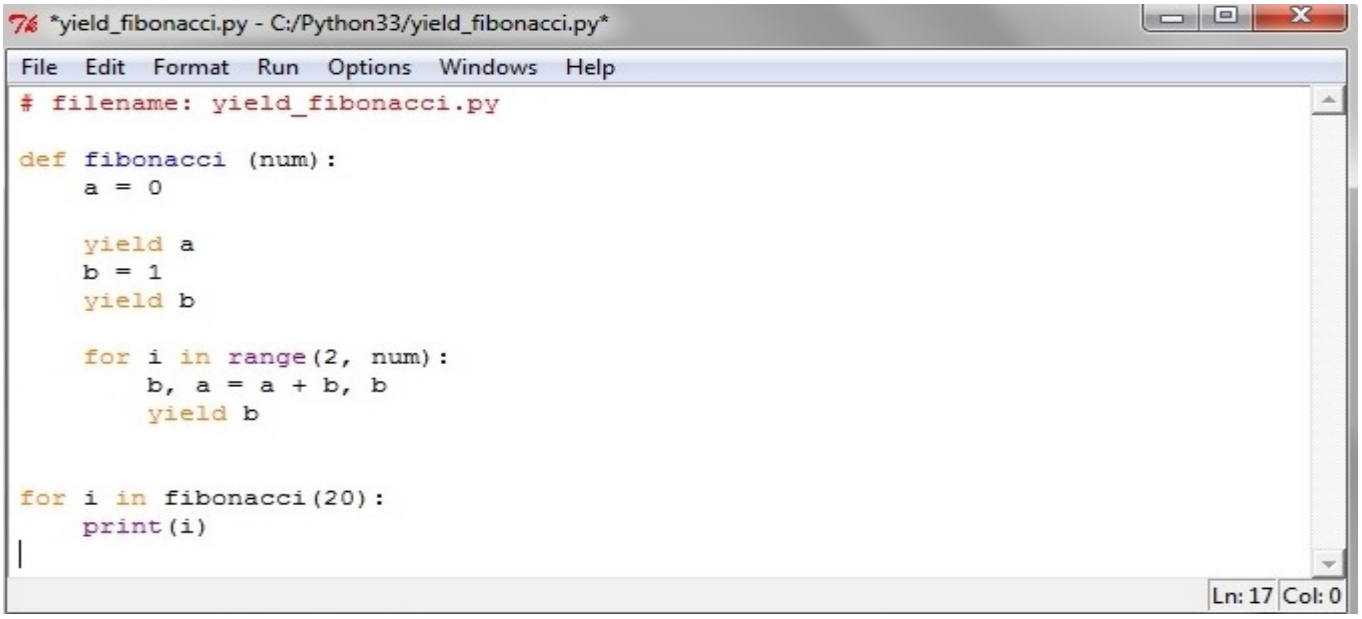
Γεννήτορας μια ειδική συνάρτηση η οποία χρησιμοποιείται για την παραγωγή iterators. Κάθε φορά που καλούμε έναν γεννήτορα, συνεχίζει την εκτέλεση του από το σημείο που είχε μείνει στην προηγούμενη εκτέλεση του. ΄Ετσι, η έξοδος του παράγεται τμηματικά. Συνεπώς είναι ιδιαίτερα χρήσιμος όταν η παραγόμενη έξοδος είναι πολύ μεγάλη και μπορούμε να την τμηματοποιήσουμε.

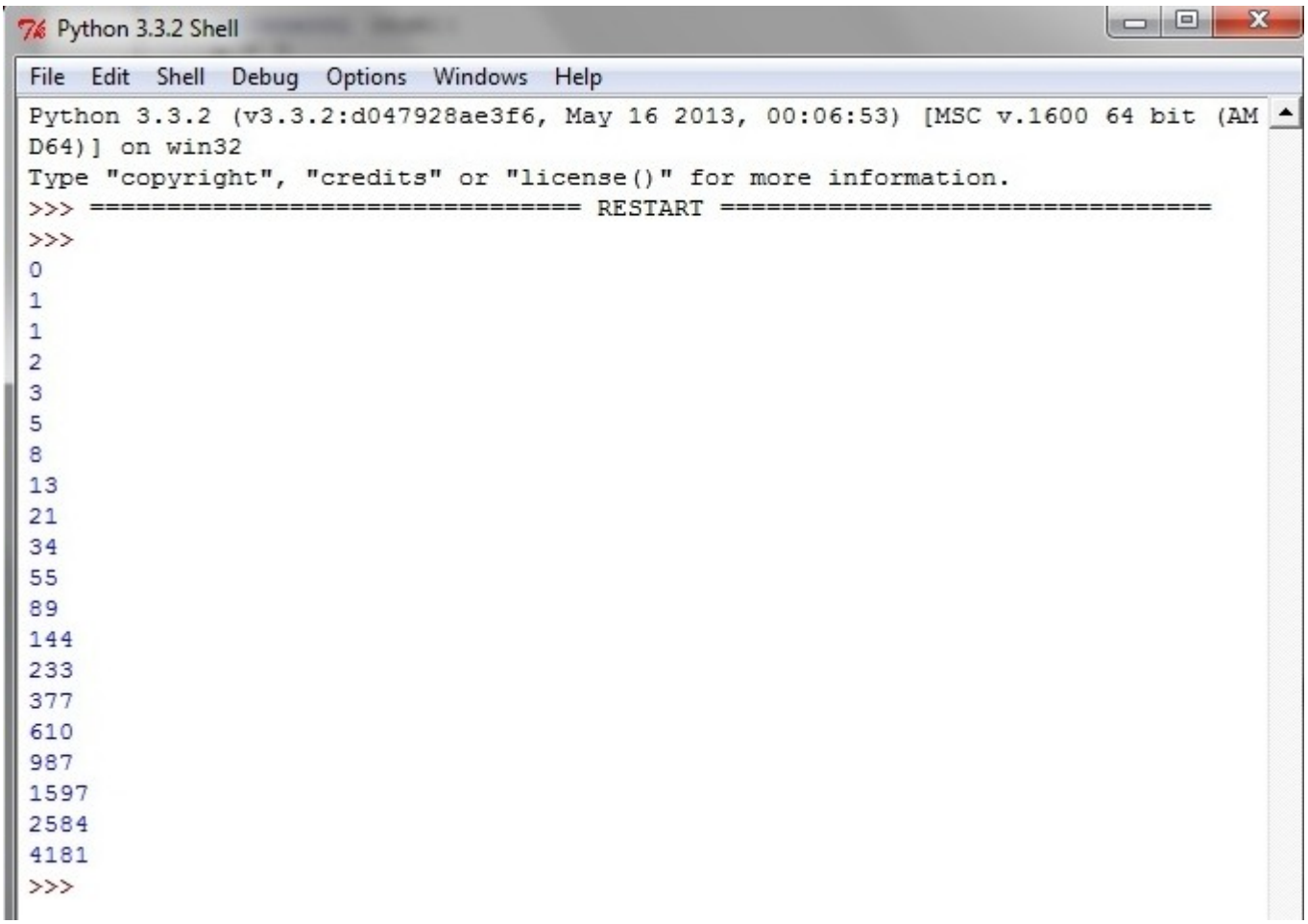
➔ Στο ακόλουθο παράδειγμα βλέπουμε μια συνάρτηση η οποία δουλεύει σαν την range() με την διαφορά ότι αν το δεύτερο όρισμα (end) είναι μικρότερο από το πρώτο (start), τότε θα κάνει την πράξη modulo μέχρι να φθάσει σε αυτό.



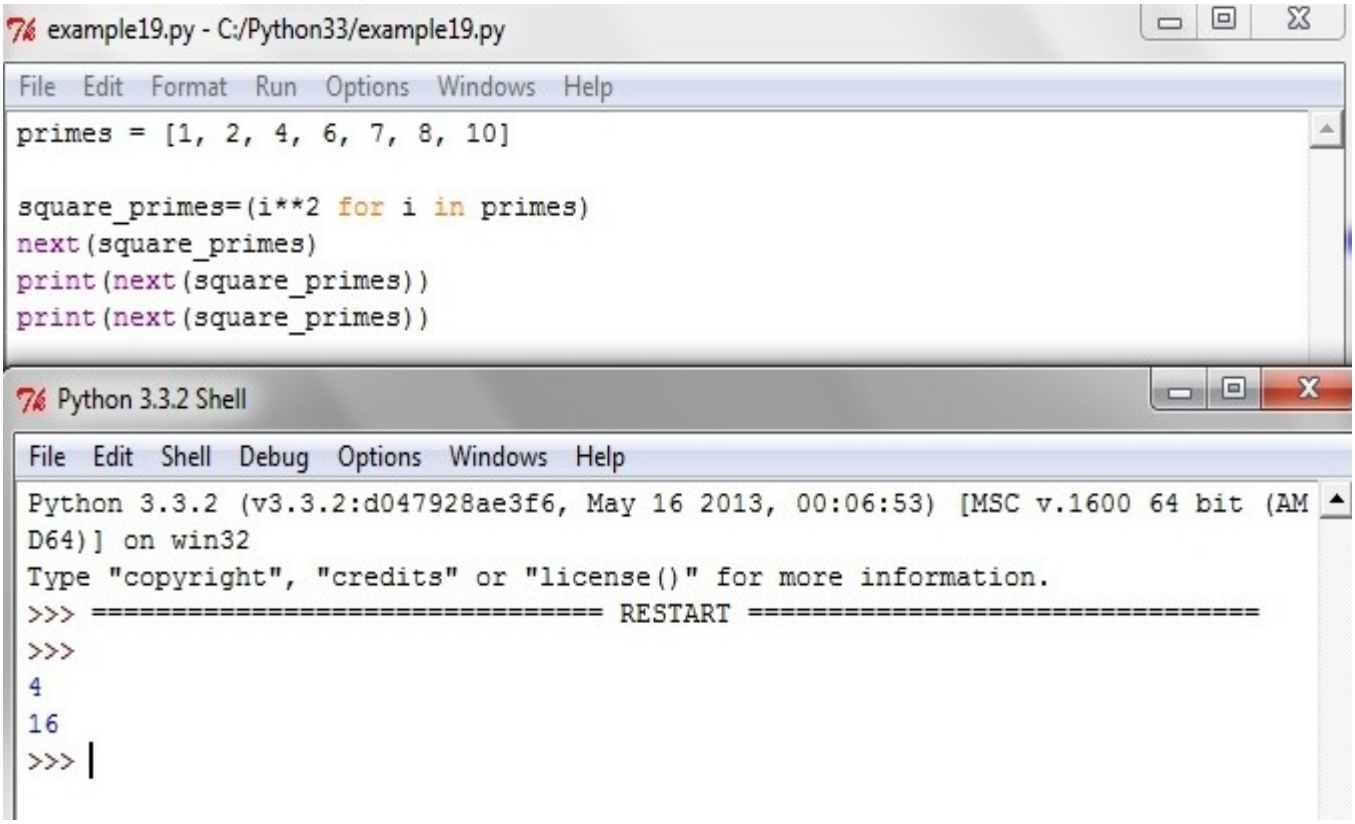
Ο έλεγχος αν start > end γίνεται μόνο μια φορά, όταν καλείται για πρώτη φορά ο παραπάνω γεννήτορας. Στην συνέχεια, η yield ρίσκεται μέσα σε έναν βρόγχο for. Αυτό πρακτικά σημαίνει, ότι κάθε φορά που καλείται ο γεννήτορας, θα βρίσκεται πάντα μέσα σε αυτόν τον βρόχο, παράγοντας έτσι την επόμενη τιμή.

➔ ΄Ενα άλλο παράδειγμα είναι η δημιουργία της ακολουθίας των Fibonacci αριθμών.





- ✓ Μπορούμε να δημιουργήσουμε γεννήτορες μέσω εκφράσεων γεννητόρων σε αντιστοιχία με τις lists comprehensions.



΄Ενα γεννήτορας, επομένως, μπορεί να θεωρηθεί ως ένα αντικείμενο στο οποίο μπορούμε να καλέσουμε την συνάρτηση next(). Κάθε κλήση της συνάρτησης next() προκαλεί την επιστροφή του επόμενου αντικειμένου μέχρι να δημιουργηθεί μια εξαίρεση τύπου StopIteraton η οποία και σημαίνει το τέλος των αντικειμένων των οποίων μπορούν να παραχθούν από τον συγκεκριμένο γεννήτορα.

Παρότι το αντικείμενο του γεννήτορα δημιουργείται μόνο μια φορά, ο κώδικας του δεν τρέχει μόνο μια φορά. Κάθε φορά τρέχει ένα μέρος του κώδικα του (μέχρι να συναντηθεί το yield) και κρατιέται η κατάσταση του στη μνήμη. Την επόμενη φορά συνεχίζει η εκτέλεση από το ίδιο σημείο, από την κατάσταση που είχε σταματήσει μέχρι το επόμενο yield.

Συγγραφή κώδικα φιλικό προς τους γεννήτορες

Ορισμένα πλεονεκτήματα των γεννητορων είναι :

1. Μικρότερη κατανάλωση μνήμης όταν χρειάζεται να παραχθούν πολλά και μεγάλα αντικείμενα. Αντί να πρέπει να δημιουργηθεί μια συνάρτηση που θα επιστρέφει μια τεράστια λίστα γεμάτη αντικείμενα, μπορούμε να γράψουμε έναν γεννήτορα ο οποίο θα παράγει τα απαραίτητα αντικείμενα όταν αυτά χρειαστούν (on the fly). ΄Ετσι, μπορούμε να παράγουμε σειρές αντικειμένων που αλλιώς δεν θα χωρούσαν στη μνήμη.
2. Είναι ένας πολύ αποτελεσματικός τρόπος να γίνει parsing σε μεγάλα αρχεία χωρίς να χρειάζεται να τα φορτώσουμε ολόκληρα στη μνήμη. ια παράδειγμα μπορούμε να τα διαβάζουμε γραμμή γραμμή και να τα επεξεργαζόμαστε.
3. Αποτελούν έναν φυσικό τρόπο να περιγραφούν άπειρες σειρές δεδομένων. Παραδείγματος χάρη θα μπορούσαμε να δημιουργήσουμε έναν γεννήτορα που να μπορεί να παράγει σιγά σιγά κάθε αριθμό Fibonacci.

Πολλές φορές, συναρτήσεις που γράφουμε περιμένουν ως είσοδο κάποια λίστα ενώ θα μπορούσαν να δεχθούν και κάποιον γεννήτορα στην θέση της βελτιώνοντας έτσι την χρήση της κύριας μνήμης του προγράμματος. Για αυτό τον λόγο, αν δεν απαιτείται τυχαία προσπέλαση στα στοιχεία μιας λίστας αλλά αρκεί μια διάτρεξη π.χ. μέσω ενός βρόγχου for, τότε καλό είναι ο κώδικας μας να μπορεί να δουλέψει και με τους δύο τύπους εισόδου. ια αυτό το λόγο υπάρχουν ορισμένοι κανόνες που θα μπορούσαμε να ακολουθούμε:

1) Να μην χρησιμοποιείται η συνάρτηση len(). Πολλές φορές την χρησιμοποιούμε μόνο και μόνο για να προσδιορίσουμε πότε τελείωσε η διάτρεξη της λίστας (αντίστοιχα θα ήταν η παραγωγή των αντικειμένων από τον γεννήτορα) ενώ αυτό μπορεί να γίνει αυτόματα από το for βρόγχο.

2) Να μην ζητούνται αυθαίρετα στοιχεία (πχ το έβδομο στοιχείο) όταν τελικά θα τα προσπελαύσουμε όλα και δεν μας ενδιαφέρει η σειρά τους.

΄Ενα κλασικό παράδειγμα που συνοψίζει τους παραπάνω κανόνες, είναι η αποφυγή του:

```
for i in range(0, len (l)):
        e = l[i]
```

και η αντικατάσταση του με:

```
for i, e in enumerate (l):
```

που έχει ακριβώς το ίδιο αποτέλεσμα και είναι πιο καλογραμμένο.

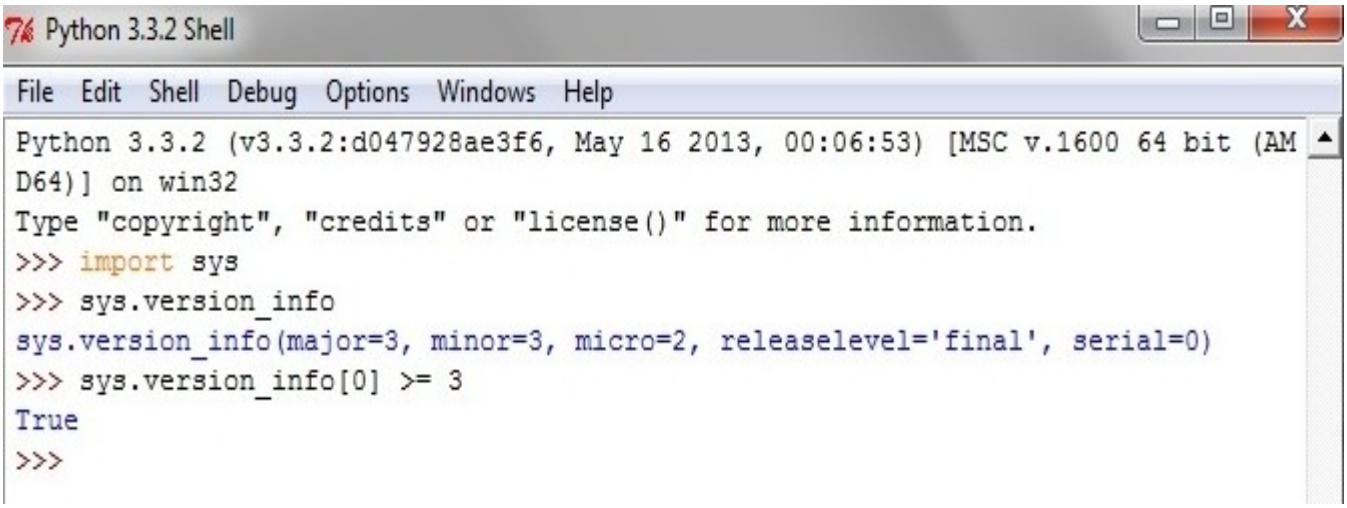
Πρότυπη βιβλιοθήκη

Η πρότυπη βιβλιοθήκη της Python περιέχει έναν τεράστιο αριθμό χρήσιμων αρθρωμάτων και είναι μέρος κάθε πρότυπης εγκατάστασης Python. Είναι σημαντικό να εξοικειωθείτε με την πρότυπη βιβλιοθήκη, επειδή πολλά προβλήματα μπορούν να λυθούν γρήγορα εάν είστε εξοικειωμένοι με το εύρος των πραγμάτων που μπορούν να κάνουν οι βιβλιοθήκες.

Θα μελετήσουμε μερικά από τα πιο συνηθισμένα αρθρώματα σε αυτή τη βιβλιοθήκη.

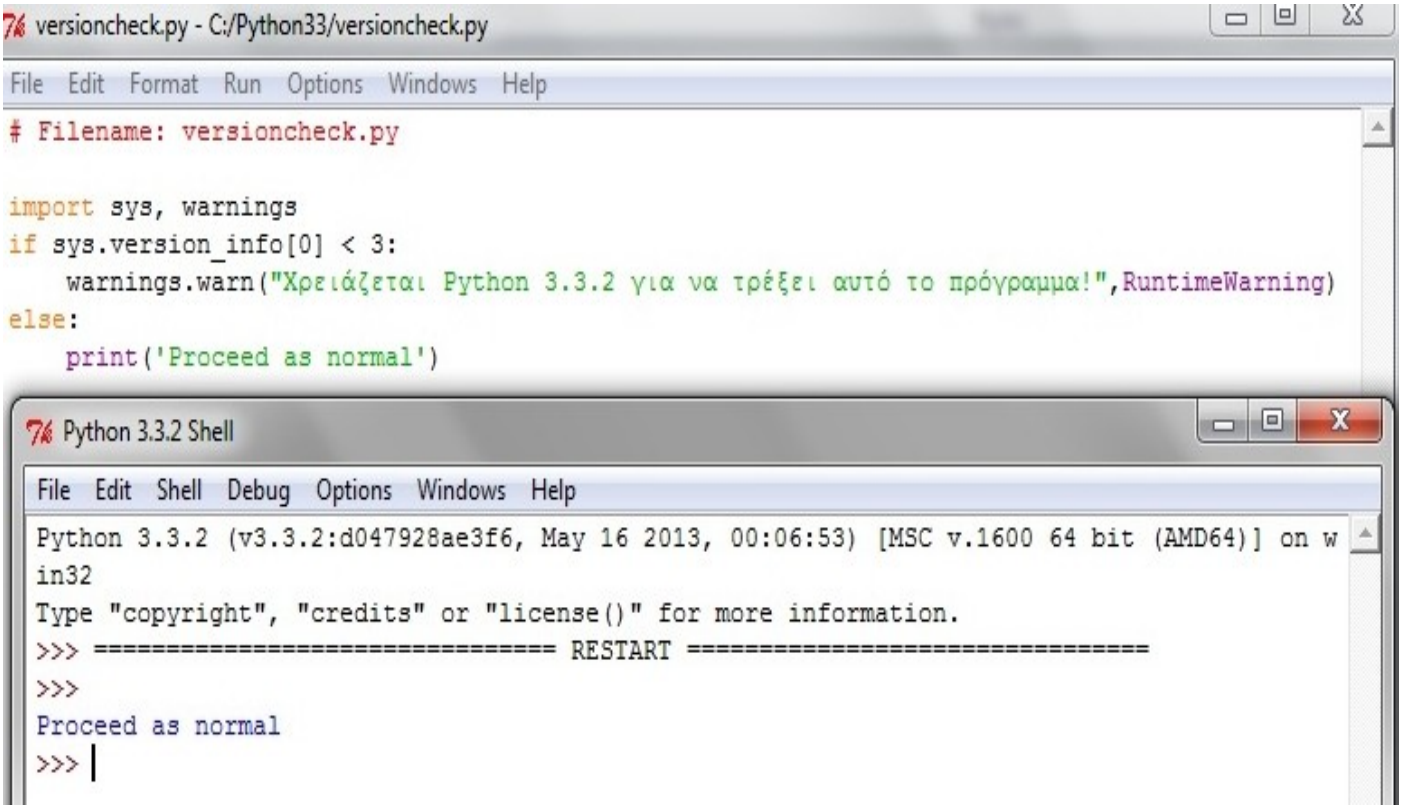
Το άρθρωμα sys

Το άρθρωμα sys περιέχει λειτουργικότητα ειδικά για το σύστημα. Έχουμε ήδη δει ότι η λίστα sys.argv περιέχει τα ορίσματα της γραμμής εντολών. Υποθέτουμε ότι θέλουμε να ελέγξουμε την έκδοση της Python που χρησιμοποιεί το σύστημά μας.Το άρθρωμα sys μας δίνει αυτή τη λειτουργία.



Πώς δουλεύει το παραπάνω :

Το άρθρωμα sys έχει την πλειάδα version_info που μας δίνει την πληροφορία για την έκδοση. Η πρώτη καταχώρηση είναι η κύρια έκδοση. Αυτό μπορούμε να το ελέγξουμε, για παράδειγμα, για να επιβεβαιώσουμε ότι το πρόγραμμα τρέχει μόνο υπό την Python 3.3.2

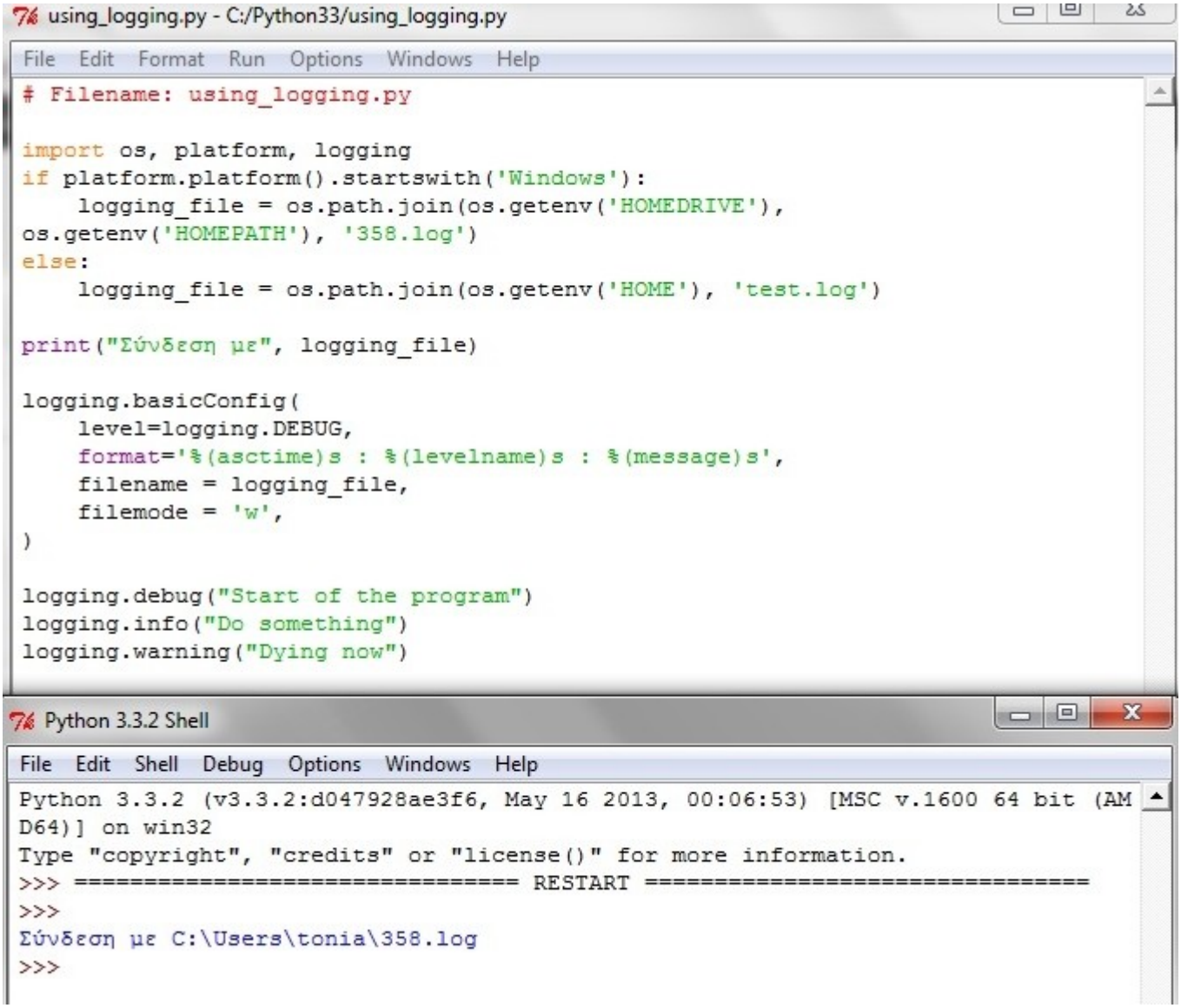


Πώς δουλεύει το παραπάνω πρόγραμμα :

Χρησιμοποιούμε ένα άλλο άρθρωμα από την πρότυπη βιβλιοθήκη που ονομάζεται warnings, που χρησιμοποιείται για να εμφανίσει προειδοποιήσεις στον τελικό χρήστη. Εάν το νούμερο της έκδοσης Python δεν είναι τουλάχιστον 3, εμφανίζουμε την αντίστοιχη προειδοποίηση.

Το άρθρωμα logging

Τι θα κάνατε εάν θέλατε να αποθηκεύονται κάπου μερικά μηνύματα εκσφαλμάτωσης ή άλλα σημαντικά μηνύματα, έτσι ώστε να μπορείτε να ελέγξετε εάν το πρόγραμμά σας έχει τρέξει όπως θα περιμένατε; Πώς “αποθηκεύετε κάπου” αυτά τα μηνύματα; Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας το άρθρωμα logging!



Πως λειτουργεί το παραπάνω πρόγραμμα :

Χρησιμοποιούμε τρία αρθρώματα από την πρότυπη βιβλιοθήκη - το άρθρωμα os για αλληλεπίδραση με το λειτουργικό σύστημα, το άρθρωμα platform για πληροφορίες σχετικά με την πλατφόρμα, δηλαδή το λειτουργικό σύστημα, και το άρθρωμα logging για τις καταγραφές (log).

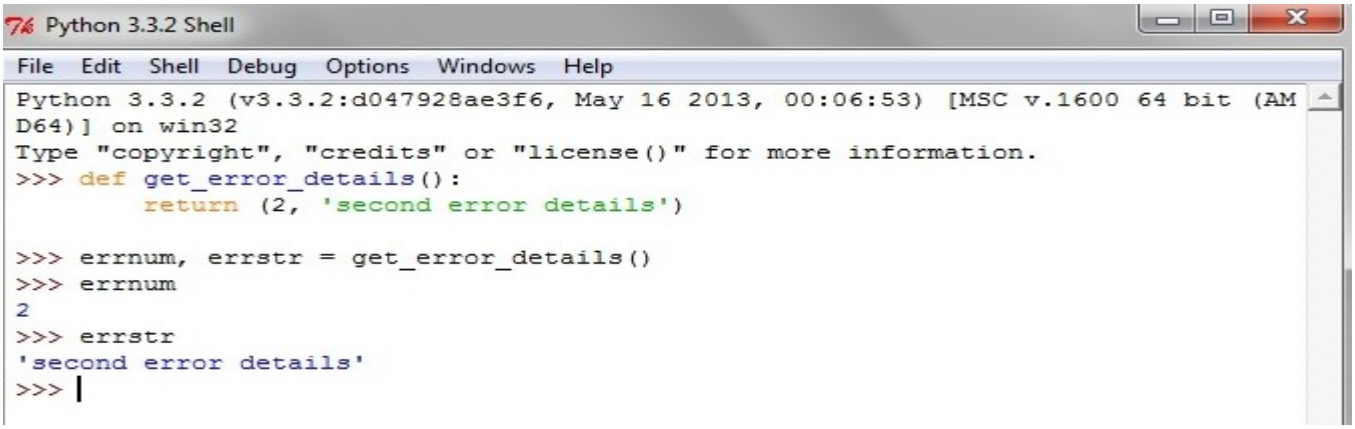
Αρχικά, ελέγχουμε ποιο λειτουργικό σύστημα χρησιμοποιούμε, ελέγχοντας τη συμβολοσειρά που επιστρέφεται από την platform.platform() (για περισσότερες πληροφορίες, δείτε την import platform; help(platform)). Εάν είναι Windows εμφανίζουμε τον home drive, το φάκελο home και το όνομα του αρχείου (filename) όπου θέλουμε να αποθηκεύσουμε τις πληροφορίες. Τοποθετώντας αυτά τα τρία μέρη μαζί, παίρνουμε την ολοκληρωμένη θέση του αρχείου. Για άλλες πλατφόρμες, απαιτείται να ξέρουμε μόνο το φάκελο home του χρήστη και παίρνουμε την πλήρη θέση του αρχείου.Χρησιμοποιούμε τη συνάρτηση os.path.join() για να τοποθετήσουμε αυτά τα τρία μέρη της θέσης του αρχείου μαζί. Ο λόγος για τον οποίο χρησιμοποιούμε μια ειδική συνάρτηση, αντί να προσθέσουμε απλά τις

συμβολοσειρές μαζί, είναι διότι αυτή η συνάρτηση θα επιβεβαιώσει ότι η πλήρης θέση ταιριάζει με το μορφότυπο (format) που αναμένεται από το λειτουργικό σύστημα.

Ρυθμίζουμε το άρθρωμα logging για να γράψουμε όλα τα μηνύματα σε ένα ιδιαίτερο μορφότυπο στο αρχείο που έχουμε καθορίσει. Τελικά, μπορούμε να βάλουμε μηνύματα που προορίζονται είτε για εκσφαλμάτωση, είτε για πληροφορία, είτε για προειδοποίηση, είτε κρίσιμα μηνύματα. Άπαξ και το πρόγραμμα έχει τρέξει, μπορούμε να ελέγξουμε αυτό το αρχείο και θα γνωρίζουμε τι συνέβει στο πρόγραμμα, ακόμα κι αν καμία πληροφορία δεν εμφανίστηκε στον χρήστη που τρέχει το πρόγραμμα.

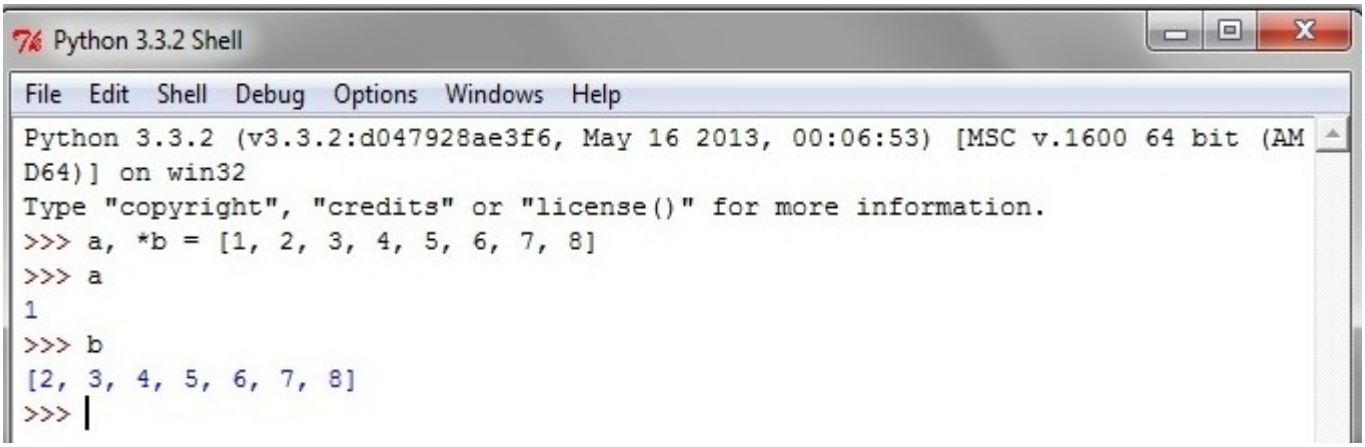
Μερικές σημαντικές επιπρόσθετες παρατηρήσεις

- Όταν θέλουμε να επιστρέψουμε δύο διαφορετικές τιμές από μια συνάρτηση,το μόνο που χρειάζεται είναι να χρησιμοποιήσετε μια πλειάδα.

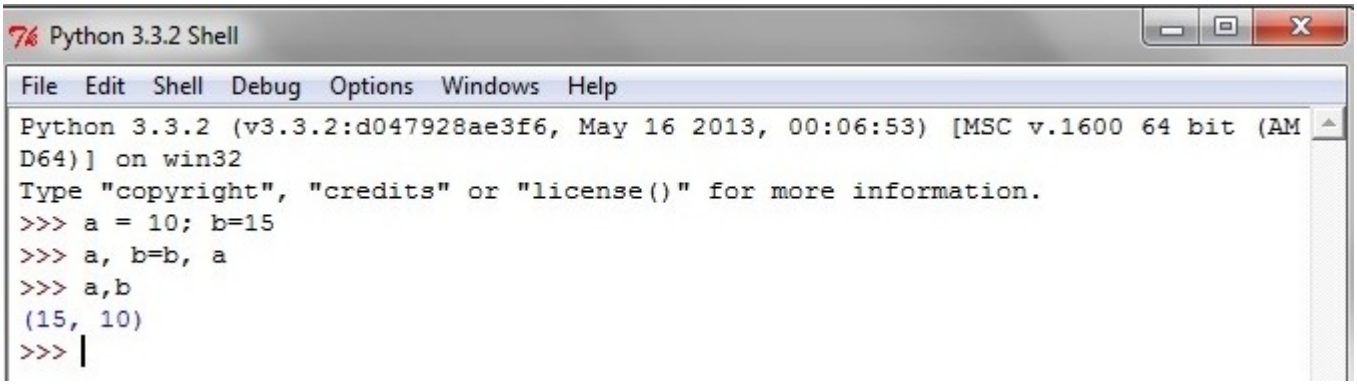


Παρατηρούμε ότι η χρήση του a, b = <κάποια έκφραση> ερμηνεύει το αποτέλεσμα της έκφρασης ως μια πλειάδα με δύο τιμές.

Αν θέλετε να ερμηνεύσετε τα αποτελέσματα ως (a, <οτιδήποτε άλλο>), τότε απλά προσθέστε ένα αστερίσκο, όπως θα κάνατε και με τις παραμέτρους μιας συνάρτησης:



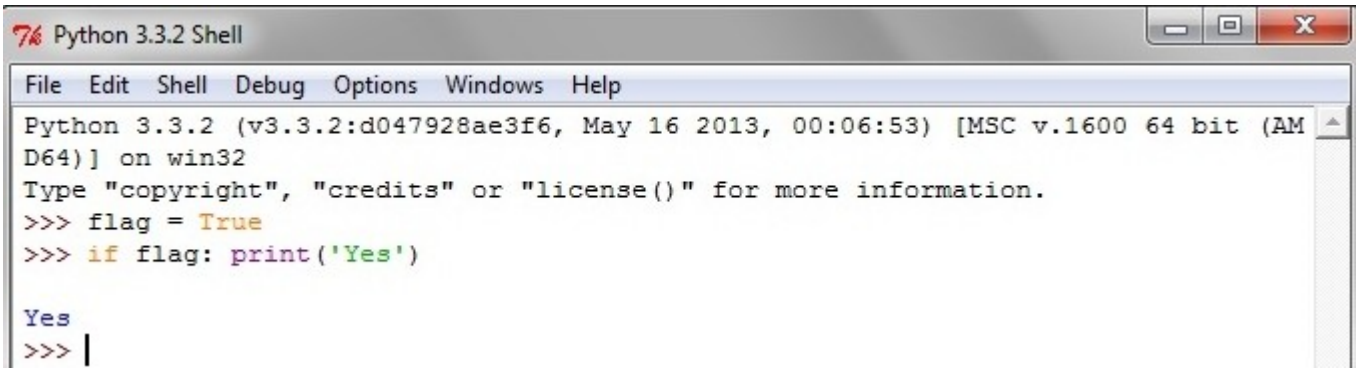
Συνεπώς, ο πιο γρήγορος τρόπος για να ανταλλάξετε τις τιμές δύο μεταβλητων στην Python είναι:



Πλοκάδες μοναδικών εντολών

Αναφέραμε ότι κάθε πλοκάδα εντολών ξεχωρίζεται από τις υπόλοιπες από το δικό της επίπεδο εσοχής του κώδικα. Όμως, υπάρχει ένα μειονέκτημα. Αν η πλοκάδα σας περιέχει μόνο μια μοναδική εντολή, τότε μπορείτε να την ορίσετε στην ίδια γραμμή με μια συνθήκη ή ένα βρόχο, για παράδειγμα.

➔ Ακολουθεί σχετικό παράδειγμα για καλύτερη κατανόηση :

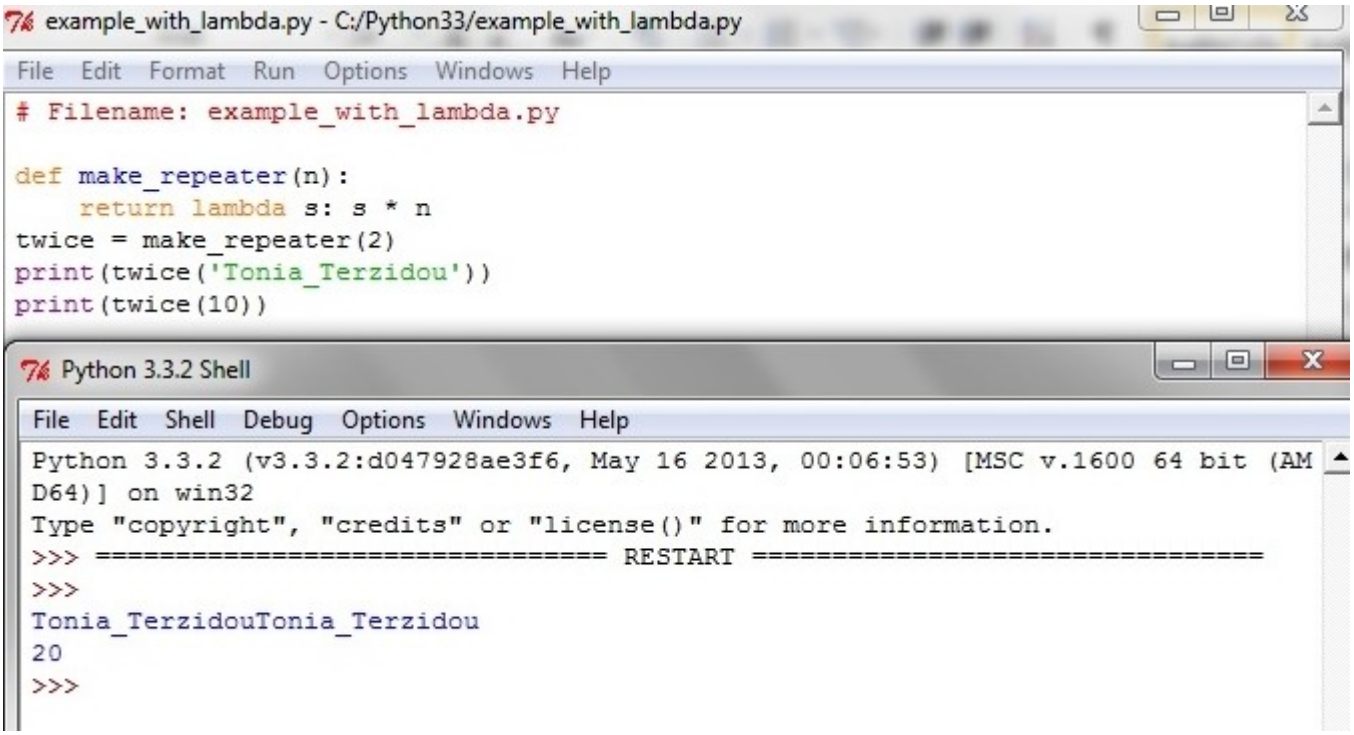


Παρατηρούμε ότι η μοναδική εντολή χρησιμοποποιείται στην ίδια θέση και όχι ως ξεχωριστή πλοκάδα. Παρότι μπορείτε να το χρησιμοποιήσετε αυτό για να κάνετε το πρόγραμμά σας "μικρότερο", συνιστάται να αποφύγετε αυτή τη μέθοδο συντόμευσης, εκτός όταν ελέγχετε για λάθη, κυρίως γιατί θα είναι πολύ ευκολότερο να προσθέσετε μια γραμμή κώδικα αν χρησιμοποιείτε κανονική εσοχή κώδικα.

Lambda Forms

Η εντολή lambda, για την οποία κάναμε μια σύντομη αναφορά και παραπάνω(βλέπε ανώνυμες συναρτήσεις) χρησιμοποιείται για να δημιουργήσει νέα αντικείμενα συναρτήσεων και να τα επιστρέψει κατά την εκτέλεση.

➔ Ακολουθεί ένα παράδειγμα :



Πώς δουλεύει το παραπάνω πρόγραμμα :

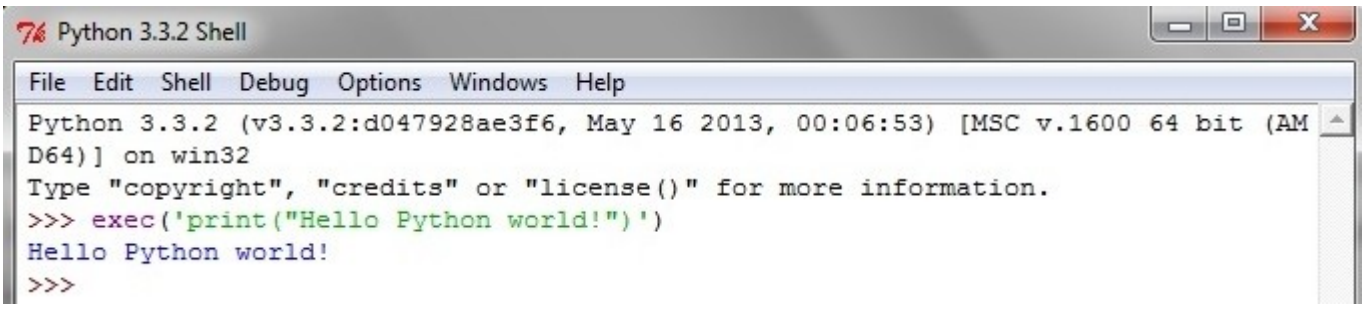
Εδώ χρησιμοποιούμε τη συνάρτηση make_repeater για να δημιουργήσουμε καινούρια αντικείμενα συναρτήσεων κατά την εκτέλεση και να τα επιστρέψουμε. Η εντολή lambda χρησιμοποιείται για να δημιουργήσει το αντικείμενο συνάρτησης.

Ουσιαστικά η lambda δέχεται μια παράμετρο ακολουθούμενη από μια μοναδική έκφραση η οποία γίνεται το σώμα της συνάρτησης και η τιμή αυτής της έκφρασης επιστρέφεται από τη νέα συνάρτηση. Σημειώστε ότι ούτε καν μια εντολή print δε μπορεί να χρησιμοποιηθεί μέσα σε μια lambda, μόνο εκφράσεις.

Η συνάρτηση exec και η συνάρτηση eval

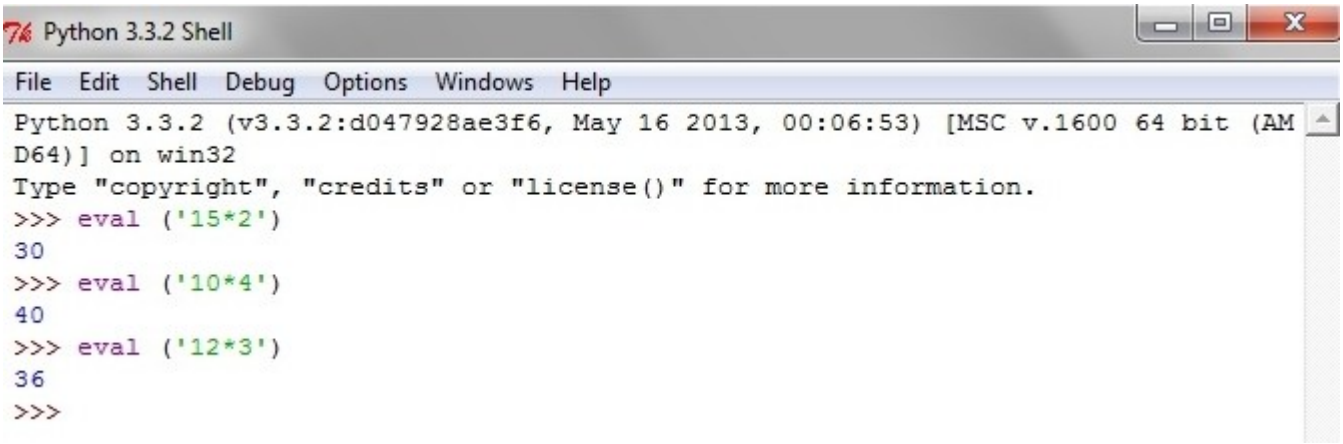
Η **συνάρτηση exec** χρησιμοποιείται για να εκτελέσει εντολές της Python οι οποίες είναι αποθηκευμένες σε μια συμβολοσειρά ή σε ένα αρχείο, αντί να είναι γραμμένες μέσα στο ίδιο το πρόγραμμα.

➔ Για παράδειγμα, μπορούμε να δημιουργήσουμε μια συμβολοσειρά που να περιέχει κώδικα Python κατά την εκτέλεση, και να εκτελέσουμε αυτές τις εντολές χρησιμοποιώντας την εντολή exec.



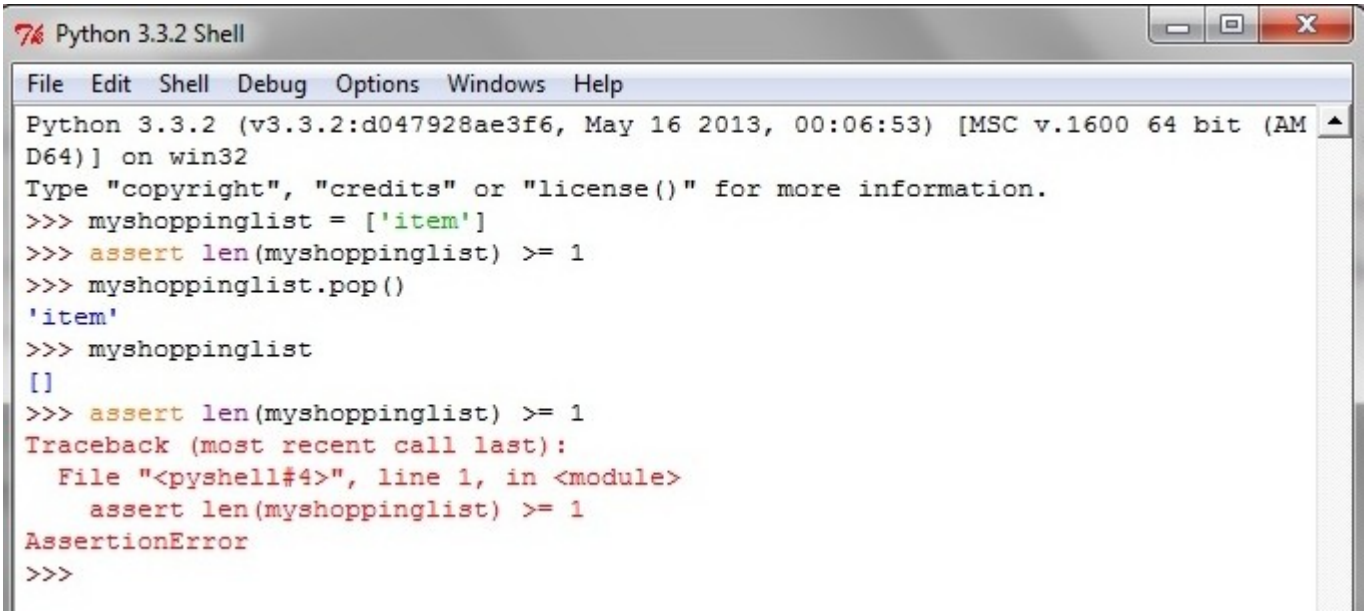
Η **συνάρτηση eval** χρησιμοποιείται για να αποτιμήσει έγκυρες εκφράσεςι της Python οι οποίες αποθηκεύονται σε μια συμβολοσειρά.

➔ Παρακάτω παραθέτω ένα παράδειγμα :



Η εντολή assert

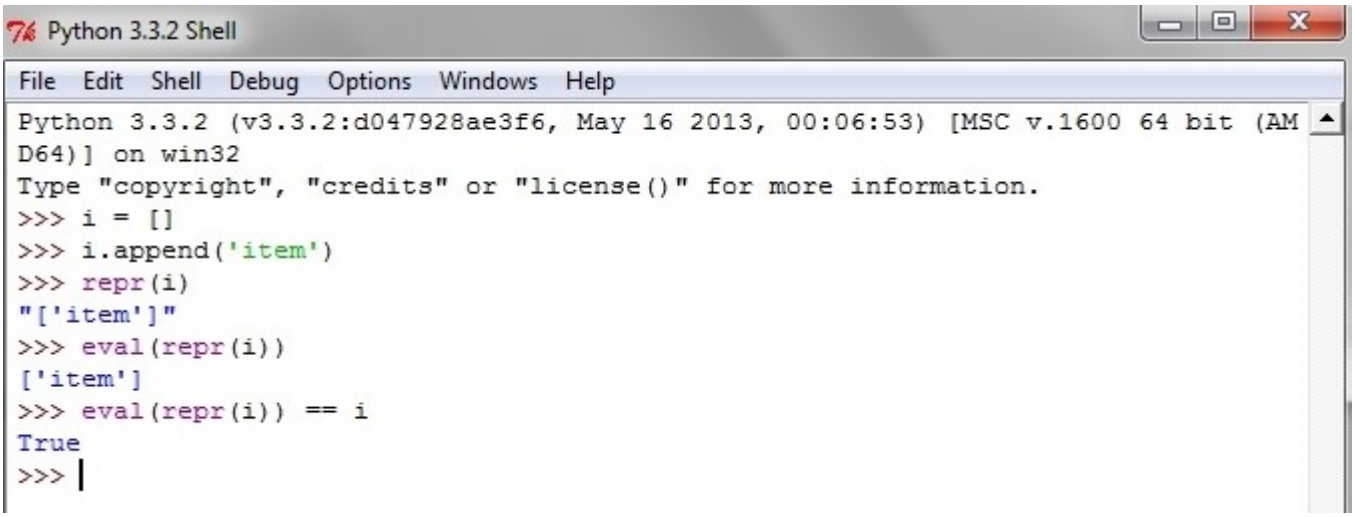
Η εντολή assert χρησιμοποιείται για να δηλώσουμε ότι κάτι είναι αληθές. Για παράδειγμα, αν είστε απόλυτα σίγουροι ότι έχετε τουλάχιστον ένα στοιχείο σε μια λίστα που χρησιμοποιείτε και θέλετε να το ελέγξετε αυτό, και να εγείρετε ένα σφάλμα σε αν δεν είναι αληθές, τότε η εντολή assert είναι ιδανική γι' αυτή την περίπτωση. Αν η εντολή assert αποτύχει, τότε εγείρεται ένα σφάλμα AssertionError.



Η εντολή assert θα πρέπει να χρησιμοποιείται προσεκτικά. Τις περισσότερες φορές είναι καλύτερο να αρπάζετε τις εξαιρέσεις και είτε να χειρίζεστε το πρόβλημα με κάποιο τρόπο, είτε να ενημερώνετε το χρήστη γι' αυτό και να τερματίζετε την εφαρμογή.

Η συνάρτηση repr

Η συνάρτηση repr χρησιμοποιείται για να λάβουμε μια κανονικοποιημένη αναπαράσταση ως συμβολοσειράς του αντικειμένου. Το ενδιαφέρον μέρος είναι ότι θα έχετε eval(repr(object)) == object τις περισσότερες φορές.



Η συνάρτηση repr χρησιμοποιείται για να λάβουμε μια ευανάγνωστη και εκτυπώσιμη αναπαράσταση του αντικειμένου. Μπορείτε να ελέγξετε τι επιστρέφουν οι κλάσεις σας για τη συνάρτηση repr ορίζοντας τη μέθοδο __repr__ στην κλάση.

Κανονικές εκφράσεις

Οι κανονικές εκφράσεις αποτελούν ένα από τα σημαντικότερα εργαλεία που έχουμε στη διάθεσή μας όταν θέλουμε να βρούμε κομμάτια κειμένου, τα οποία μπορούν να εκφραστούν είτε ως αλφαριθμητικά που έχουμε δει, είτε γενικότερα ως κλάσεις αλφαριθμητικών (πχ όλα τα αλφαριθμητικά που αναπαριστουν δεκαδικούς/ακέραιους αριθμούς). Αυτές τις κλάσεις αλφαριθμητικών τις ονομάζουν **πρότυπα (pattern)**.

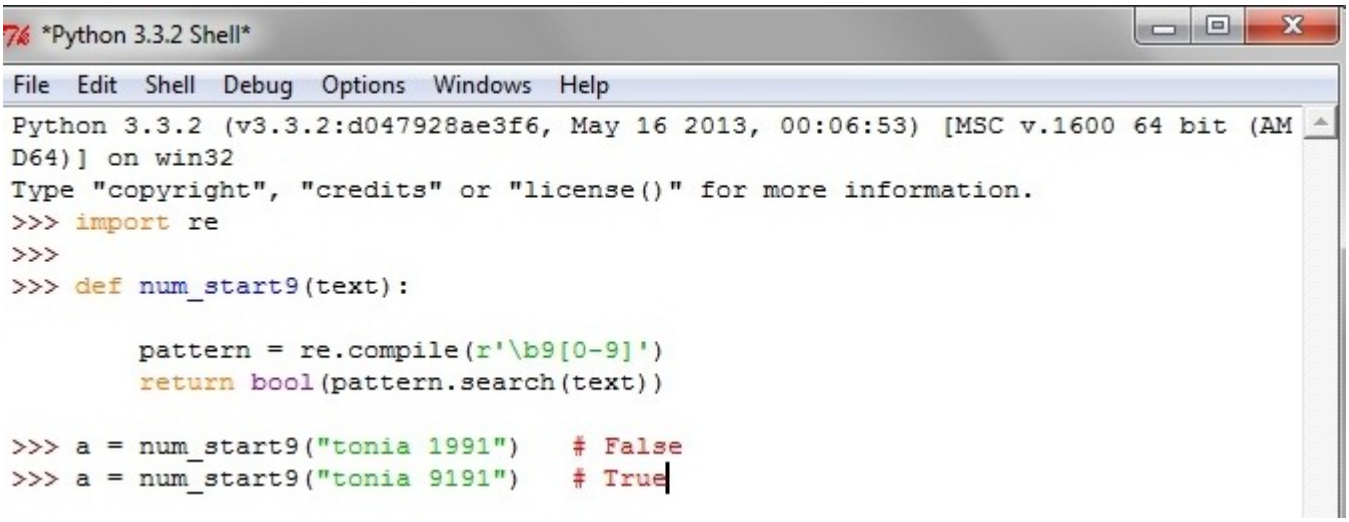
Υπάρχουν δυο κύριες χρήσεις των κανονικών εκφράσεων στην Python : **η αναζήτηση και το ταίριασμα.** Στην αναζήτηση, μας ενδιαφέρει να βρούμε την πρώτη φορά που συναντάμε ένα πρότυπο. Στο ταίριασμα, βρίσκουμε όλες θέσεις όπου μπορεί να βρίσκεται το πρότυπο που αναζητούμε.

Αναζήτηση

Χρησιμοποιώντας τις τετράγωνες αγκύλες μπορούμε να ορίσουμε μια κλάση από χαρακτήρες που μπορεί να ταιριάξει με κάποιον χαρακτήρα. Στο ακόλουθο παράδειγμα ορίζουμε την κλάση με τους χαρακτήρες που αντιστοιχούν σε αριθμούς.

Αντίθετα, όλοι οι χαρακτήρες που βρίσκονται εκτός των αγκυλών μπορούν να ταιριάξουν μόνο αν βρεθεί ακριβώς ο ίδιος χαρακτήρας στο αλφαριθμητικό στο οποίο ψάχνουμε.

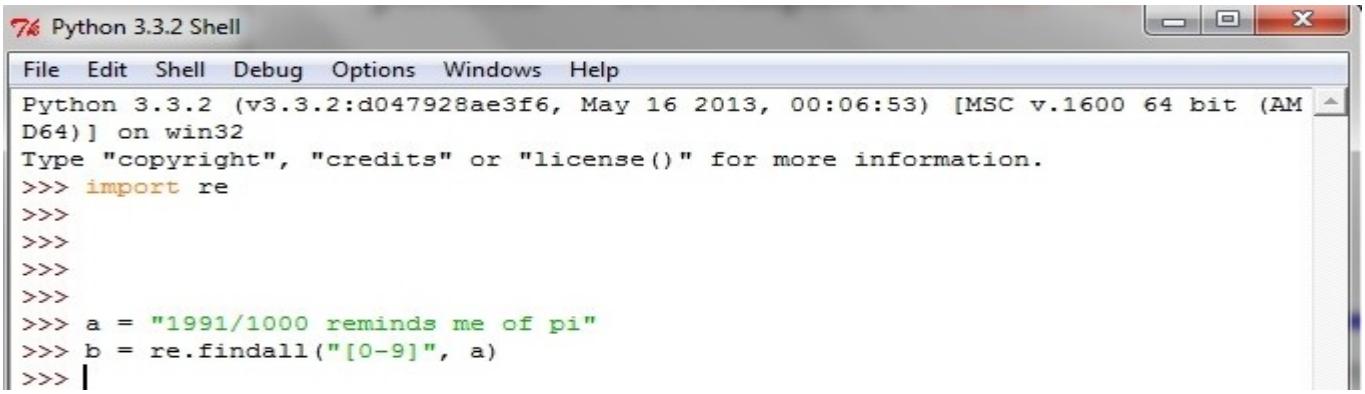
➔ Στο παράδειγμα που ακολουθεί, ψάχνουμε αριθμούς που ξεκινάνε με 9.



Μπορούμε να παρατηρήσουμε πως χρησιμοποιήθηκε ο ειδικός χαρακτήρας \b. Ο ειδικός αυτός χαρακτήρας ταιριάζει όρια των λέξεων. Κατά αυτό τον τρόπο, σιγουρεύουμε ότι θα βρούμε αριθμούς που αρχίζουν από 9 όπως το 9191 και όχι αριθμούς που απλά περιέχουν το 9 όπως το 1991.

Αντίστοιχα λειτουργεί και η **συνάρτηση findall** η οποία όμως επιστρέφει όλες τις περιπτώσεις όπου βρέθηκε στο αλφαριθμητικό στο οποίο ψάχνουμε το πρότυπο.

➔ Στο παρακάτω παράδειγμα, επιστρέφονται όλοι οι αριθμοί που βρίσκονται μέσα στο αλφαριθμητικό.



GUI με tkinter

Με τον όρο GUI αναφερόμαστε στην **γραφική διεπαφή (Graphical User Interface)** που έχει η εφαρμογή μας με τον χρήστη. Για τον σκοπό αυτό υπάρχουν αρκετές βιβλιοθήκες που λειτουργούν σε διαφορετικές πλατφόρμες χωρίς αλλαγή στον κώδικα τους. Μερικές από αυτές είναι:

→ **wxPython**
→ **pyQT**
→ **PyGTK**
→ **tkinter**

Τις τρεις πρώτες ίσως να τις γνωρίζετε και από άλλες γλώσσες, καθώς έχουν γίνει οι απαραίτητες ενέργειες γύρω από τον κώδικα των αντίστοιχων βιβλιοθηκών ώστε να μπορούν να χρησιμοποιηθούν από το περιβάλλον της γλώσσας Python. Αντίθετα, η tkinter (= Tk interface) έρχεται μαζί με την Python, είναι σχετικά απλή ενώ και δεν θα χρειαστούμε καμία επιπλέον βιβλιοθήκη για να την χρησιμοποιήσουμε. Χαρακτηρίζεται για την απλότητα της και στηρίζεται πάνω σε Tcl/Tk. Το tkinter προσφέρει μια αντικειμενοστραφή διεπαφή προς την εργαλειοθήκη Tcl/Tk.

Στο παρόν project θα εξετάσουμε την **tkinter** .

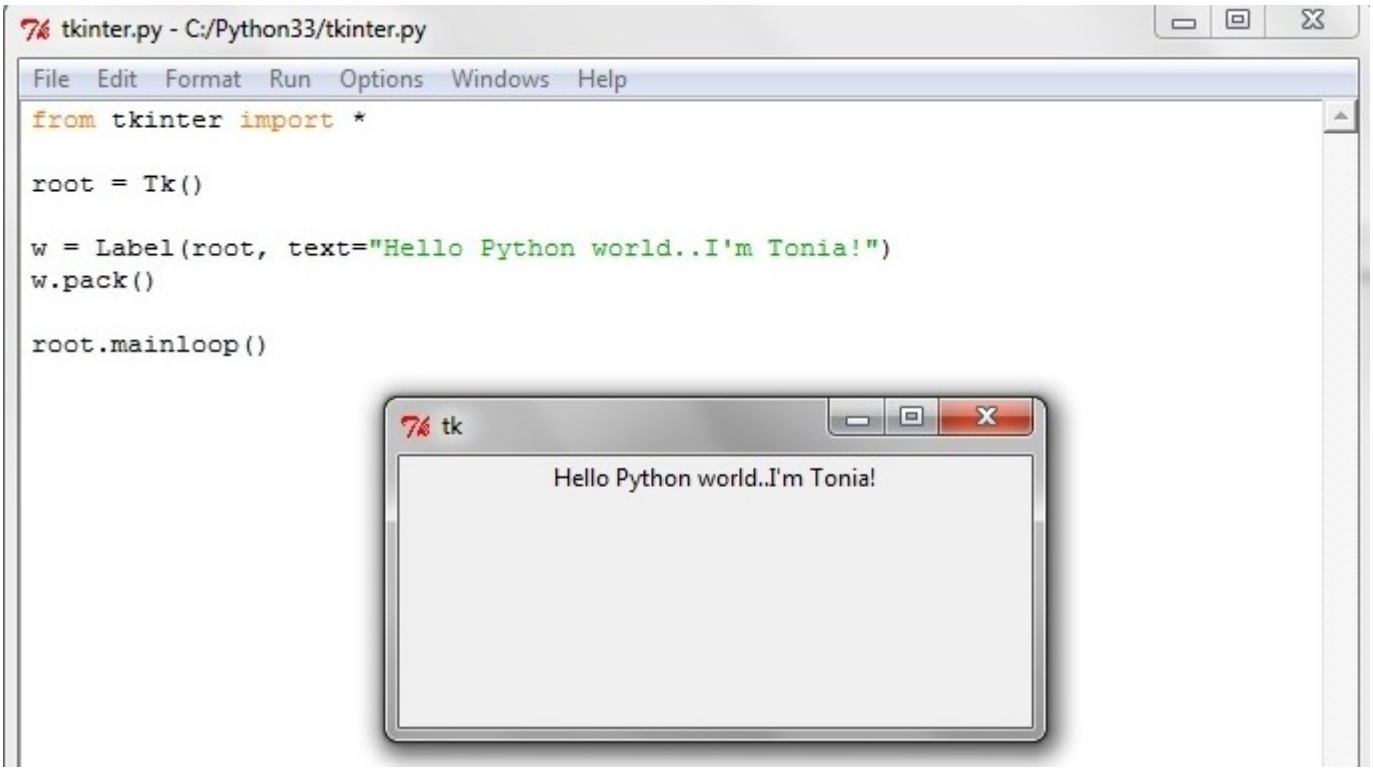
Tkinter Προγραμματισμός

Η Tkinter είναι η πρότυπη βιβλιοθήκη GUI για την Python. Η Python όταν συνδυάζεται με την Tkinter προσφέρει ένα γρήγορο και εύκολο τρόπο για τη δημιουργία GUI εφαρμογών. Η Tkinter παρέχει μια ισχυρή αντικειμενοστραφή διεπαφή (object-oriented interface) στη Tk GUI εργαλειοθήκη.

Το να δημιουργήσεις μια GUI εφαρμογή που χρησιμοποιεί Tkinter είναι ένα εύκολο έργο. Το μόνο που χρειάζεται να κάνετε είναι να εκτελέσετε τα παρακάτω βήματα:

1. Εισαγωγή της Tkinter μονάδας
2. Δημιουργία κυρίου παραθύρου της εφαρμογής GUI.
3. Προσθήκη ενός ή περισσότερων από τις προαναφερθείσες widgets στην εφαρμογή GUI.
4. Εισαγωή του κύριου βρόγχου εκδήλωσης για να αναλάβει δράση εναντίον κάθε συμβάντος που ενεργοποιείται από το χρήστη.

➔ Το πιο απλό πρόγραμμα που μπορούμε να δημιουργήσουμε είναι :



Παρατηρούμε πως αφού εισάγουμε από την βιβλιοθήκη tkinter ό,τι χρειαζόμαστε, μπορούμε να δημιουργήσουμε ένα απλό παράθυρο καλώντας τον δημιουργό της κλάσης. Στην συνέχεια προσθέτουμε μια ετικέτα (label) την οποία εμφανίζουμε με την συνάρτηση pack(). Τέλος, το πρόγραμμα μας εισέρχεται σε έναν βρόγχο (mainloop) όπου περιμένει από τον χρήστη ένα καινούργιο συμβάν.

Tkinter Widgets (Γραφικά στοιχεία)

Η Tkinter παρέχει διάφορα στοιχεία ελέγχου, όπως κουμπιά, ετικέτες και πλαίσια κειμένου, που χρησιμοποιούνται σε μια εφαρμογή GUI. Οι έλεγχοι αυτοί συνήθως ονομάζονται **widgets (γραφικά στοιχεία)**.

Υπάρχουν επί του παρόντος 15 τύποι γραφικών στοιχείων (widgets) στην Tkinter. Παραθέτω αυτά τα γραφικά στοιχεία, καθώς και μια σύντομή περιγραφή στον πίνακα που ακολουθεί :

Χειριστής	Περιγραφή
Κουμπί	Το widget κουμπί χρησιμοποιείται για να εμφανίσετε τα κουμπιά στην εφαρμογή σας.
Καμβάς	Ο widget καμβάς χρησιμοποιείται για τη σχεδίαση σχημάτων, όπως γραμμές, ελλείψεις, πολύγωνα, και ορθογώνια, στην εφαρμογή σας.
Checkbutton	Το Checkbutton widget χρησιμοποιείται για να εμφανίσει μια σειρά από επιλογές, όπως πλαίσια ελέγχου. Ο χρήστης μπορεί να

	επιλέξει πολλαπλές επιλογές σε μια στιγμή.
Είσοδος	Το widget εισόδου χρησιμοποιείται για την εμφάνιση μονής γραμμής πεδίο κειμένου για την αποδοχή των τιμών από ένα χρήστη.
Πλαίσιο	Το widget Frame χρησιμοποιείται ως ένα widget δοχείο για να οργανώσει άλλες μικροσυσκευές.
Επιγραφή	Η widget επιγραφή χρησιμοποιείται για να παρέχει μια ενιαία γραμμή λεζάντα για άλλα widgets. Μπορεί επίσης να περιέχει εικόνες.
Listbox	Το widget πλαίσιο λίστας χρησιμοποιείται για να παρέχει μια λίστα επιλογών σε ένα χρήστη.
ΠλήκτροMENU	Το widget πλήκτρο MENU χρησιμοποιείται για να εμφανίζει μενού στην εφαρμογή σας.
Μενού	Το widget Menu χρησιμοποιείται για να παρέχει διάφορες εντολές σε ένα χρήστη. Αυτές οι εντολές περιέχονται μέσα στο πλήκτρο MENU.
Μήνυμα	Το widget Μήνυμα χρησιμοποιείται για την εμφάνιση πολλών γραμμών πεδία κειμένου για την αποδοχή των τιμών από ένα χρήστη.
RadioButton	Το widget RadioButton χρησιμοποιείται για να εμφανίσει μια σειρά από επιλογές, όπως κουμπιά επιλογής. Ο χρήστης μπορεί να επιλέξει μόνο μία επιλογή σε μια στιγμή.
Κλίμακα	Η κλίμακα widget χρησιμοποιείται για να παρέχει ένα widget slider.
Κύλισης	Η Scrollbar widget χρησιμοποιείται για να προσθέσετε την ικανότητα κύλισης σε διάφορα widgets, όπως στα πλαίσια λίστας.
Κείμενο	Το widget κειμένου χρησιμοποιείται για την εμφάνιση κειμένου σε πολλαπλές γραμμές.
Toplevel	Το γραφικό toplevel χρησιμοποιείται για να παρέχει ένα ξεχωριστό window container.
Spinbox	Το widget Spinbox είναι μια παραλλαγή του προτύπου έναρξης Tkinter widget, το οποίο μπορεί να χρησιμοποιηθεί για να επιλέξετε από ένα σταθερό αριθμό τιμών.
PanedWindow	Ένα PanedWindow είναι ένα widget container που μπορεί να περιέχει οποιοδήποτε αριθμό των υαλοπινάκων, που διοργανώνονται οριζόντια ή κάθετα.
LabelFrame	Ένα labelframe είναι ένα απλό widget δοχείο. Πρωταρχικός σκοπός του είναι να δρα ως διαχωριστικό ή ως container για πολύπλοκες διατάξεις παράθυρο.
tkMessageBox	Χρησιμοποιείται για την εμφάνιση πλαισίων μηνυμάτων σε εφαρμογές σας.

Διαχείριση Γεωμετρίας

Όλα τα Tkinter widgets έχουν πρόσβαση σε συγκεκριμένες μεθόδους διαχείρισης της γεωμετρίας, που έχουν ως σκοπό την οργάνωση widgets σε όλη την περιοχή του widget γονέα. Η Tkinter εκθέτει τις ακόλουθες κατηγορίες διαχείρισης γεωμετρίας : **pack, grid, και place.**

- ✓ Το πακέτο () Μέθοδος - Αυτός ο διαχειριστής γεωμετρίας οργανώνει τα widgets σε τεμάχια πριν την τοποθέτησή τους στο widget γονέα.
- ✓ Το πλέγμα () Μέθοδος - Αυτός ο διαχειριστής γεωμετρίας οργανώνει τα widgets σε έναν πίνακα-όπως δομή στο widget γονέα.
- ✓ Η θέση () Μέθοδος -Αυτός ο διαχειριστής γεωμετρίας οργανώνει τα widgets με την τοποθέτηση τους σε μια συγκεκριμένη θέση στο widget γονέα.

Αναδυόμενα παράθυρα διαλόγου

Τα αναδυόμενα παράθυρα διαλόγου pop-up μπορούν να χρησιμοποιηθούν για να ενημερώσουν τον χρήστη για ένα συμβάν, ή για να του ζητήσουν μια απλή επιλογή. Μπορούμε να τα δούμε σαν αντικαταστάτες των print που χρησιμοποιούν τα προγράμματα σε κονσόλα ώστε να ενημερώσουν σε συντομία για ένα γεγονός που συνέβη (και συνήθως διακόπτει την ροή του προγράμματος).

΄Εχουμε στην διάθεση μας επτά διαφορετικούς τύπους συνηθισμένων παραθύρων που μπορούμε εύκολα να χρησιμοποιήσουμε, αρκεί να εισάγουμε το **`tkinter.messagebox`**. Το αποτέλεσμα τους, εμφανίζεται στην αντίστοιχη εικόνα της κάθε συνάρτησης που αντιστοιχεί στα αναδυόμενα παράθυρα.

1. showinfo()
2. askokcancel()
3. showwarning()
4. showerror()
5. askquestion()
6. askyesno()
7. askretrycancel()

Τα ορίσματα που μπορούμε να περάσουμε με σειρά στις παραπάνω συναρτήσεις είναι:

- ✓ Τίτλος: Ο τίτλος του παραθύρου
- ✓ Μήνυμα: Το κείμενο που θα εμφανιστεί ως μήνυμα
- ✓ Επιλογές: Προαιρετικά. Χρησιμοποιείται για την επιλογή εικονιδίου, ποιο κουμπί να είναι προεπιλογή και άλλα.

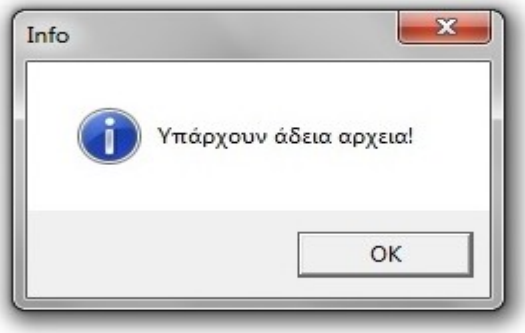
Ας δούμε στην πράξη πως εφαρμόζονται οι παραπάνω συναρτήσεις …

showinfo()

showinfo.py - C:/Python33/showinfo.py

File Edit Format Run Options Windows Help

```
# filename: showinfo.py

from tkinter import*
from tkinter.messagebox import*

showinfo('Info', 'Υπάρχουν άδεια αρχεια!')
```



askokcancel()

askokcancel.py - C:/Python33/askokcancel.py

File Edit Format Run Options Windows Help

```
# filename: askokcancel.py

from tkinter import*
from tkinter.messagebox import*

askokcancel('Tell me!', 'Would you like to quit now?')
```

showwarning()

showwarning.py - C:/Python33/showwarning.py

File Edit Format Run Options Windows Help

```
# filename: showwarning.py

from tkinter import*
from tkinter.messagebox import*

showwarning('Warning!!', 'A new window detected!')
```

showerror()

showerror.py - C:/Python33/showerror.py

File Edit Format Run Options Windows Help

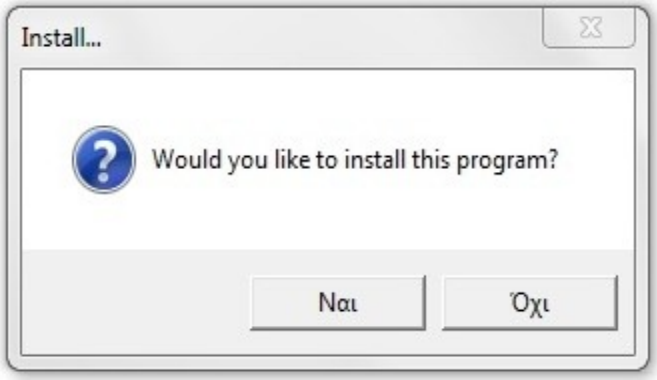
```
# filename: showerror.py

from tkinter import*
from tkinter.messagebox import*

showerror('Error!', 'So many windows have been detected!')
```

askquestion()

askquestion.py - C:/Python33/askquestion.py

File Edit Format Run Options Windows Help

```
# filename: askquestion.py

from tkinter import*
from tkinter.messagebox import*

askquestion('Install...', 'Would you like to install this program?')
```



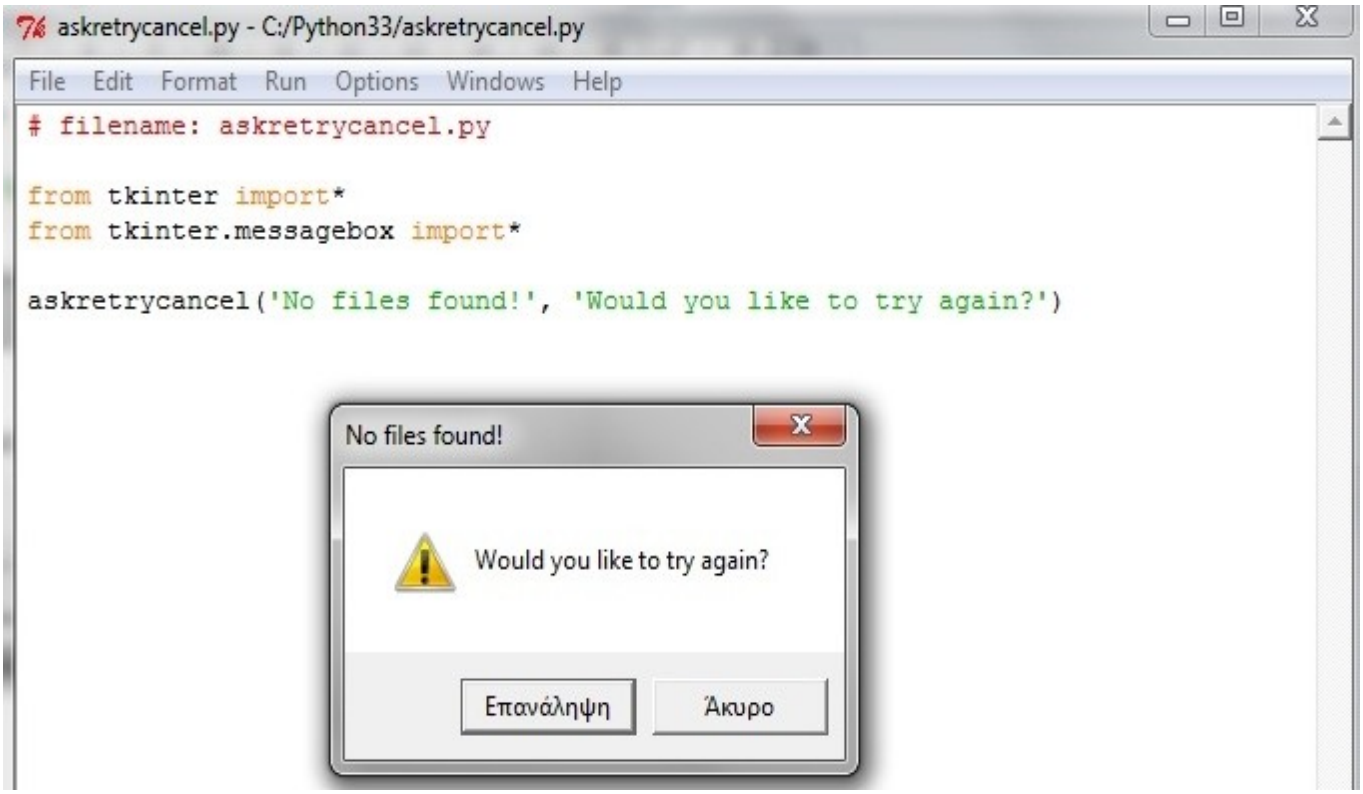
askyesno()

askyesno.py - C:/Python33/askyesno.py

File Edit Format Run Options Windows Help

```
# filename: askyesno.py

from tkinter import*
from tkinter.messagebox import*

askyesno('Please tell me...', 'Would you like run this program?')
```

askretycancel()



Παρατήρηση → Στα παράθυρα όπου ο χρήστης έχει την δυνατότητα να διαλέξει κάποια επιλογή (πχ OK, ή άκυρο), ανάλογα με το τι διαλέγει επιστρέφεται από την συνάρτηση η τιμή true ή false αντίστοιχα.

Standard attributes

Ας μελετήσουμε το πώς μερικά από τα common attributes (όπως το μέγεθος, το χρώμα και η γραμματοσειρά) ορίζονται.

- ✓ Dimensions (διαστάσεις)
- ✓ Colors (χρώματα)
- ✓ Fonts (γραμματοσειρές)
- ✓ Anchors (άγκυρες)
- ✓ Relief styles (στυλ αρωγής)
- ✓ Bitmaps
- ✓ Cursors (κέρσορες)

Διαστάσεις (Dimensions)

Διάφορα μήκη, πλάτη, και άλλες διαστάσεις των widgets μπορούν να περιγραφούν σε πολλές διαφορετικές μονάδες.

→ Αν ορίσετε μια διάσταση σε έναν ακέραιο, θεωρείται ότι αυτό είναι σε pixels.
→ Μπορείτε να καθορίσετε τις μονάδες με τη διάσταση σε ένα string που περιέχει έναν αριθμό ακολουθούμενο από :
- c → centimeters
- i → inches
- m → milimeters
- p → Printer's points (about 1/72")

Η tkinter εκφράζει ένα μήκος ως έναν ακέραιο αριθμό από pixels. Παρακάτω δίνεται η λίστα των κοινών επιλογών μήκους :

- borderwidth → το πλάτος των συνόρων το οποίο δίνει μια τρισδιάστατη όψη στο widget.
- Highlightthickness → το πλάτος του τονισμένου ορθογωνίο, όταν το widget έχει εστίαση.
- padX Pady → Extra χώρο τα αιτήματα widget από το διαχειριστή του σχεδίου του πέρα από το ελάχιστο που χρειάζεται το widget για να εμφανίσει τα περιεχόμενά του στην x και y.
- Selectborderwidth → Πλάτος του τρισδιάστατο περίγραμμα γύρω από επιλεγμένα στοιχεία του widget.
- Wraplength → Μέγιστο μήκος γραμμής για τα widgets που εκτελούν αναδίπλωση λέξεων.
- Height → επιθυμητό ύψος του widget, πρέπει να είναι μεγαλύτερη από ή ίσο με 1.
- Underline → Δείκτης του χαρακτήρα για να υπογραμμίσει το κείμενο του widget (0 είναι ο πρώτος χαρακτήρας, 1 το δεύτερο, και ούτω καθεξής).
- Width → Επιθυμητό πλάτος του widget

Χρώματα (Colors)

Η Tkinter αντιπροσωπεύει τα χρώματα με strings. Υπάρχουν δύο γενικοί τρόποι για να καθορίσετε τα χρώματα στην Tkinter:

1. Μπορείτε να χρησιμοποιήσετε ένα string διευκρινίζοντας το ποσοστό του κόκκινου, πράσινου και μπλε σε δεκαεξαδικά ψηφία. Για παράδειγμα, "# fff" είναι λευκό, "# 000000" είναι μαύρο, "# 000fff000" είναι καθαρό πράσινο, και "# 00ffff" είναι καθαρό κυανό (πράσινο συν μπλε).

2. Μπορείτε επίσης να χρησιμοποιήσετε οποιοδήποτε τοπικά καθορισμένο πρότυπο όνομα χρώματος. Δηλαδή, τα χρώματα "λευκό", "μαύρο", "κόκκινο", "πράσινο", "μπλε", "κυανό", "κίτρινο" θα είναι πάντα διαθέσιμα.

Χρωματικές επιλογές

Οι κοινές επιλογές χρωμάτων είναι:

→ activebackground →χρώμα φόντου για το widget, όταν το widget είναι ενεργό.
→ activeforeground →Χρώμα προσκηνίου για το widget, όταν το widget είναι ενεργό.
→ background →χρώμα φόντου για το widget. Αυτό μπορεί επίσης να παρασταθεί ως bg .
→ disabledforeground → Χρώμα προσκηνίου για το widget, όταν το widget είναι απενεπεργοποιημένο.
→ foreground →ο χρώμα προσκηνίου για το widget. Αυτό μπορεί επίσης να παρασταθεί ως fg.
→ highlightbackground → Χρώμα φόντου της τονισμένης περιοχής τονίζουν όταν το widget έχει εστίαση (focus).
→ highlightcolor →Χρώμα προσκηνίου της τονισμένης περιοχής ,όταν το widget έχει εστίαση.
→ selectbackground →χρώμα φόντου για τα επιλεγμένα στοιχεία του widget.
→ selectforeground → Χρώμα προσκηνίου για τα επιλεγμένα στοιχεία του widget.

Γραμματοσειρές (fonts)

Υπάρχουν έως και τρεις τρόποι για να προσδιορίσετε το type style.

1) Simple Tuple Fonts (Απλές γραμματοσειρές πλειάδων)

Ως μια πλειάδα της οποίας το πρώτο στοιχείο είναι η οικογένεια γραμματοσειρών,ακολουθείται προαιρετικά από ένα string που περιέχει ένα ή περισσότερα από τα style modifiers έντονη, πλάγια ή υπογράμμισμένη.

Παράδειγμα :

```
("Times New Roman", "14")
```

```
("Consolas","20","bold italic")
```

2) Γραμματοσειρές αντικειμένου font (Font object Fonts)

Μπορείτε να δημιουργήσετε ένα "αντικείμενο γραμματοσειρά" από την εισαγωγή της μονάδας tkFont και χρησιμοποιώντας Font κατασκευαστή της κατηγορίας:

```
import tkFont

font = tkFont.Font(επιλογή ,  ...)
```

Εδώ είναι η λίστα των επιλογών :

→ family : Το όνομα της οικογένειας γραμματοσειράς ως string.
→ size : Το ύψος της γραμματοσειράς ως ακέραιος στα σημεία. Για να πάρετε μια γραμματοσειρά n pixels, χρησιμοποιείτε -n.
→ weight : "bold" για έντονη γραφή, «regular» για κανονική.
→ slant : "italic" για την πλάγια γραφή, "roman" για unslanted.
→ underline : 1 για το υπογραμμισμένο κείμενο, 0 για την κανονικό.
→ overstrike : 1 για το κείμενο overstruck, 0 για την κανονικό.

Παράδειγμα :

```
Helv22 =
tkFont.Font(family="Helvetica",size=22,weight="bold",slant=
"italic")
```

3) X Window Fonts

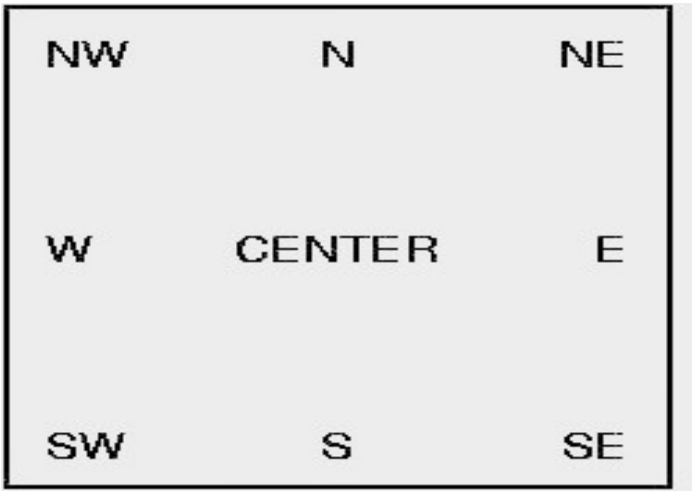
Αν δουλεύετε σε X Window System, μπορείτε να χρησιμοποιήσετε οποιοδήποτε από τα ονόματα των γραμματοσειρών X. Χρησιμοποιήστε το ***xfontsel πρόγραμμα*** ,το οποίο θα σας βοηθήσει να επιλέξετε κατάλληλες και ευχάριστες γραμματοσειρές.

Anchors

Οι άγκυρες χρησιμοποιούνται για να καθορίσουν που τοποθετείται το κείμενο σε σχέση με ένα σημείο αναφοράς. Ακολουθεί η λίσυα από τις πιθανές σταθερές οι οποίες μπορούν να χρησιμοποιηθούν για το γνώρισμα της άγκυρας(anchor attribute).

NW	CENTER	SE
N	E	
NE	SW	
W	S	

Οι σταθερές άγκυρες (constant anchors) φαίνονται στο παρακάτω διάγραμμα :



Python Tkinter Relief styles

Το relief style ενός widget αναφέρεται σε ορισμένες προσομοιώσεις 3D εφέ γύρω από το εξωτερικό του widget.

Ακολουθεί ένας κατάλογος από πιθανές σταθερές οι οποίες μπορούν να χρησιμοποιηθούν για το relief attribute.

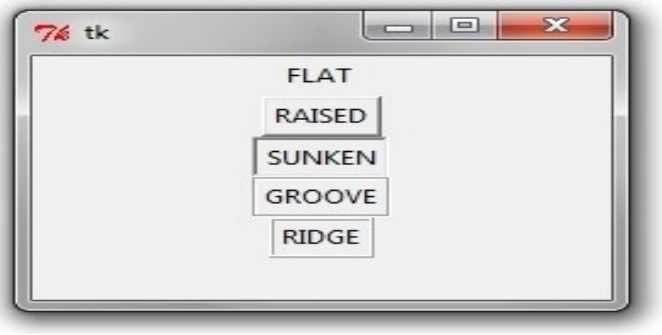
FLAT **SUNKEN**
GROOVE **RIDGE**
RAISED

➔ Ακολουθεί ένα παράδειγμα :

relief.py - C:/Python33/relief.py

File Edit Format Run Options Windows Help

```
# filename: relief.py

from tkinter import *
import tkinter

top = tkinter.Tk()

B1 = tkinter.Button(top, text ="FLAT", relief=FLAT )
B2 = tkinter.Button(top, text ="RAISED", relief=RAISED )
B3 = tkinter.Button(top, text ="SUNKEN", relief=SUNKEN )
B4 = tkinter.Button(top, text ="GROOVE", relief=GROOVE )
B5 = tkinter.Button(top, text ="RIDGE", relief=RIDGE )

B1.pack()
B2.pack()
B3.pack()
B4.pack()
B5.pack()
top.mainloop()
```



Python Tkinter Bitmaps

Μπορείτε να χρησιμοποιήσετε αυτό το χαρακτηριστικό για να εμφανιστεί ένα bitmap. Υπάρχουν οι ακόλουθοι τύποι bitmaps διαθέσιμοι :

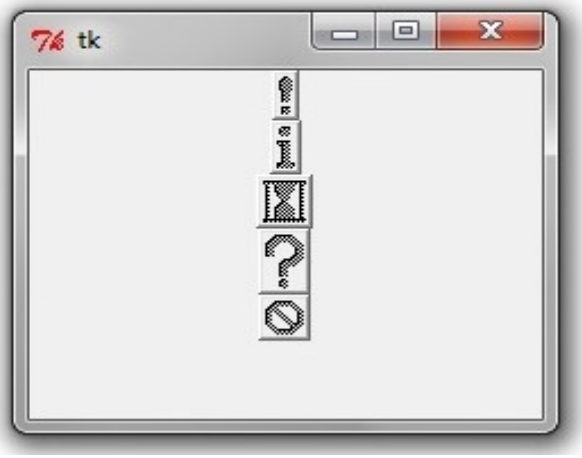
"error"	"hourglass"
"gray75"	"info"
"gray50"	"questhead"
"gray25"	"question"
"gray12"	"warning"

➔ Ακολουθεί παράδειγμα με bitmaps.

bitmaps.py - C:/Python33/bitmaps.py

File Edit Format Run Options Windows Help

```
# filename: bitmaps.py

from tkinter import *
import tkinter

top = tkinter.Tk()

B1 = tkinter.Button(top, text ="warning", relief=RAISED,\
                    bitmap="warning")
B2 = tkinter.Button(top, text ="info", relief=RAISED,\
                    bitmap="info")
B3 = tkinter.Button(top, text ="hourglass", relief=RAISED,\
                    bitmap="hourglass")
B4 = tkinter.Button(top, text ="question", relief=RAISED,\
                    bitmap="question")
B5 = tkinter.Button(top, text ="error", relief=RAISED,\
                    bitmap="error")
B1.pack()
B2.pack()
B3.pack()
B4.pack()
B5.pack()
top.mainloop()
```



Python Tkinter Cursors

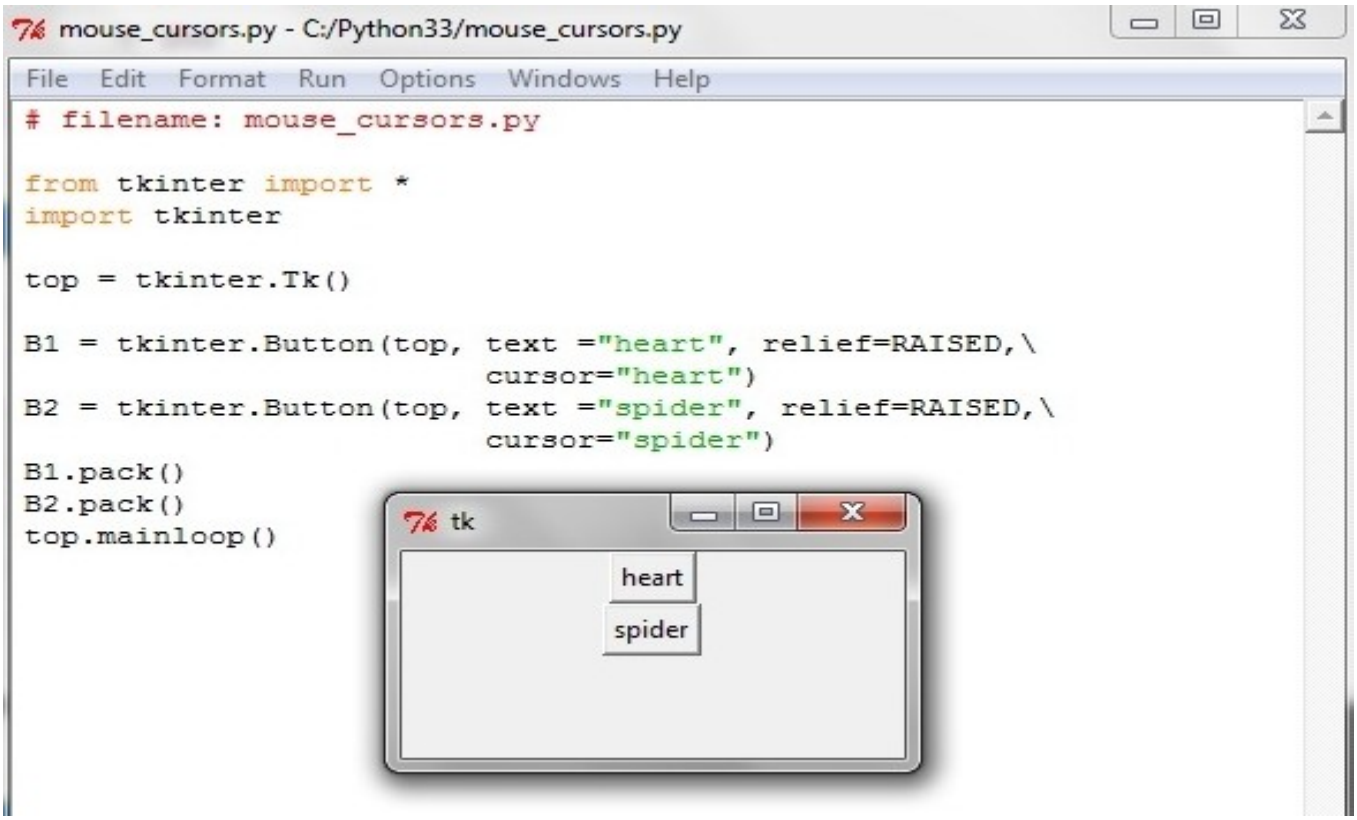
Η Python Tkinter υποστηρίζει ένα μεγάλο αριθμό διαφορετικών mouse cursors. Η ακριβής γραφική απεικόνιση μπορεί να διαφέρει ανάλογα με το λειτουργικό σας σύστημα.

Στη συνέχεια παρατίθεται μια λίστα με τα πιο ενδιαφέροντα mouse cursors :

"arrow"	"pirate"
"circle"	"plus"
"clock"	"shuttle"
"cross"	"sizing"
"dotbox"	"spider"
"exchange"	"spraycan"
"fleur"	"star"
"heart"	"target"
"man"	"tcross"

"mouse"	"trek" "watch"

➔ Ακολουθεί ένα παράδειγμα με mouse cursors :



Παρατήσηση → Αν πλησιάσουμε στον κέρσορα πάνω στο εικονίδιο heart , αυτός παίρνει το σχήμα καρδιάς και πάνω στο spider, παίρνει το σχήμα αράχνης.

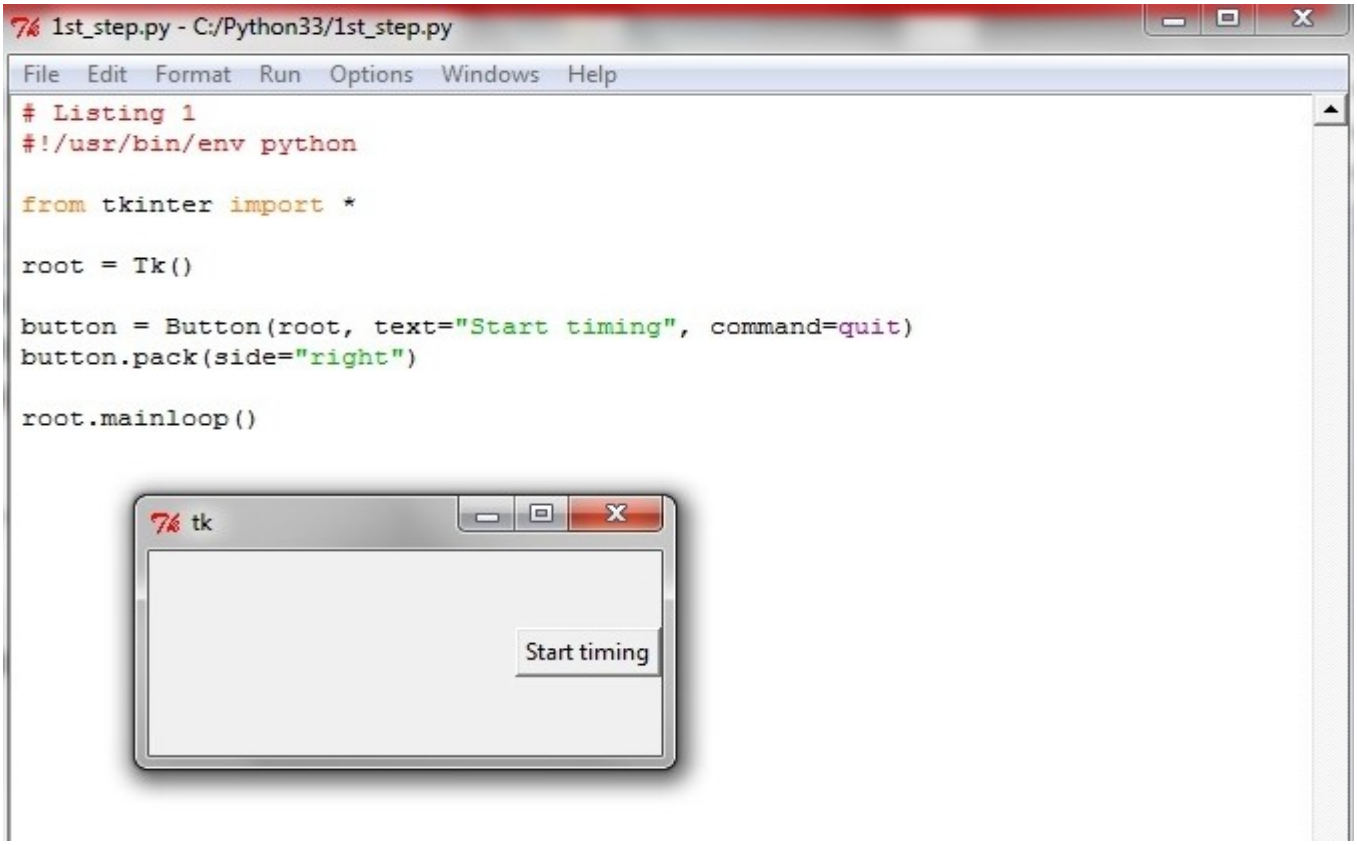
➔ Στη συνέχεια θα δούμε την δημιουργία μιας εφαρμογής σε Python ,ενός χρονομέτρου.

Ένα απλό μονό κουμπί δεν είναι και τόσο χρήσιμο.Ένα χρονόμετρο, θα πρέπει να είναι σε θέση να καθορίσει πότε θα σας προειδοποιήσει. Και γι 'αυτό, η κλίμακα(Scale) είναι μια καλή επιλογή. Σας δίνει ένα slider που μπορείτε να χρησιμοποιήσετε για να επιλέξετε πόσα λεπτά θέλετε να περιμένει το χρονόμετρο. Προσθέστε αυτές τις δύο γραμμές στο script σας ακριβώς πριν από τη γραμμή όπου θα δημιουργήσετε το κουμπί σας:

```
scale = Scale(root, from_=1, to=20,
orient=HORIZONTAL, length=500)

scale.pack()
```

Μπορούμε λοιπόν να αναπτύξουμε την εφαρμοφή μας ΣΤΑΔΙΑΚΑ ,προσθέτοντας επιπλέον δυνατότητες ως εξής:

1ο στάδιο

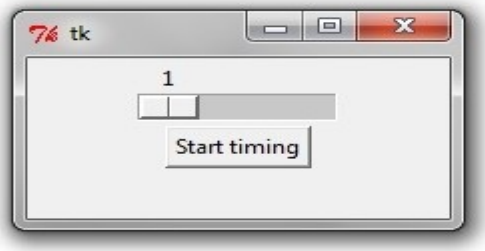


2ο στάδιο

2nd_step.py - C:/Python33/2nd_step.py

File Edit Format Run Options Windows Help

```
# Listing 2
#!/usr/bin/env python

from tkinter import *

root = Tk()

scale = Scale(root, from_=1, to=20, orient=HORIZONTAL)
scale.pack()

button = Button(root, text="Start timing", command=quit)
button.pack()

root.mainloop()
```

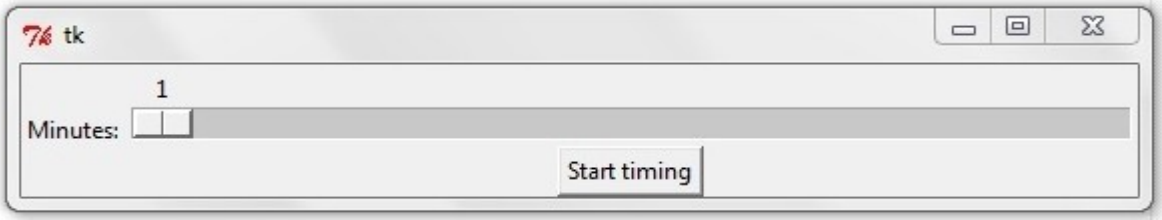


3ο στάδιο

3rd_step.py - C:/Python33/3rd_step.py

File Edit Format Run Options Windows Help

```
# Listing 3
#!/usr/bin/env python

from tkinter import *

root = Tk()

minutes = Label(root, text="Minutes:")
minutes.pack(side=LEFT)

scale = Scale(root, from_=1, to=20, orient=HORIZONTAL, length=500)
scale.pack()

button = Button(root, text="Start timing", command=quit)
button.pack()

root.mainloop()
```



Σημείωση→ Μπορείτε επίσης να προσθέσετε στοιχεία ελέγχου για να έχετε καλύτερη γραφική αναπαράσταση. Για παράδειγμα, μπορείτε να προσθέσετε μια ετικέτα λίγο πριν τη δημιουργία της κλίμακας,η οποία να διευκρινίζει ότι το ρυθμιστικό σημαίνει ένα χρόνο σε λεπτά(minutes)[3ο στάδιο].

«Χτίσιμο» μιας βασικής GUI εφαρμογής(GUI application) βήμα-βήμα σε Python με Tkinter

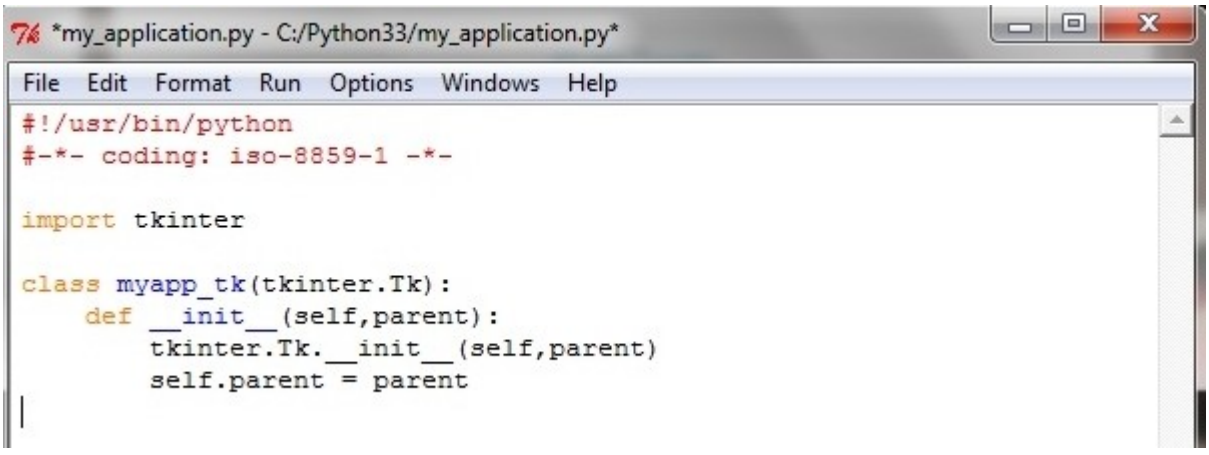
Ένα widget είναι ένα στοιχείο GUI (ένα κουμπί, ένα πλαίσιο ελέγχου, ένα σήμα, μια καρτέλα, ένα παράθυρο). Η παρούσα εφαρμογή θα έχει 3 widgets:

1. Ένα πεδίο κειμένου ("Παρακαλώ εισάγετε κείμενο..")
2. Ένα κουμπί ("Πιέστε αυτό το κουμπί!")
3. Ένα μπλε ετικέτα («Γειά σου,αυτή είναι η πρώτη μου GUI application!")

Πως θέλω να λειτουργεί η εφαρμογή μου :

1) Όταν πατάμε το ENTER στο πεδίο κειμένου, ή κάνουμε κλικ στο κουμπί, η μπλε ετικέτα θα εμφανίζει το κείμενο που έχει εισαχθεί.
2) Θέλουμε να περιορισμό αλλαγής μεγέθους παραθύρου, έτσι ώστε το παράθυρο να μπορεί να αλλάξει το μέγεθός του οριζόντια.
3) Εάν το παράθυρο έχει αλλάξει μέγεθος, το πεδίο κειμένου και κόκκινο περίγραμμα θα επεκταθεί οριζόντια.

1ο βήμα – Δημιουργία κλάσης και κατασκευαστή-παρακολούθηση γονέα (parent)



Παρατηρήσεις

→ Είναι καλό και πρακτικό να θέσουμε την εφαρμογή μας σε μια κλάση.
→ tkinter.Tk : είναι η βασική κλάση για πρότυπα παράθυρα.

→ myapp_tk : προέρχεται από tkinter.Tk , οπότε θα πρέπει να καλέσουμε τον κατασκευαστή tkinter.Tk (tkinter.Tk.__init__ ())

→ Ένα γραφικό περιβάλλον GUI είναι μια ιεραρχία αντικειμένων: Ένα κουμπί μπορεί να περιέχεται σε ένα παράθυρο,που περιέχεται σε ένα tab, το οποίο περιέχεται σε ένα παράθυρο, κ.λπ. Έτσι, κάθε στοιχείο GUI έχει ένα γονέα(parent). Για αυτό και οι δύο κατασκευαστές έχουν μια παράμετρο γονέα .Η παρακολούθηση των γονέων είναι χρήσιμη όταν έχουμε να εμφανίσουμε / αποκρύψουμε μια ομάδα widgets, να τα βελτιώσουμε την οθόνη ή απλά να τα καταστρέψουμε, όταν τερματίζει η εφαρμογή.

→ Σε αυτό το σημείο,σημειώνεται ότι το wx.Frame αντικείμενο (που αρκετοί προγραμματιστές python χρησιμοποιού αντί του tkinter) έχει δύο παραμέτρους: 1) το id (ένα αναγνωριστικό του widget) και 2) τον τίτλο (ο τίτλος του παραθύρου).

→ Είναι μια καλή συνήθεια, κατά την κατασκευή κάθε είδους στοιχείου GUI,να κρατάμε μια αναφορά στο γονέα(parent).

2° βήμα – Αρχικοποίηση της GUI – Δημιουργία της main

my_application.py - C:/Python33/my_application.py

File Edit Format Run Options Windows Help

```
#!/usr/bin/python
#-*- coding: iso-8859-1 -*-

import tkinter

class myapp_tk(tkinter.Tk):
    def __init__(self,parent):
        tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        pass

if __name__ == "__main__":
    app = myapp_tk(None)
    app.title("Tonia's application")
```

→ Είναι συνήθως καλύτερο να έχουμε το τμήμα το κώδικα το οποίο δημιουργεί όλα τα GUI στοιχεία(κουμπί,πεδία κειμένου χωριστά από τη λογική του προγράμματος . Γι'αυτό έχουμε δημιουργήσει τη μέθοδο initialize(). Με αυτή τη μέθοδο θα δημιουργήσουμε όλα τα widgets.

→ Δημιουργούμε μια main που εκτελείται όταν το πρόγραμμα τρέχει από την γραμμή εντολών (command-line).

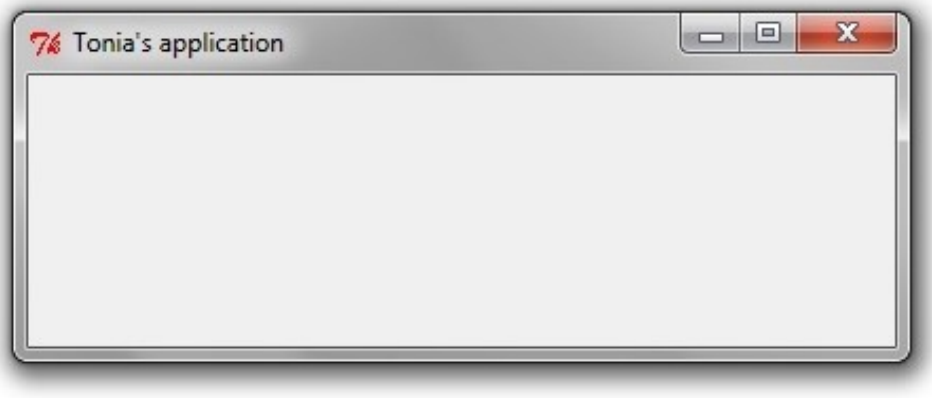
→ Στο tkinter δίνουμε υπόσταση στην τάξη μας (app=myapp_tk()). Δε δίνουμε γονέα(parent) (None) επειδή είναι το πρώτο GUI στοιχείο που χτίζουμε.

4ο βήμα – Loop

my_application.py - C:/Python33/my_application.py

File Edit Format Run Options Windows Help

```
#!/usr/bin/python
#-*- coding: iso-8859-1 -*-

import tkinter

class myapp_tk(tkinter.Tk):
    def __init__(self,parent):
        tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        pass

if __name__ == "__main__":
    app = myapp_tk(None)
    app.title("Tonia's application")
    app.mainloop()
```



→ Κάθε πρόγραμμα θα κάνει loop με την .mainloop ()

→ **Τι σημαίνει αυτό;** Σημαίνει ότι κάθε πρόγραμμα θα κάνει loop επ 'αόριστον, περιμένοντας για γεγονότα (να πατήσει ο χρήστης ένα κουμπί, το πάτημα των πλήκτρων, το λειτουργικό σύστημα να ζητήσει έξοδο από την εφαρμογή μας κλπ).

→ Τα Tkinter main loops θα λάβουν αυτά τα γεγονότα και θα τα χειριστούν αναλόγως.

→ Αυτή ονομάζεται “event-driven programming” κλήση(Επειδή το πρόγραμμα δεν θα κάνει τίποτα, αλλά να περιμένει για τα γεγονότα, και θα αντιδρά μόνο όταν λαμβάνει ένα γεγονός.)

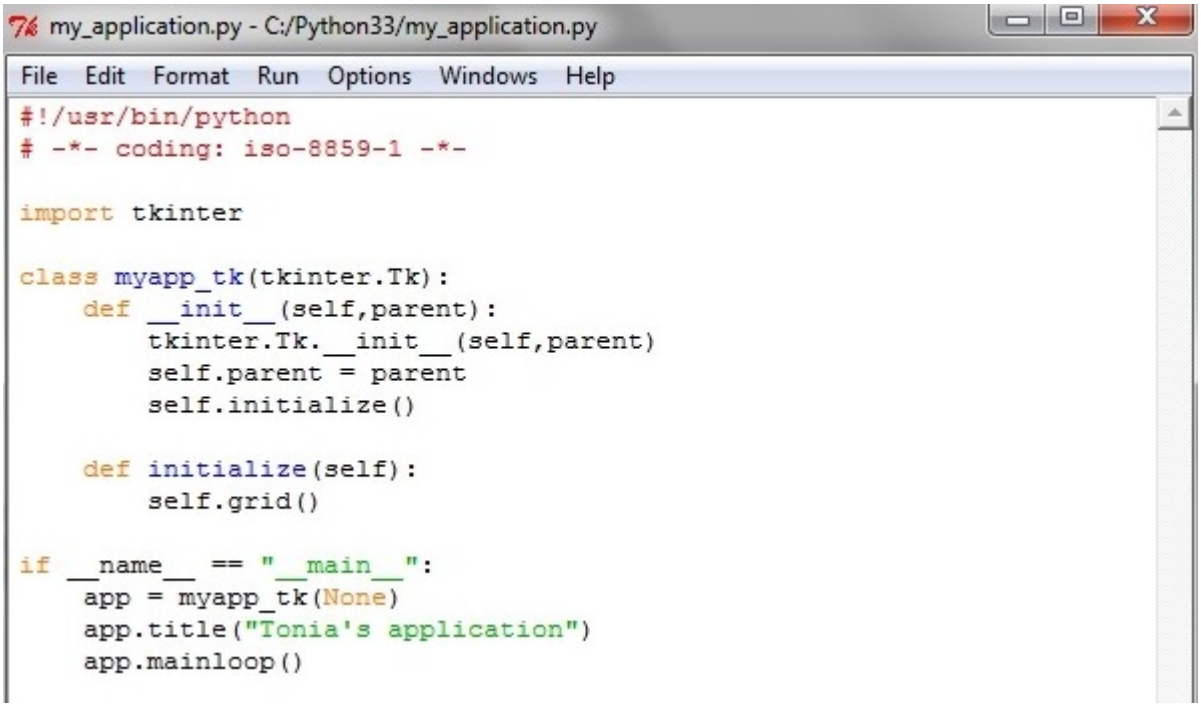
→ Με την εκτέλεση του προγράμματος εμφανίζεται ένα άδειο παράθυρο. Το κλείνουμε και πηγαίνουμε στον κώδικα για να προσθέσουμε widgets

5ο βήμα – Layout manager

→ Υπάρχουν πολλοί τρόποι για να βάλουμε widgets σε ένα παράθυρο: μπορούμε να τα προσθέσουμε από την πλευρά της σελίδας οριζόντια,κάθετα κτλπ.

→ Έτσι υπάρχουν διαφορετικές τάξεις (που ονομάζονται layout managers-διαχειριστές διάταξης) ικανές να τοποθετήσουν τα widgets στα δοχεία με διαφορετικούς τρόπους.Μερικές είναι πιο ευέλικτες από άλλες.

→ Στο Tkinter, καλώντας self.grid () θα δημιουργήσουμε απλά το διαχειριστή διάταξης του δικτύου , και θα πει στο παράθυρο μας να το χρησιμοποιήσει.



6ο βήμα – Προσθέτοντας την είσοδο κειμένου

Για να προσθέσουμε ένα widget ,πρέπει :

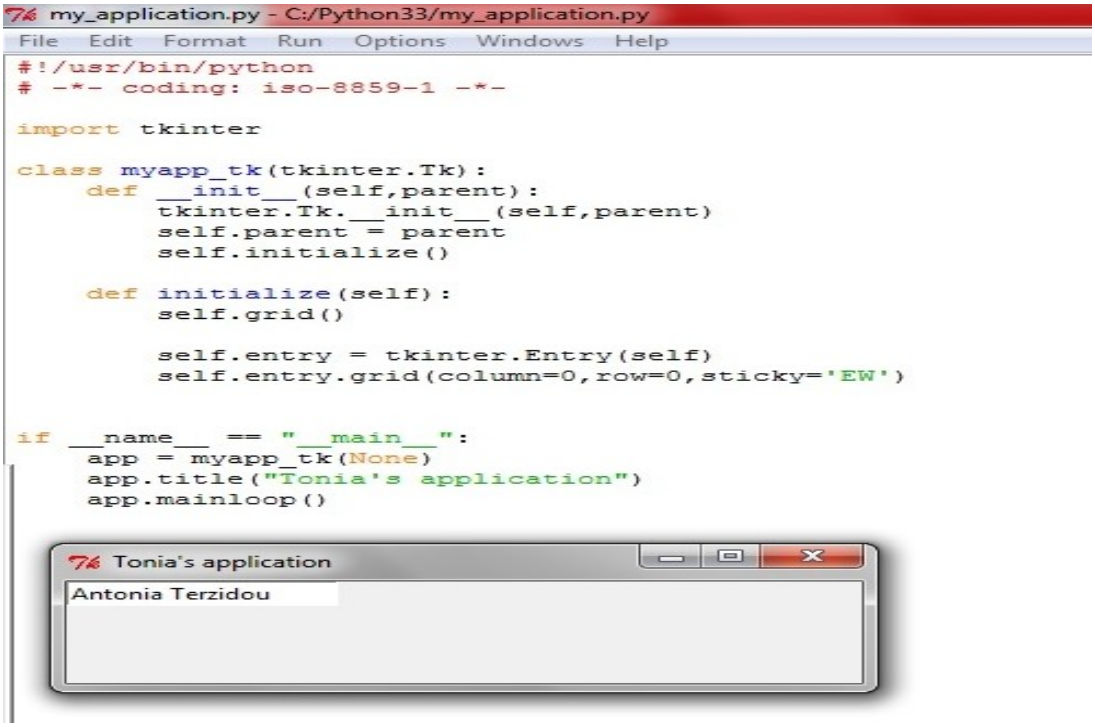
- ✓ Αρχικά δημιουργώ το widget

Στο tkinter, δημιουργούμε το widget εισόδου ως εξής → (self.entry = Tkinter.Entry ()). Σημείωση →Σημειώστε ότι κρατάμε μια αναφορά(reference) σε αυτό το widget στην τάξη μας: self.entry = tkinter.Entry(self). Αυτό συμβαίνει γιατί χρειάζεται να έχουμε πρόσβαση σε αυτό αργότερα, σε άλλες μεθόδους.

- ✓ Στη συνέχεια, το προσθέτω σε ένα layout manager.

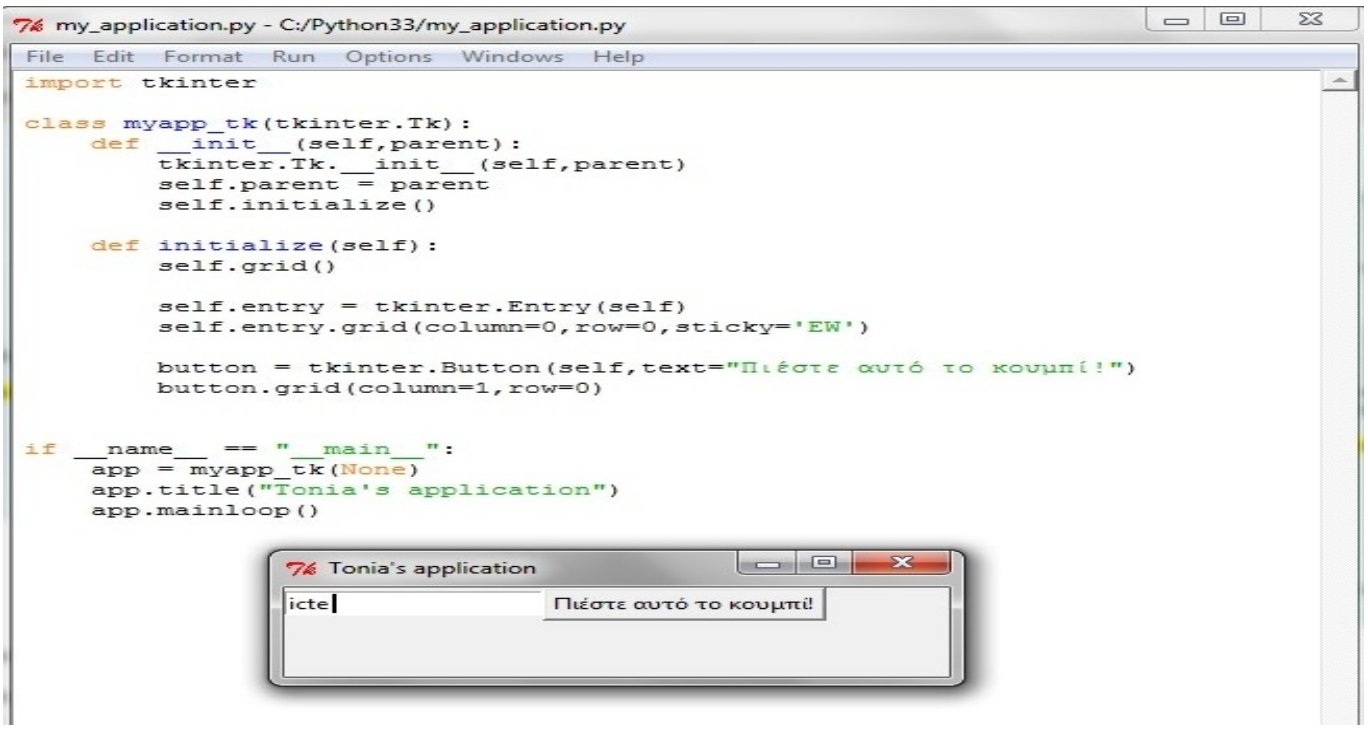
Στο tkinter, καλούμε τη μέθοδο .grid() για το widget. Υποδεικνύουμε που το βάζουμε στο grid (column=0, row=0). Όταν ένα κελί-στοιχείο γίνεται μεγαλύτερο από το widget που το περιέχει, μπορούμε να ζητήσουμε από το widget για να «κολλήσει» σε κάποια άκρα του κελιού. Εξού και το «EW».

(E = ανατολικά (αριστερά), W = West (δεξιά), N = Βόρειος (επάνω), S = South (κάτω)). Ορίζουμε «EW», που σημαίνει ότι το widget θα προσπαθήσει να κολλήσει στα δύο αριστερά και δεξιά άκρα του κελιού του.(Αυτός είναι ένας από τους στόχους μας: να έχουμε την επέκταση εισαγωγής κειμένου όταν το παράθυρο αλλάζει μέγεθος.)



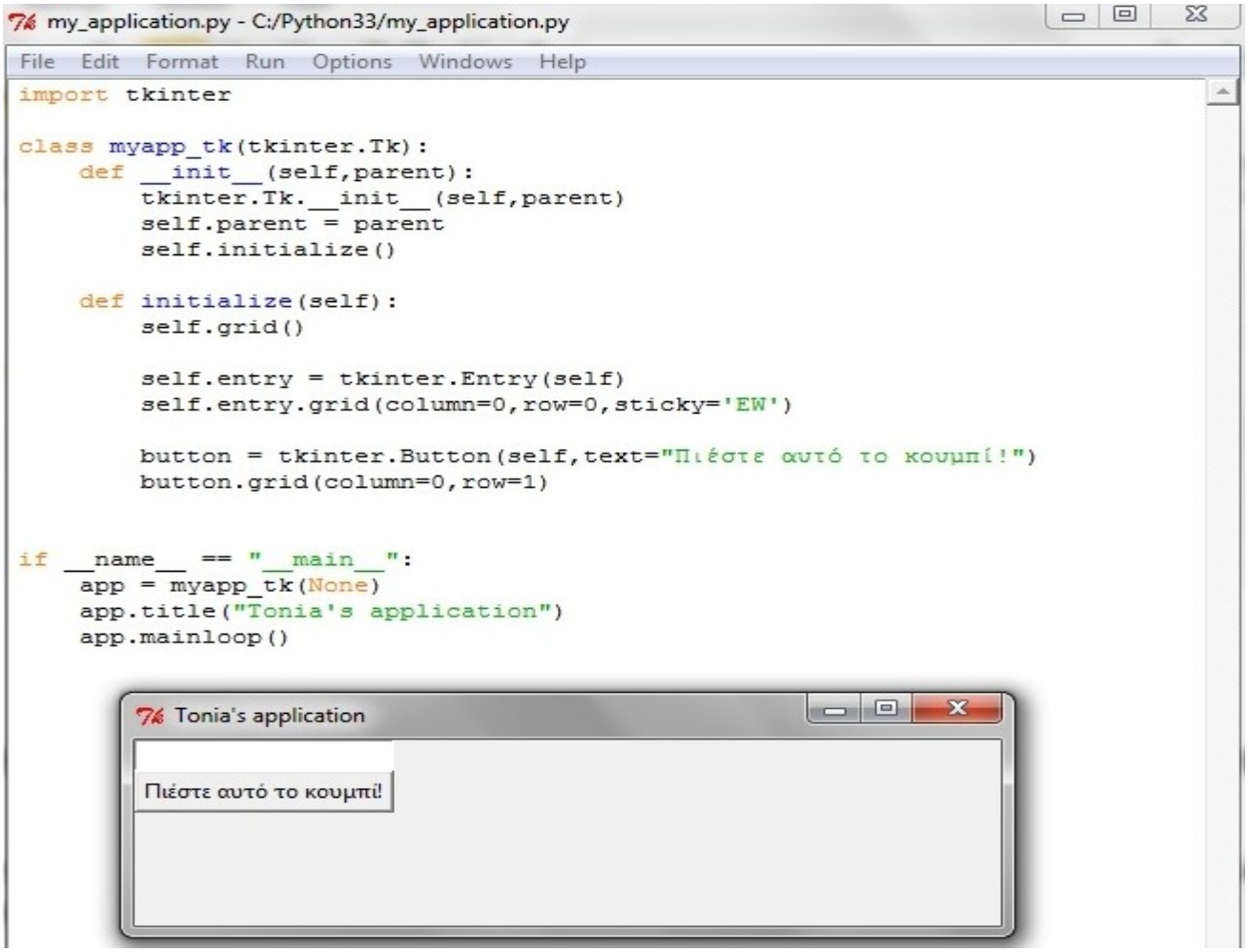
Έχουμε ένα παράθυρο με ένα πεδίο κειμένου στο οποίο μπορούμε να πληκτρολογήσουμε κάποιο κείμενο όπως φαίνεται παραπάνω.

7ο βήμα – Προσθέτοντας το κουμπί (button)



Σημείωση → Σε αυτή την περίπτωση, δεν τηρούνται references στο κουμπί (επειδή δεν θα διαβάσουμε ή θα τροποποιήσουμε την τιμή[value] του αργότερα.

→ Αλλάζοντας τις τιμές row και column μπορούμε να τροποποιήσουμε την γραφική αναπαράσταση της εφαρμογής,αλλάζοντας την διάταξη του πεδίου κειμένου αλλά και του button. Κάνουμε την τελική μας επιλογή μετά από δοκιμές και βλέποντας πως φαίνεται αισθητικά καλύτερη η διάταξη του παραθύρου.Για παράδειγμα δίνοντας column=0 και row=1 προκύπτει :



8ο βήμα – Προσθήκη ετικέτας-περίγραμμα(label)

Μπορούμε επίσης να δημιουργήσουμε και να προσθέσουμε μια ετικέτα. Αυτό είναι ένα Label Object στο tkinter.

- Για το χρώμα: Χρησιμοποιούμε μπλέ κείμενο σε κόκκινο φόντο. Πιο συγκεκριμένα → fg="blue",bg="red"
- Για την στοίχιση κειμένου: anchor="w" σημαίνει ότι το κείμενο θα πρέπει να έχει αριστερή στοίχιση στην ετικέτα

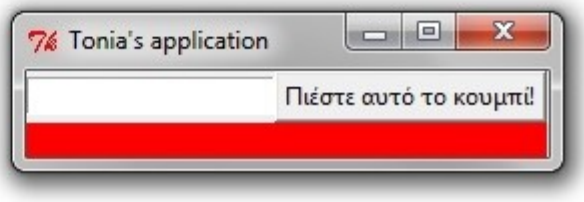
- Για την θέση της ετικέτας στο πλέγμα(grid): εδώ έχουμε και πάλι την .grid() μέθοδο, αλλά αυτή τη φορά εκτεινόμαστε σε δύο κελιά (έτσι ώστε να εμφανίζεται κάτω από το πεδίο κειμένου και το κουμπί.): Αυτή είναι η columnspan = 2 παράμετρος.
- Για την επέκταση ετικέτας (label expansion): Και πάλι, χρησιμοποιούμε sticky = "EW".

Μέχρι στιγμής έχουμε προσθέσει 3 widgets στην εφαρμογή μας!

my_application.py - C:/Python33/my_application.py

File Edit Format Run Options Windows Help

```
import tkinter

class myapp_tk(tkinter.Tk):
    def __init__(self,parent):
        tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')

        button = tkinter.Button(self,text="Πιέστε αυτό το κουμπί!")
        button.grid(column=1,row=0)

        label = tkinter.Label(self,
                              anchor="w",fg="blue",bg="red")
        label.grid(column=0,row=1,columnspan=2,sticky='EW')


if __name__ == "__main__":
    app = myapp_tk(None)
    app.title("Tonia's application")
    app.mainloop()
```



9ο βήμα - Ενεργοποίηση αλλαγής μεγέθους (Enable resizing)

→ Σε αυτό το σημείο λέμε στο διαχειριστή διάταξης(layout manager) να αλλάξει το μέγεθος των στηλών και των γραμμών του, όταν το παράθυρο αλλάζει μέγεθος. Όχι τις σειρές,μόνο την πρώτη στήλη column(0).

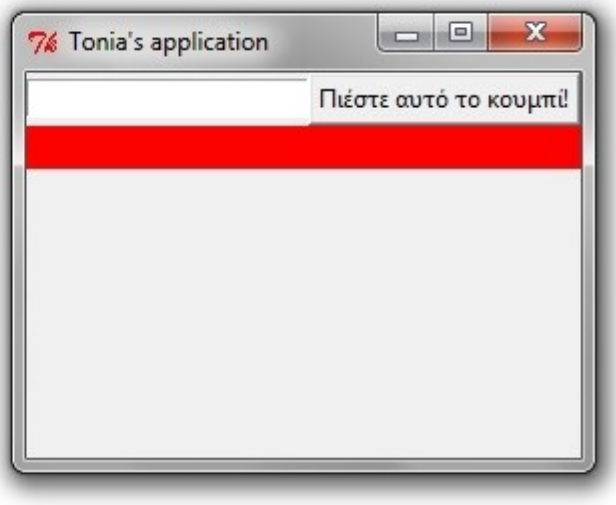
→ Βάζουμε στον κώδικα grid_columnconfigure ()

→ Σημείωση→ υπάρχουν πρόσθετες παράμετροι. Για παράδειγμα, κάποια στήλη μπορεί να αυξηθεί περισσότερο από άλλες, όταν αλλάξει μέγεθος. Για αυτό είναι η παράμετρος του βάρους (weight parameter). Σε αυτή την περίπτωση θέτουμε απλά weight =1.

my_application.py - C:\Python33\my_application.py

File Edit Format Run Options Windows Help

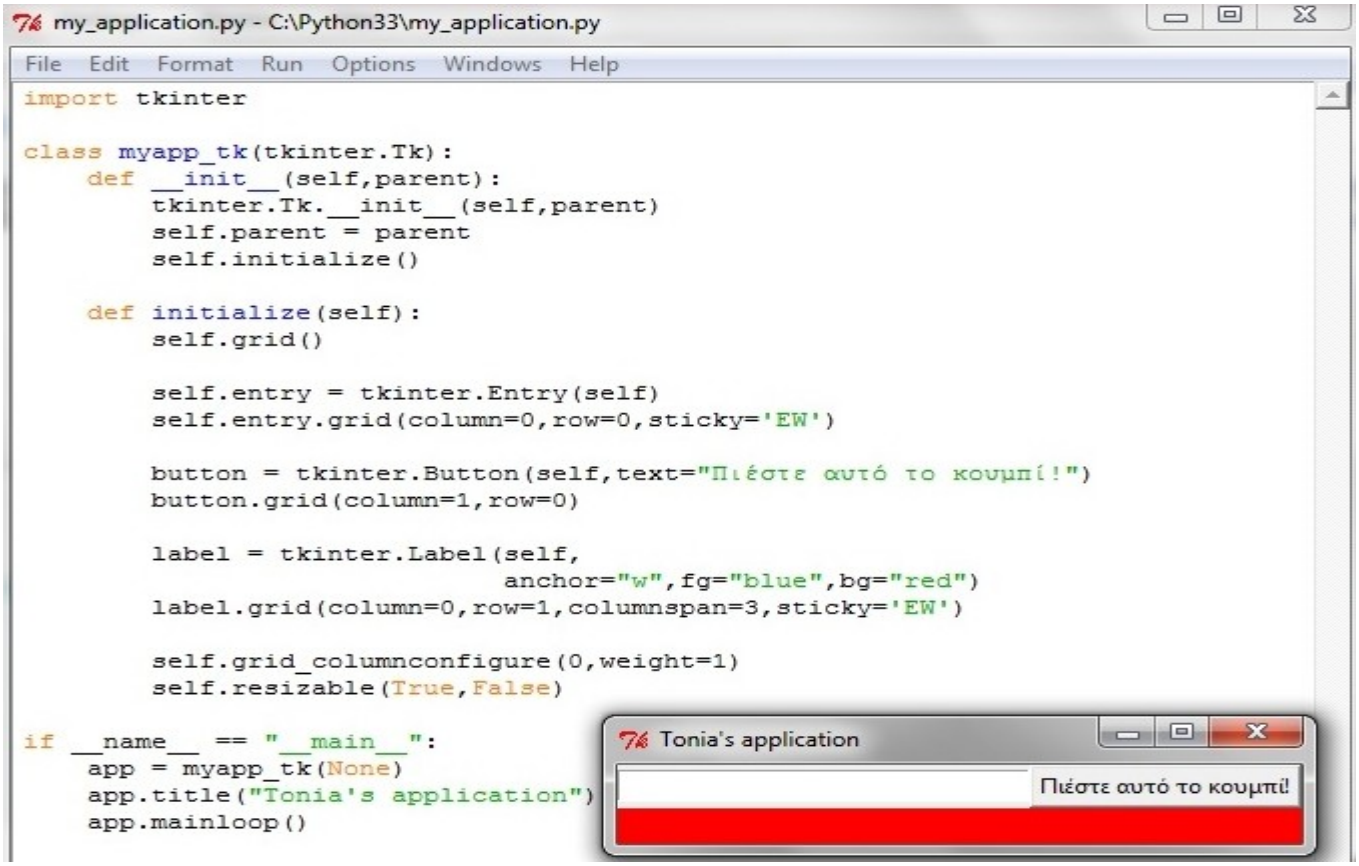
```
import tkinter

class myapp_tk(tkinter.Tk):
    def __init__(self,parent):
        tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')

        button = tkinter.Button(self,text="Πιέστε αυτό το κουμπί!")
        button.grid(column=1,row=0)

        label = tkinter.Label(self,
                              anchor="w",fg="blue",bg="red")
        label.grid(column=0,row=1,columnspan=3,sticky='EW')

        self.grid_columnconfigure(0,weight=1)

if __name__ == "__main__":
    app = myapp_tk(None)
    app.title("Tonia's application")
    app.mainloop()
```



→ Δοκιμάζουμε να αλλάξουμε μέγεθος στο παράθυρο.Παρατηρούμε ότι τα πεδία κειμένου και η κόκκινη ετικέτα αλλάζουν κατάλληλα μέγεθος και προσαρμόζονται στο μέγεθος του παραθύρου.

→ Για να αποτρέψουμε την κάθετη αλλαγή μεγέθους του παραθύρου,χρησιμοποιούμε .resizable(True,False).

→ Παρατηρούμε ότι με την εκτέλεση του προγράμματος δεν μπορούμε να επεκτείνουμε το μέγεθος του παραθύρου κάθετα.



10ο βήμα - Προσθήκη χειρισμού συμβάντων (event handlers)

Οι **χειρισμοί συμβάντων** (event handlers) είναι μέθοδοι οι οποίες θα κληθούν όταν κάτι συμβαίνει στο GUI. Έχουμε δεσμεύσει τους χειρισμούς συμβάντων σε συγκεκριμένα widgets για συγκεκριμένα γεγονότα μόνο. Άρα πρέπει να κάνουμε κάτι όταν πατηθεί το κουμπί ή όταν πατηθεί το ENTER στο πεδίο κειμένου.

- Δημιουργούμε μια OnButtonClick() μέθοδο η οποία θα καλείται όταν πατάμε το κουμπί. Επίσης δημιουργούμε μια OnPressEnter() μέθοδο, η οποία καλείται όταν πατάμε το ENTER στο πεδίο κειμένου.
- Στη συνέχεια, δεσμεύουμε αυτές τις μεθόδους στα widgets
- Για το κουμπί → απλά χρησιμοποιούμε **command=self.OnButtonClick**

Για το πεδίο κειμένου → Χρησιμοποιούμε μια .bind() μέθοδο. “<Return>” είναι το κλειδί που θέλουμε να δεσμεύσουμε. Η self.OnPressEnter είναι η μέθοδος που προκαλείται όταν πατάμε το πλήκτρο ENTER στο πεδίο κειμένου.

→ Δημιουργούμε μια ειδική tkinter μεταβλητή (self.labelVariable = Tkinter.StringVar())
→ Έπειτα το δεσμεύπυμε στο widget (textvariable=self.labelVariable)
→ Στη συνέχεια χρησιμοποιούμε set() ή get() για να ορίσουμε ή να διαβάσουμε τις τιμές του (self.labelVariable.set("Πατήσατε το κουμπί !"))

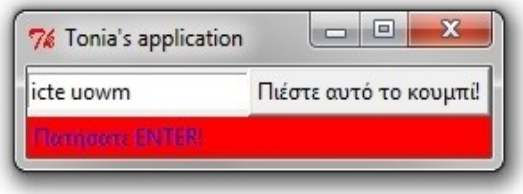
→ Κάθε φορά που θέλουμε να διαβάσουμε ή να θέσουμε τιμές σε ένα widget (text filed,label,checkbox,radio button κτλπ), πρέπει να δημιουργούμε μια tkinter μεταβλητή η οποία θα δεσμεύεται από το widget.

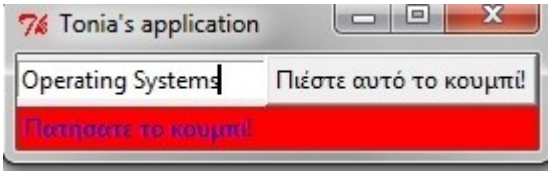
→ Υπάρχουν αρκετοί tkinter τύποι μεταβλητών (StringVar, IntVar, DoubleVar, BooleanVar).

my_application.py - C:\Python33\my_application.py

File Edit Format Run Options Windows Help

```
import tkinter

class myapp_tk(tkinter.Tk):
    def __init__(self,parent):
        tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')
        self.entry.bind("<Return>", self.OnPressEnter)

        button = tkinter.Button(self,text="Πιέστε αυτό το κουμπί!",command=self.OnButtonClick)
        button.grid(column=1,row=0)

        self.labelVariable = tkinter.StringVar()
        label = tkinter.Label(self,textvariable=self.labelVariable,
                              anchor="w",fg="blue",bg="red")
        label.grid(column=0,row=1,columnspan=3,sticky='EW')

        self.grid_columnconfigure(0,weight=1)
        self.resizable(True,False)

    def OnButtonClick(self):
        self.labelVariable.set("Πατήσατε το κουμπί!")

    def OnPressEnter(self,event):
        self.labelVariable.set("Πατήσατε ENTER!")

if __name__ == "__main__":
    app = myapp_tk(None)
    app.title("Tonia's application")
    app.mainloop()
```

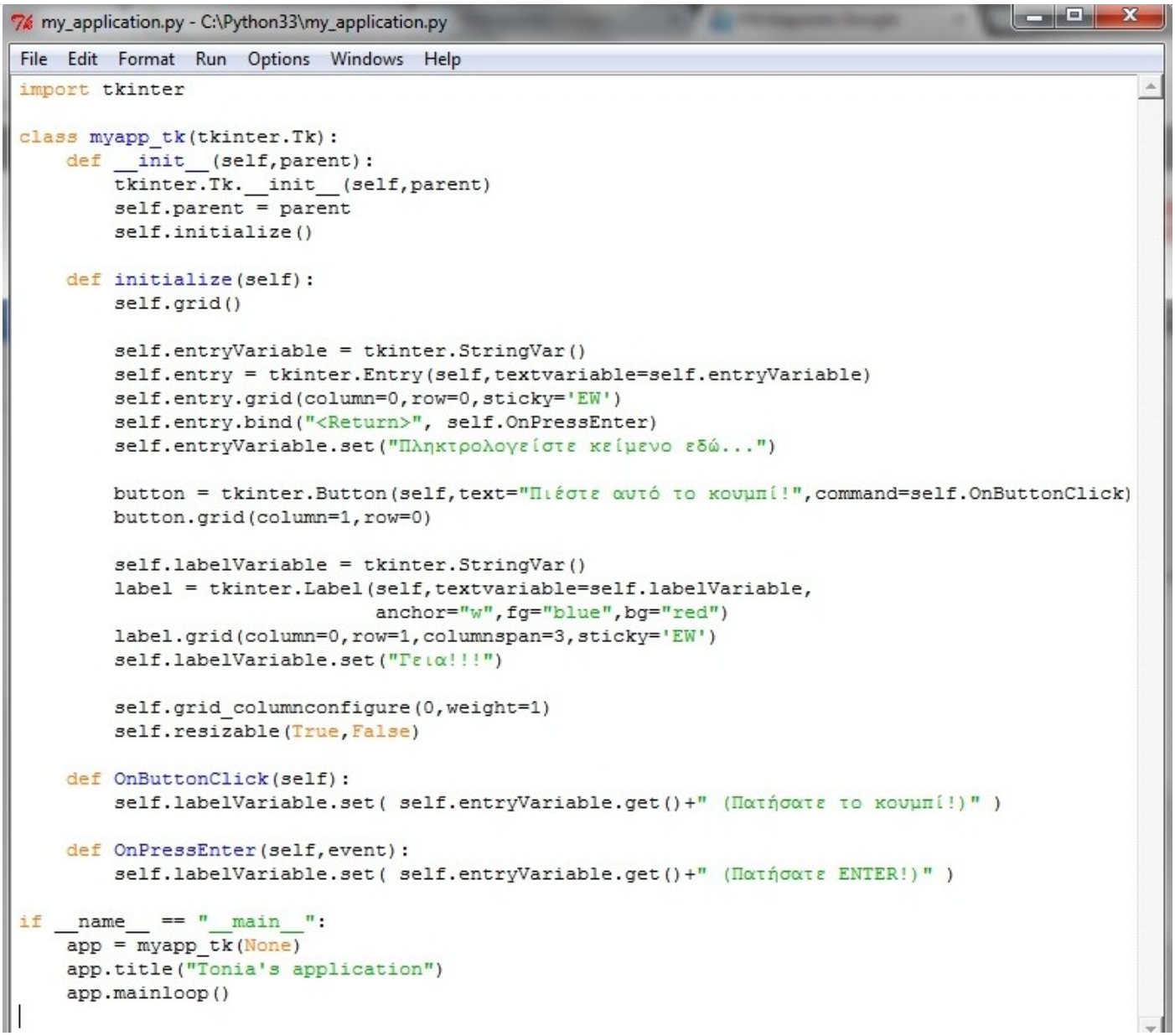




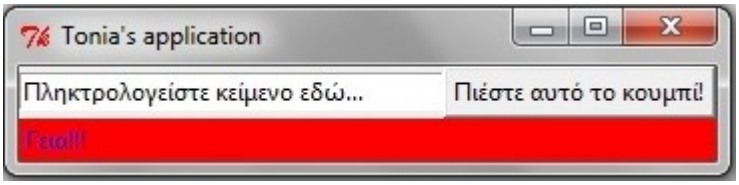
11° βήμα – Εμφάνιση τιμής

Σε αυτό το σημείο θα διαβάσουμε την τιμή του πεδίου κειμένου και θα την εμφανίσουμε στο κόκκινο περίγραμμα. Θα δημιουργήσουμε μια tkinter μεταβλητή την self.entryVariable.get().

Όπως έχουμε μια μεταβλητή κειμένου για να έχουμε πρόσβαση στο περιεχόμενο του πεδίο κειμένου, μπορούμε επίσης να ορίσουμε την προεπιλεγμένη τιμή (("Πληκτρολογείστε κείμενο εδώ...") και ("Γεια!!!")



Εκτελώντας το πρόγραμμα παρατηρώ ότι εισάγοντας κάποιο κείμενο στο πεδίο κειμένου και πατώντας ENTER ή κάνοντας κλικ στο κουμπί , το περίγραμμα(label) θα εμφανίσει το κείμενο που πληκτρολογήσαμε.



12ο βήμα – Αυτόματη επιλογή του πεδίου κειμένου

my_application.py - C:\Python33\my_application.py

File Edit Format Run Options Windows Help

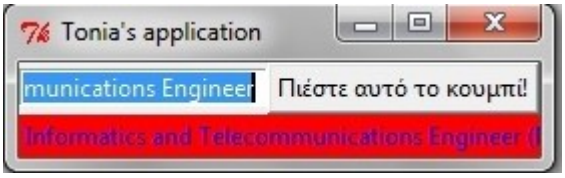
```
import tkinter

class myapp_tk(tkinter.Tk):
    def __init__(self,parent):
        tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entryVariable = tkinter.StringVar()
        self.entry = tkinter.Entry(self,textvariable=self.entryVariable)
        self.entry.grid(column=0,row=0,sticky='EW')
        self.entry.bind("<Return>", self.OnPressEnter)
        self.entryVariable.set("Πληκτρολογείστε κείμενο εδώ...")

        button = tkinter.Button(self,text="Πιέστε αυτό το κουμπί!",command=self.OnButtonClick)
        button.grid(column=1,row=0)

        self.labelVariable = tkinter.StringVar()
        label = tkinter.Label(self,textvariable=self.labelVariable,
                              anchor="w",fg="blue",bg="red")
        label.grid(column=0,row=1,columnspan=3,sticky='EW')
        self.labelVariable.set("Γεια!!!")

        self.grid_columnconfigure(0,weight=1)
        self.resizable(True,False)
        self.entry.focus_set()
        self.entry.selection_range(0, tkinter.END)

    def OnButtonClick(self):
        self.labelVariable.set( self.entryVariable.get()+" (Πατήσατε το κουμπί!)" )
        self.entry.focus_set()
        self.entry.selection_range(0, tkinter.END)

    def OnPressEnter(self,event):
        self.labelVariable.set( self.entryVariable.get()+" (Πατήσατε ENTER!)" )
        self.entry.focus_set()
        self.entry.selection_range(0, tkinter.END)
```

```
if __name__ == "__main__":
    app = myapp_tk(None)
    app.title("Tonia's application")
    app.mainloop()
```

- Το πεδίο κειμένο επιλέγεται αυτόματα όταν ο χρήστης κάνει κλικ στο κουμπί ή πατήσει ENTER, έτσι ώστε ο χρήστης να μπορεί να πληκτρολογήσει αμέσως ένα νέο κείμενο(αντικαθιστώντας το ήδη υπάρχον).
- Αρχικά, θέτουμε focus σε αυτό το στοιχείο (focus_set() ή SetFocus()), και έπειτα επιλέγουμε όλο το κείμενο (selection_range() ή SetSelection()).

Παρατήρηση → Σημειώνεται ότι όταν πληκτρολογήσουμε ένα μεγάλο ή πολύ μικρό κείμενο στο πεδίο κειμένου και πατήσω ENTER , το παράθυρο αυτόματα μεγαλώνει ή μικραίνει αντιστοιχα. Αυτό είναι αισθητικά ωραίο , αλλά δεν είναι επιθυμητή συμπερφορά το παράθυρο της εφαρμογής να μεγαλώνει και να μικραίνει όλη την ώρα, είναι κάτι το οποίο δεν θα αρέσει στο χρήστη. Για αυτό μπορούμε να καθορίσουμε το μέγεθος του παραθύρου θέτωντας ως μέγεθος του παραθύρου το μέγεθός του (self.geometry(self.geometry())). Με αυτό τον τρόπο το tkinter θα σταματήσει να προσπαθεί να προσαρμόζει το μέγεθος του παραθύρου όλη την ώρα.

```
self.grid_columnconfigure(0,weight=1)
self.resizable(True,False)
self.update()
self.geometry(self.geometry())
self.entry.focus_set()
self.entry.selection_range(0, tkinter.END)
```



Drawing in tkinter (χρώματα,σχήματα)

Πραγματοποιείται στο widget Canvas. Ο Canvas είναι μια υψηλού επιπέδου δυνατότητα στα γραφικά του tkinter. Μπορεί να χρησιμοποιηθεί για την δημιουργία γραφημάτων,σύνηθων widgets ή την δημιουργία παιχνιδιών.

Χρώματα (Colors)

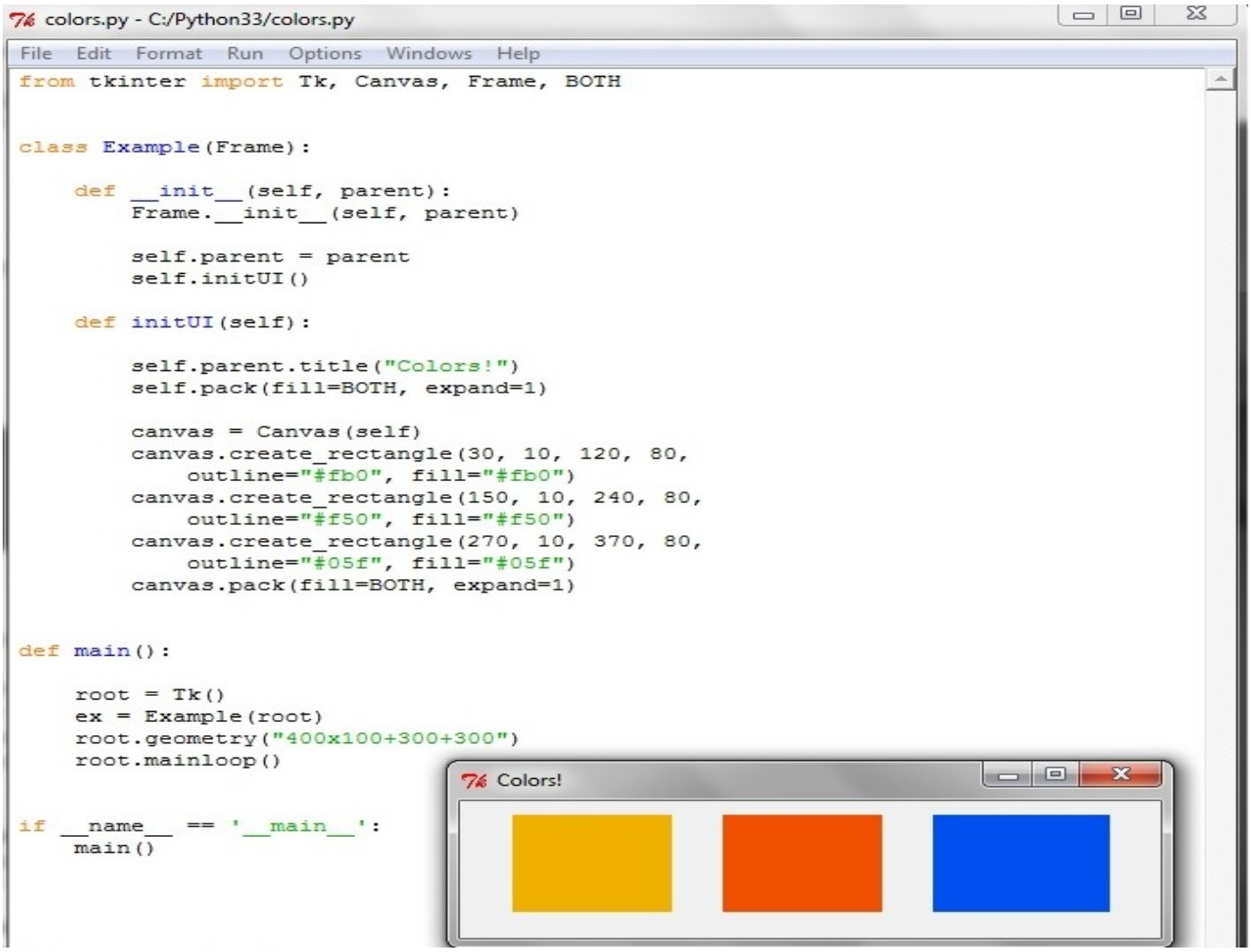
Ένα χρώμα είναι ένα αντικείμενο που αντιπροσωπεύει ένα συνδυασμό του κόκκινου, πράσινου και μπλε (RGB).

➔ Παραθέτω ένα παράδειγμα κώδικα, στο οποίο σχηματίζουμε 3 ορθογώνια,τα οποία γεμίζουμε με χρώματα.

- Αρχικά, δημιουργούμε το canvas widget → `canvas = Canvas(self)`
- Στη συνέχεια, δημιουργούμε ένα ορθογώνιο στον καμβά ως εξής : `create_rectangle()`

```
canvas.create_rectangle(30, 10, 120, 80,
    outline="#fb0", fill="#fb0")
```

Οι τέσσερις πρώτες παράμετροι είναι οι x,y συντεταγμένες των δύο σημείων οριοθέτησης(το top-left και button-right).Με την παράμετρο περιγράμματος ελέγχουμε το χρώμα του περιγράμματος του ορθογωνίου. Ομοίως η παράμετρος γεμίσματος παρέχει ένα χρώμα για το εσωτερικό του ορθογωνίου.



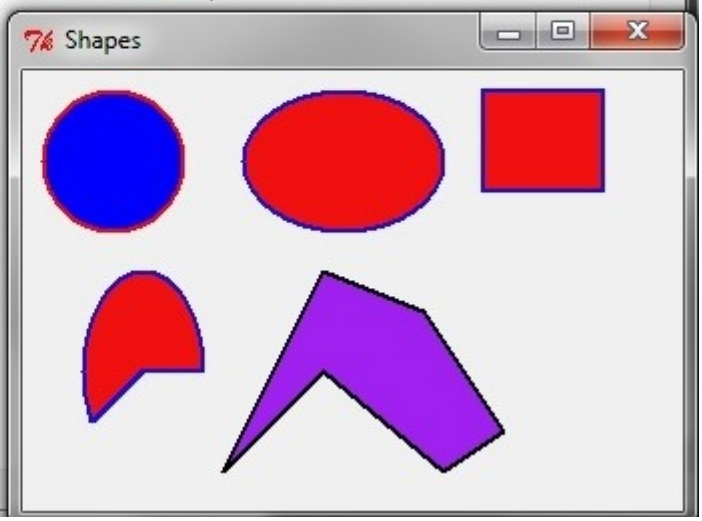
Σχήματα (shapes)

Μπορούμε να χρησιμοποιήσουμε διάφορα σχήματα πάνω στον καμβά. Στο παράδειγμα κώδικα που ακολουθεί θα δούμε μερικά από αυτά.

shape.py - C:/Python33/shape.py

File Edit Format Run Options Windows Help

```
from tkinter import Tk, Canvas, Frame, BOTH

class Example(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)

        self.parent = parent
        self.initUI()

    def initUI(self):

        self.parent.title("Shapes")
        self.pack(fill=BOTH, expand=1)

        canvas = Canvas(self)
        canvas.create_oval(10, 10, 80, 80, outline="red",
            fill="blue", width=2)
        canvas.create_oval(110, 10, 210, 80, outline="#11f",
            fill="#f11", width=2)
        canvas.create_rectangle(230, 10, 290, 60,
            outline="#11f", fill="#f11", width=2)
        canvas.create_arc(30, 200, 90, 100, start=0,
            extent=210, outline="#11f", fill="#f11", width=2)
        points = [150, 100, 200, 120, 240, 180, 210,
            200, 150, 150, 100, 200]
        canvas.create_polygon(points, outline='black',
            fill='purple', width=2)
        canvas.pack(fill=BOTH, expand=1)

def main():

    root = Tk()
    ex = Example(root)
    root.geometry("330x220+300+300")
    root.mainloop()


if __name__ == '__main__':
    main()
```



Παρατηρήσεις :

→ Έχω σχεδιάσει 5 διαφορετικά σχήματα στο παράθυρο:ένα κύκλο, μια έλλειψη, ένα ορθογώνιο,ένα τόξο και ένα πολύγωνο. Έχω δώσει διαφορετικά χρώματα στο γέμισμα και στο περίγραμμα (http://color-book.org/)

→ Το πλάτος του περιγράμματος είναι 2px.

→
```
canvas.create_oval(10, 10, 80, 80, outline="red",
                   fill="blue", width=2)
```
Για να σχεδιάσω ένα κύκλο, χρησιμοποιώ την μέθοδο create_oval(). Οι τέσσερις πρώτες παράμετροι είναι οι συντεταγμένες οριοθέτησης του κύκλου. Με άλλα λόγια, είναι οι x,y συντεταγμένες του top-left και bottom-right σημείων, στο οποίο έχει σχεδιασθεί ο κύκλος.

→
```
canvas.create_rectangle(230, 10, 290, 60,
          outline="#11f", fill="#f11", width=2)
```

Σχεδιάζω ένα ορθογώνιο. Οι συντεταγμένες είναι και πάλι το πλαίσιο οριοθέτησης του ορθογωνίου που θα σχεδιασθεί.

→
```
canvas.create_arc(30, 200, 90, 100, start=0,
        extent=210, outline="#11f", fill="#f11", width=2)
```
Αυτό το κομμάτι κώδικα δημιουργεί ένα τόξο. Ένα τόξο είναι μέρος της περιφέρειας του κύκλου. Η παράμετρος start είναι η γωνία έναρξης του τόξου. Η παράμετρος extent είναι το μέγεθος της γωνίας.

→
```
points = [150, 100, 200, 120, 240, 180, 210,
        200, 150, 150, 100, 200]
canvas.create_polygon(points, outline='black',
         fill='purple', width=2)
```

Σε αυτό το σημείο δημιουργείται ένα πολύγωνο. Είναι ένα σχήμα με πολλαπλές γωνίες. Για να δημιουργήσουμε ένα πολύγωνο στο tkinter, παρέχουμε μια λίστα από συντεταγμένες πολυγώνου στην create_polygon() μέθοδο.

Drawing text

→ Θέλουμε να σχεδιάσουμε κάποια μαθήματα της σχολής στο παράθυρο.

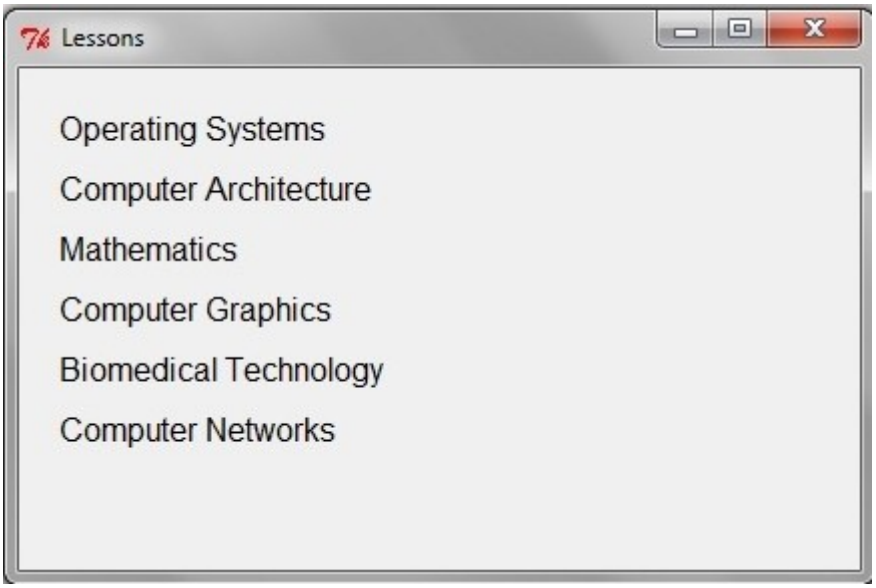
→
```
canvas.create_text(20, 30, anchor=W, font="Purisa",
           text="Operating Systems")
```

Οι δύο πρώτες παράμετροι είναι οι x, y συντεταγμένες του κεντρικού σημείου του κειμένου. Αν βάλουμε το στοιχείο κειμένου προς τα δυτικά, το κείμενο ξεκινά από τη θέση αυτή. Η παράμετρος font παρέχει τη γραμματοσειρά του κειμένου και η παράμετρος text είναι το κείμενο που θα εμφανίζεται.

drawing_text.py - C:/Python33/drawing_text.py

File Edit Format Run Options Windows Help

```
from tkinter import Tk, Canvas, Frame, BOTH, W


class Example(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent)

        self.parent = parent
        self.initUI()

    def initUI(self):

        self.parent.title("Lessons")
        self.pack(fill=BOTH, expand=1)

        canvas = Canvas(self)
        canvas.create_text(20, 30, anchor=W, font="Purisa",
            text="Operating Systems")
        canvas.create_text(20, 60, anchor=W, font="Purisa",
            text="Computer Architecture")
        canvas.create_text(20, 90, anchor=W, font="Purisa",
            text="Mathematics")
        canvas.create_text(20, 120, anchor=W, font="Purisa",
            text="Computer Graphics")
        canvas.create_text(20, 150, anchor=W, font="Purisa",
            text="Biomedical Technology")
        canvas.create_text(20, 180, anchor=W, font="Purisa",
            text="Computer Networks")
        canvas.pack(fill=BOTH, expand=1)


def main():

    root = Tk()
    ex = Example(root)
    root.geometry("420x250+300+300")
    root.mainloop()
```



Python - Multithreading

Το να τρέχεις διάφορα νήματα είναι παρόμοιο με την εκτέλεση πολλών διαφορετικών προγραμμάτων ταυτόχρονα, αλλά με τις ακόλουθες παροχές:

- ✓ Πολλαπλά threads(νήματα) μέσα σε μια διαδικασία μοιράζονται τον ίδιο χώρο δεδομένων με το κύριο νήμα και μπορούν να μοιραστούν συνεπώς πληροφορίες ή να επικοινωνήσουν μεταξύ τους πιο εύκολα από ό, τι αν ήταν ξεχωριστές διαδικασίες.
- ✓ Τα νήματα μερικές φορές ονομάζονται light-weight διαδικασίες και δεν απαιτούν πολύ μνήμη. Προτιμούνται από τις διεργασίες γιατί είναι «φθηνότερες».

Ένα νήμα (thread) έχει μια αρχή, μια ακολουθία εκτέλεσης κι ένα συμπέρασμα. Έχει ένα δείκτη εντολών (instruction pointer) που παρακολουθεί την πορεία του στα πλαίσια αυτού που βρίσκεται σε λειτουργία.

- ✓ Μπορεί να είναι pre-empted(διακόπτεται)
- ✓ Μπορεί προσωρινά να τεθεί σε αναμονή (επίσης γνωστή ως sleeping), ενώ άλλα νήματα τρέχουν - αυτό ονομάζεται yielding.

Ξεκινώντας ένα νήμα..

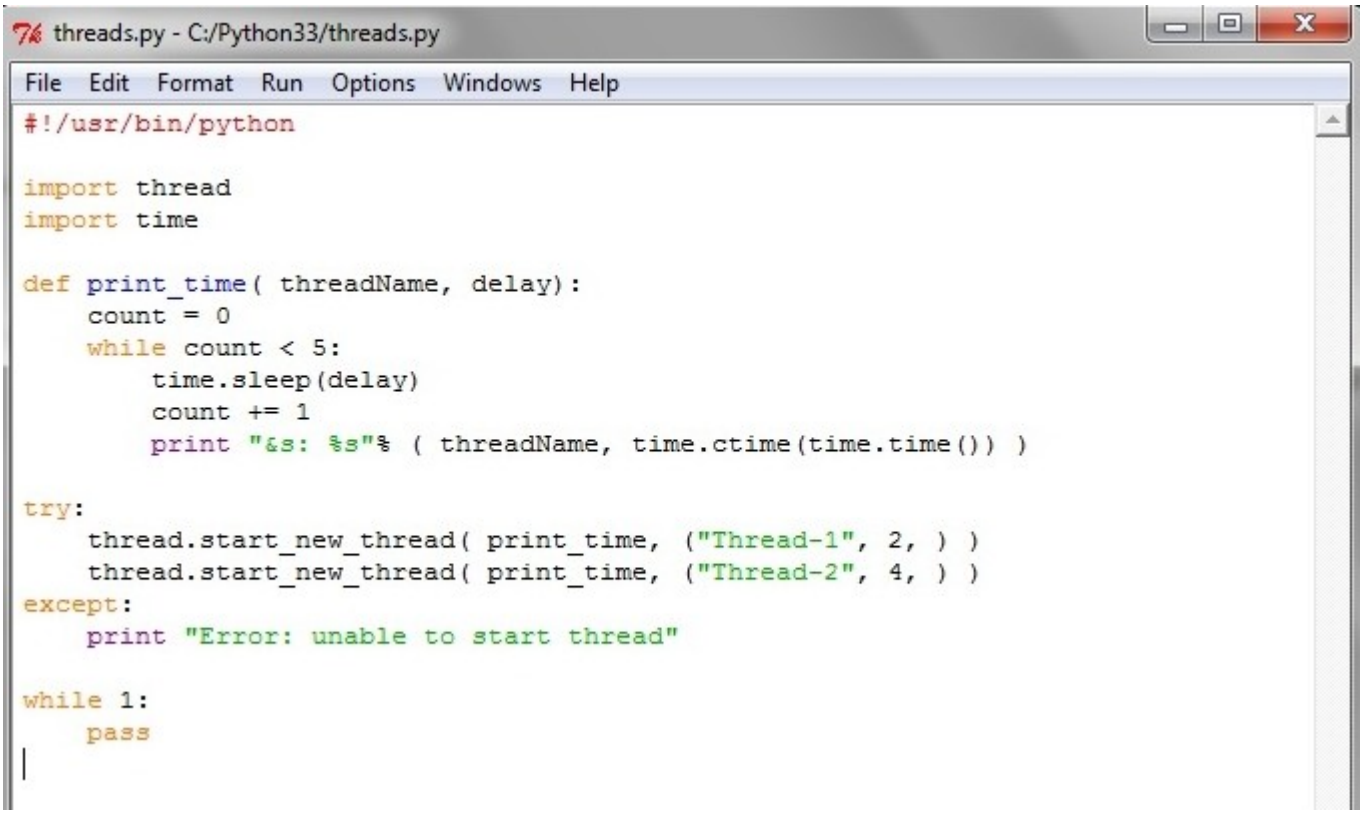
Για να δημιουργήσουμε ένα άλλο νήμα, θα πρέπει να κλέσουμε την ακόλουθη μέθοδο που είναι διαθέσιμη στο thread module :

```
thread.start_new_thread ( function, args[, kwargs] )
```

Η κλήση της μεθόδου επιτρέπει ένα γρήγορο και αποτελεσματικό τρόπο για να δημιουργήσουμε νέα νήματα τόσο σε Linux όσο και σε Windows. Η κλήση της μεθόδου επιστρέφει αμέσως και το παιδί νήμα (child thread) αρχίζει και καλεί τη συνάρτηση με την περασμένη λίστα των args. Όταν η συνάρτηση επιστρέφει, το νήμα τερματίζεται.

Εδώ τα args είναι μια πλειάδα από arguments. Χρησιμοποιούμε μια κενή πλειάδα για να καλέσουμε τη συνάρτηση χωρίς να υφίστανται κανένα argument. Τα kwargs είναι ένα προαιρετικό λεξικό των arguments λέξεων-κλειδιών.

→ Παρόλο που ο παρακάτω τρόπος είναι πολύ αποτελεσματικός για χαμηλού επιπέδου threading, αλλά το thread module είναι πολύ περιορισμένο σε σύγκριση με το νεότερο module threading θα δούμε αμέσως μετά.



Μέτρηση χρόνου εκτέλεσης

Αρκετά συχνά θέλουμε να μετρήσουμε τον χρόνο που παίρνει ένα συγκεκριμένο κομμάτι του προγράμματος μας έτσι ώστε να πάρουμε τις κατάλληλες υλοποιητικές αποφάσεις. Η χρονομέτρηση όμως της εκτέλεσης ενός προγράμματος είναι αρκετά πιο περίπλοκη διαδικασία από ό,τι ίσως να φαίνεται με την πρώτη ματιά.

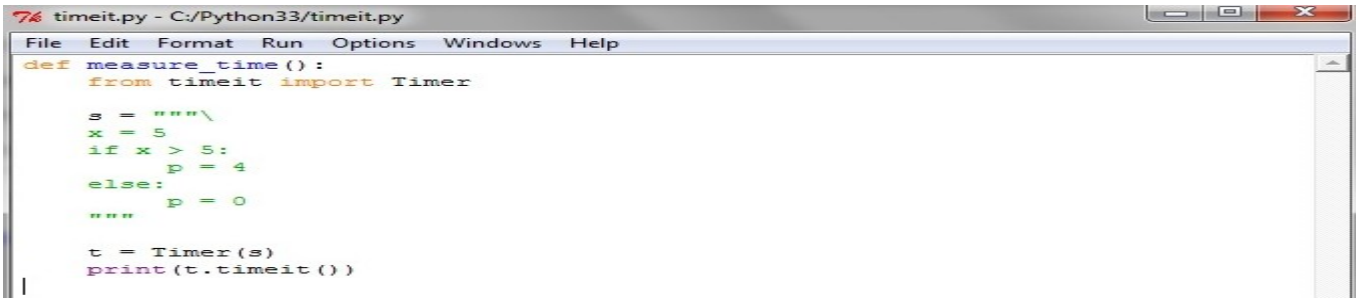
Το να γράψουμε μόνοι μας μια συνάρτηση που μετράει το χρόνο εκτέλεσης εμπεριέχει αρκετές δυσκολίες καθώς είναι μια εξαιρετικά περίπλοκη διαδικασία. Για παράδειγμα, στα σύγχρονα υπολογιστικά συστήματα, τρέχουν πολλές εφαρμογές παράλληλα, αξιοποιώντας μικρά κομμάτια επεξεργαστικού χρόνου.

Πόσος λοιπόν είναι ο πραγματικός χρόνος εκτέλεσης του προγράμματος μας ; Επίσης, ανάλογα και με τον τρόπο μέτρησης του χρόνου μπορεί να υπάρχουν διαφορές. Υπάρχει ενδεχόμενο να καλείται ο garbage collector την στιγμή που μετράμε τον χρόνο εκτέλεσης του προγράμματος, και αυτό να επηρεάζει το συνολικό χρόνο.Για αυτούς και για άλλους λόγους, καλό είναι να έχουμε ένα ενιαίο περιβάλλον με το οποίο παίρνουμε τις μετρήσεις τον προγραμμάτων μας.

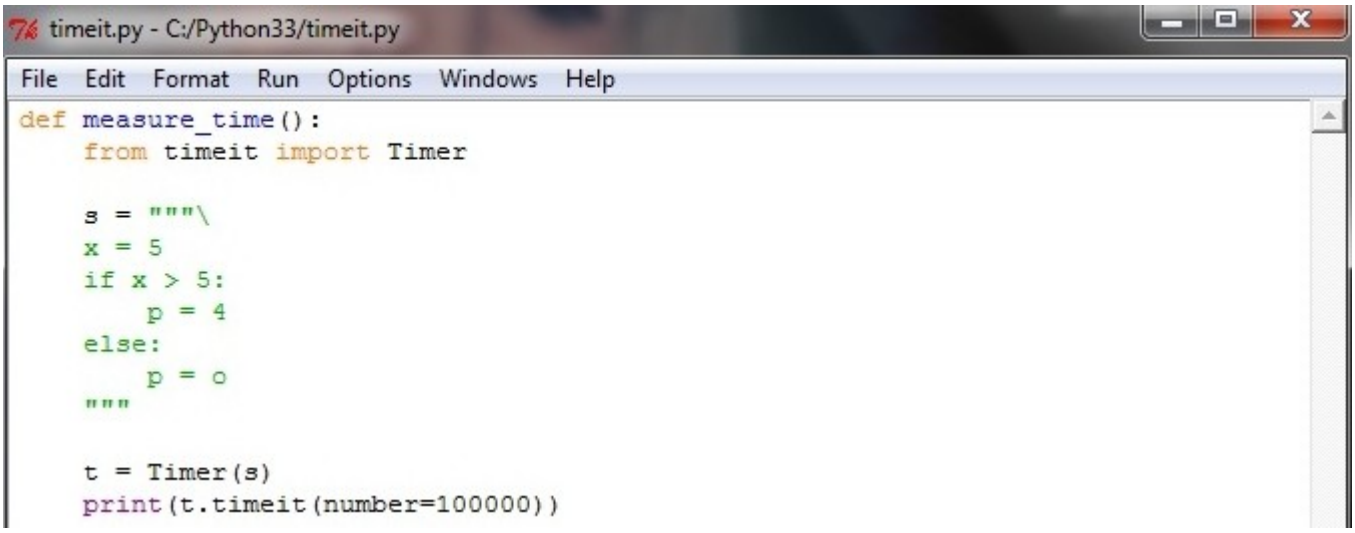
Η Python, σύμφωνα και με την φιλοσοφία της ότι έρχεται με 'τις μπαταρίες να περιέχονται' ήδη, μας προσφέρει ανάμεσα στο υπόλοιπο πλούσιο σύνολο βιβλιοθήκων, και το άρθρωμα timeit. Χρησιμοποιώντας το ***άρθρωμα timeit***, δημιουργείται ένα απομονωμένο περιβάλλον, μέσα στο οποίο μπορούμε να πάρουμε εύκολα τις μετρήσεις μας. Επίσης, το συγκεκριμένο άρθρωμα αποκρύπτει λεπτομέρειες που αφορούν την πλατφόρμα στην οποία εκτελείται ο κώδικας μας, παρέχοντας μας ένα ενιαίο περιβάλλον εργασίας ανεξαρτήτως λειτουργικού συστήματος. Μπορούμε να το χρησιμοποιήσουμε τόσο μέσα στα προγράμματα μας, όσο και από την διεπαφή γραμμής εντολών.

Μέτρηση Χρόνου Εκτέλεσης Δηλώσεων

Η πιο απλή είναι η περίπτωση όπου θέλουμε να μετρήσουμε μόνο κάποιες δηλώσεις και όχι ολόκληρη συνάρτηση. Εδώ δεν χρειάζεται να αρχικοποιήσουμε το περιβάλλον και μπορούμε απευθείας να χρονομετρήσουμε μετρήσουμε την απόδοση του προγράμματος μας.



Αξίζει να σημειωθεί ότι η συνάρτηση timeit() τρέχει τον κώδικα στον οποίο θα πάρει μετρήσεις 1.000.000 φορές, επομένως, για να έχουμε τα σωστά αποτελέσματα πρέπει τον χρόνο που μας βγάζει ως έξοδο σε δευτερόλεπτα να τον διαιρούμε με τον αριθμό των φορών που εκτελείται. Αν δεν χρειάζεται να εκτελέσουμε τόσες πολλές φορές τον κώδικα μας, μπορούμε να καθορίσουμε επακριβώς τον αριθμό των φορών εκτέλεσης θέτοντας το όρισμα number στην timeit().



Μέτρηση Χρόνου Εκτέλεσης Συνάρτησης

Για να μετρήσουμε τον χρόνο εκτέλεσης, καταρχάς πρέπει να εισάγουμε το άρθρωμα timeit. Ο πιο απλός τρόπος για να μετρήσουμε τον χρόνο εκτέλεσης μια συνάρτησης είναι να χρησιμοποιήσουμε ένα αντικείμενο Timer. Σε αυτό το αντικείμενο μας ενδιαφέρουν κυρίως δυο ορίσματα. Το πρώτο είναι οι δηλώσεις (statements) που θέλουμε να εκτελέσουμε ενώ το δεύτερο αναφέρεται στην αρχικοποίηση του περιβάλλοντος στο οποίο θα γίνει η μέτρηση.

Στο παρακάτω παράδειγμα :

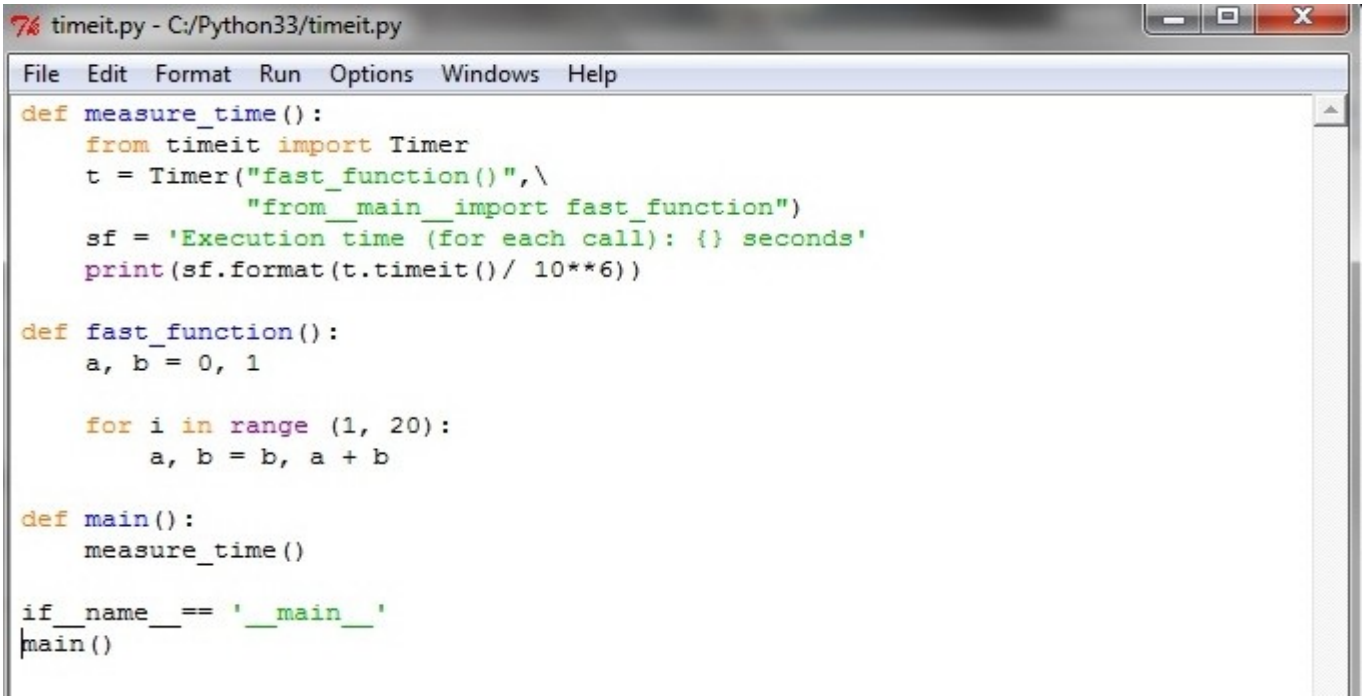
οι δύο πρώτοι fibonacci αριθμοί που είναι πάντα γνωστοί

a , b = 0, 1

#Βρίσκουμε τους 20 πρώτους fibonacci αριθμούς και τους τυπώνουμε

for i in range (1 , 20):

a , b = b,a + b



Για να μπορέσουμε να εκτελέσουμε την συνάρτηση fast_function(), πρέπει να την εισάγουμε από το περιβάλλον στο οποίο ανήκει. Για αυτό και το δεύτερο όρισμα για την αρχικοποίηση του περιβάλλοντος είναι απαραίτητο.

Προκειμένου να έχουμε αξιόπιστα αποτελέσματα, από προεπιλογή η συνάρτηση ‘‘fast_function’’ που ρίσκεται μέσα στον δημιουργό Timer θα εκτελεστεί 106 φορές, ώστε να πάρουμε πολλές μετρήσεις και να μην έχουμε λανθασμένα αποτελέσματα. Αν θέλουμε να καθορίσουμε κάτι διαφορετικό,θα πρέπει ρητά να το προσδιορίσουμε μετά, κατά τη κλήση t.timeit() όπου θα πρέπει ως όρισμα να εισάγουμε τον αριθμό των φορών που θέλουμε να επαναληφθεί η εκτέλεση της ‘‘fast_function’’. Για παράδειγμα, παρακάτω λέπουμε πως θα μπορούσαμε να εκτελέσουμε τον κώδικα μόνο 1000 φορές.

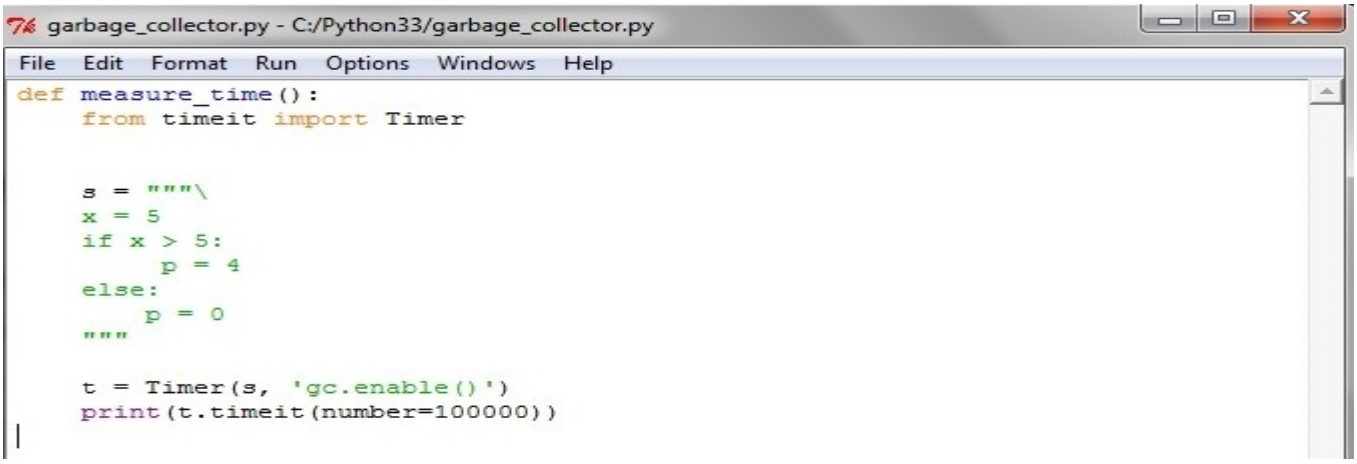
```
t  = Timer ( " fast_function ( ) " ,\

   "from __main__ import fast_function " )

 t . timeit (1000)
```

Για αυτό τον λόγο λοιπόν, και στο αρχικό παράδειγμα διαιρούμε τον χρόνο που μας επιστρέφεται με το 106, αφού τόσες είναι οι φορές που εκτελείται ο κώδικας.

Ενεργοποίηση garbage collector

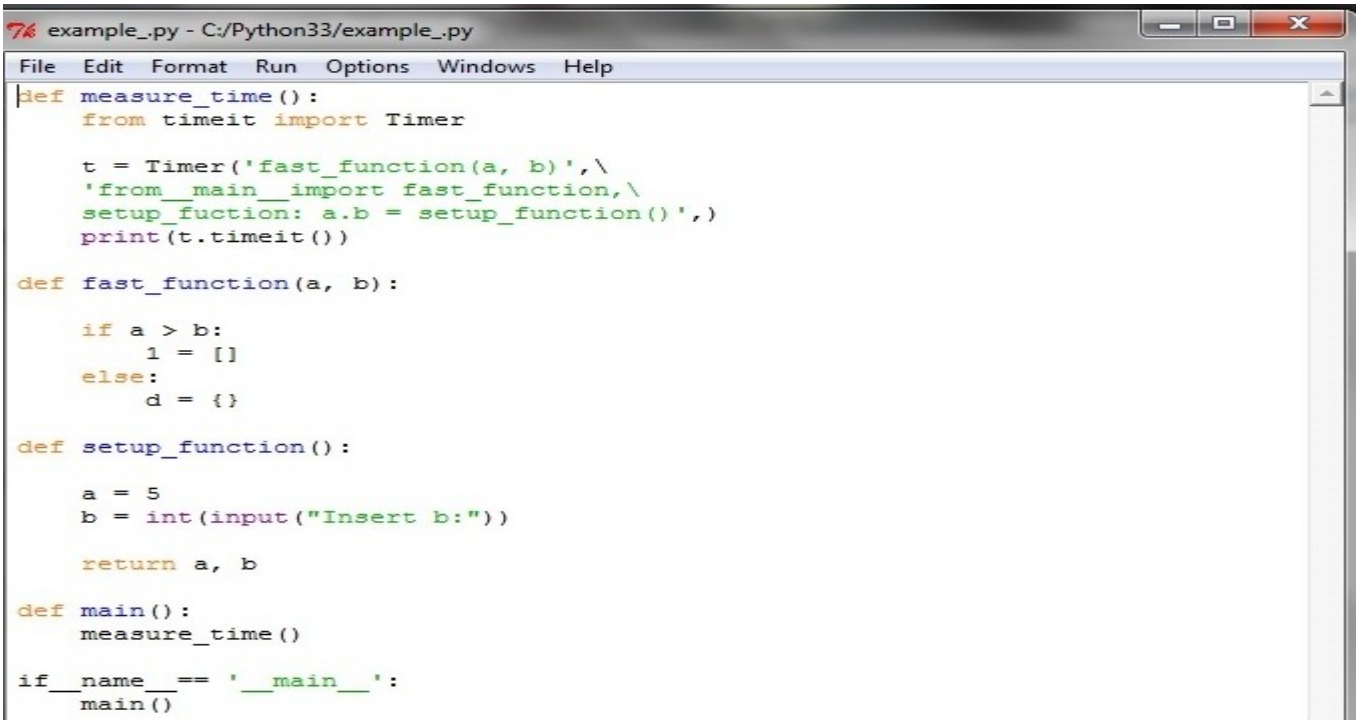
Όπως είπαμε και παραπάνω, για ακριβέστερες μετρήσεις, κατά την εκτέλεση της μεθόδου timeit() απενεργοποιείται ο garbage collector. Αν νομίζουμε πως αυτός είναι κρίσιμος για την ακριβέστερη μέτρηση του χρόνου της εφαρμογής μας

μπορούμε να τον ενεργοποιήσουμε ως ακολούθως, χρησιμοποιώντας στο όρισμα που αφορά την αρχικοποίηση του περιβάλλοντος εκτέλεσης το gc.enable().



Εκτέλεση Συνάρτησης με Ορίσματα

΄Ενα πιο δύσκολο παράδειγμα αφορά την χρονομέτρηση συνάρτησης που δέχεται ορίσματα.Για να καταλάβουμε πως γίνεται αυτό,ας σκεφθούμε πρώτα πως χρονομετράτε η συνάρτηση. Δημιουργείται ένα αντικείμενο Timer() το οποίο δημιουργεί ένα απομονωμένο περιβάλλον.Για να το κάνει αυτό καλεί τις συναρτήσεις που υπάρχουν στο όρισμα για την αρχικοποίηση αυτού του περιβάλλοντος. Επομένως για να εκτελέσουμε μια συνάρτηση που δέχεται ορίσματα, πρέπει στο περιβάλλον που δημιουργείται να υπάρχουν αυτά τα ορίσματα. Για αυτό τον λόγο καλούμε μια άλλη συνάρτηση πρώτα η οποία επιστρέφει τις τιμές των ορισμάτων που θα χρησιμοποιήσουμε στο περιβάλλον, και ύστερα τις περνάμε στην συνάρτηση της οποίας θέλουμε να μετρήσουμε την απόδοση.



΄Οπως βλέπουμε και στο παρακάτω παράδειγμα, μπορούμε κάλλιστα να έχουμε προκαθορισμένες τιμές των ορισμάτων, ή να ζητάμε από το χρήστη τις κατάλληλες τιμές.

Διεπαφή γραμμής εντολών

Αν θέλουμε να δοκιμάσουμε πόσο χρόνο απαιτείται για την εκτέλεση ενός συνόλου δηλώσεων, ένας πολύ γρήγορος τρόπος, αν αυτό είναι κάτι απλό, είναι να το κάνουμε μέσα από τη διεπαφή της γραμμής εντολών. Υπάρχει αντιστοιχία με αυτά που είδαμε παραπάνω, αλλά το κύριο που χρειάζεται να συγκρατήσει κανείς είναι η μορφή που θα έχει η δήλωση μας μέσω γραμμής εντολών και η οποία ακολουθεί το μοτίβο:

```
python -mtimeit -s"SETUP_COMMANDS" "COMMAND_TO_BE_TIMED"
```

΄Ενα απλό παράδειγμα όπου μετράμε πόσο χρόνο παίρνει η μετατροπή κάποιων αριθμών σε οκταδικό είναι το ακόλουθο, όπου μάλιστα έχουμε ενεργοποιήσει και τον σωρευτή απορριμάτων. (Garbage Collector)

```
python -mtimeit -s ' for i in range (10) : oct ( i ) ' ' gc.enable ( ) '
```

Εξαγωγή μετρικών (Profiling)

΄Οταν προσπαθούμε να βελτιώσουμε την ταχύτητα μιας εφαρμογής, σημαντικό όλο παίζει να βρούμε τα σημεία που την καθυστερούν ιδιαίτερα, καθώς μια βελτίωση σε αυτά θα επέφερε μεγαλύτερο κέρδος. Μάλιστα, στα οικονομικά υπάρχει ο όρος 'Νόμος των φθίνουσων αποδόσεων' (Law of diminishing returns). Ο νόμος αυτός περιγράφει πως η βελτίωση ενός συγκεκριμένου παράγοντα θα πάψει να αποφέρει κάποιο κέρδος από ένα σημείο και μετά, τουλάχιστον σε σχέση με τον κόπο που καταβάλλεται. Ο λόγος είναι ότι μπορεί να υπάρχουν κάποια άλλα σημεία που είναι πιο σημαντικά και θα πρέπει να επικεντρώσουμε τις προσπάθειες μας σε αυτά.

Στον προγραμματισμό, η διαδικασία προσδιορισμού αυτών των σημαντικών σημείων ονομάζεται profiling και η Python μας βοηθάει μέσω του cProfile. Μπορούμε να το τρέξουμε από την γραμμή εντολών δίνοντας του ως είσοδο το πρόγραμμα που θέλουμε να εξετάσουμε. Αυτό θα τυπώσει ως αποτέλεσμα πόσο χρόνο παίρνει η κάθε συνάρτηση που καλείται. Ο τρόπος που μπορούμε να χρησιμοποιήσουμε αυτή τη λειτουργία φαίνεται παρακάτω:

Python -m cProfile filename.py

Ως αποτέλεσμα, βλέπουμε το συνολικό αριθμό κλήσεων μιας συνάρτησης, συνολικό χρόνο εκτέλεσης, χρόνο εκτέλεσης ανά κλήση και άλλα.

The Python Profilers

Εισαγωγή στους Profilers

Το cProfile και το profile παρέχουν ντετερμινιστικό profiling των προγραμμάτων Python. Ένα προφίλ(profile) είναι ένα σύνολο στατιστικών που περιγράφει πόσο συχνά και για πόσο χρόνο εκτελούνται διάφορα μέρη του προγράμματος. Οι στατιστικές αυτές μπορούν να μορφοποιούνται σε αναφορές(reports) μέσω του pstats module.

Η πρότυπη βιβλιοθήκη της Python παρέχει τρεις διαφορετικές υλοποιήσεις της ίδιας διεπαφής προφίλ:

1. **cProfile**, συνιστάται για τους περισσότερους χρήστες.Είναι μια επέκταση της C με λογική επιβάρυνση που το καθιστά κατάλληλο για profiling long-running προγραμμάτων. Βασίζεται στο lsprof (Brett Rosen και Ted Czotter).
2. **profile**, ένα καθαρό Python module του οποίου διεπαφή είναι απομίμηση του cProfile, αλλά η οποία προσθέτει σημαντική επιβάρυνση στα profiled προγράμματα. Εάν προσπαθείτε να επεκτείνετε το profiler κατά κάποιο τρόπο, το θα μπορούσε να είναι πιο εύκολη με αυτό το module.Άλλαξε στην έκδοση 2.4→ Τώρα αναφέρει επίσης το χρόνο που δαπανάται σε κλήσεις προς ενσωματωμένες συναρτήσεις και μεθόδους.
3. **hotshot,** ήταν μια πειραματική μονάδα(module) της C, που επικεντρώθηκε στην ελαχιστοποίηση των overhead profiling, σε βάρος των μεγαλύτερου μήκους δεδομένων μετάεπεξεργασίας (post-processing) . Δεν διατηρείται πλέον και μπορεί να υπαρξει σε μελλοντική έκδοση της Python.Άλλαξε στην έκδοση 2.5→Τα αποτελέσματα θα πρέπει να είναι πιο ουσιαστικά από ό, τι στο παρελθόν: ο πυρήνας χρονισμού (timing core) περιείχε ένα κρίσιμο bug (critical bug).

Τα profile και cProfile modules εξάγουν το ίδιο interface, έτσι ώστε να είναι ως επί το πλείστον εναλλάξιμα. Το cProfile είναι νεότερο και ενδέχεται να μην είναι διαθέσιμο σε όλα τα συστήματα.Το cProfile είναι πραγματικά ένα στρώμα συμβατότητας στην κορυφή της εσωτερικής μονάδας _lsprof. Η hotshot μονάδα(module) προορίζεται για εξειδικευμένη χρήση.

Οι μονάδες profiler (profile modules) έχουν σχεδιαστεί για να παρέχουν ένα προφίλ εκτέλεσης για ένα συγκεκριμένο πρόγραμμα, όχι για σκοπούς συγκριτικής αξιολόγησης (γι 'αυτό, πάρχει timeit για αρκετά ακριβή αποτελέσματα). Αυτό ισχύει ιδιαίτερα για τη συγκριτική αξιολόγηση Python κώδικα και κώδικα C: οι profilers εισαγάγουν γενικά(overhead) για τον κώδικα Python, αλλά όχι για C-επιπέδου συναρτήσεις, και έτσι ο κώδικας C θα φαίνεται πιο γρήγορος από κάθε ένα κώδικα Python.

Για να κάνουμε profile σε μια συνάρτηση που παίρνει μονό argument, θα πρέπει να κάνουμε το εξής:

Χρησιμοποιούμε profile αντί cProfile, αν το τελευταίο δεν είναι διαθέσιμο στο σύστημά μας.

profile.py - C:/Python33/profile.py

File Edit Format Run Options Windows Help

```
import cProfile
import re
cProfile.run('re.compile("foo|bar")')
```

Python 3.3.2 Shell

File Edit Shell Debug Options Windows Help

```
         197 function calls (192 primitive calls) in 0.001 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.001    0.001 <string>:1(<module>)
        1    0.000    0.000    0.001    0.001 re.py:212(compile)
        1    0.000    0.000    0.001    0.001 re.py:268(_compile)
        1    0.000    0.000    0.000    0.000 sre_compile.py:179(_compile_charse
t)
        1    0.000    0.000    0.000    0.000 sre_compile.py:208(_optimize_chars
et)
        4    0.000    0.000    0.000    0.000 sre_compile.py:25(_identityfunctio
n)
      3/1    0.000    0.000    0.000    0.000 sre_compile.py:33(_compile)
        1    0.000    0.000    0.000    0.000 sre_compile.py:365(_compile_info)
        2    0.000    0.000    0.000    0.000 sre_compile.py:471(isstring)
        1    0.000    0.000    0.000    0.000 sre_compile.py:474(_code)
        1    0.000    0.000    0.001    0.001 sre_compile.py:489(compile)
        5    0.000    0.000    0.000    0.000 sre_parse.py:128(__len__)
       12    0.000    0.000    0.000    0.000 sre_parse.py:132(__getitem__)
        7    0.000    0.000    0.000    0.000 sre_parse.py:140(append)
      3/1    0.000    0.000    0.000    0.000 sre_parse.py:142(getwidth)
        1    0.000    0.000    0.000    0.000 sre_parse.py:180(__init__)
       10    0.000    0.000    0.000    0.000 sre_parse.py:185(__next)
        2    0.000    0.000    0.000    0.000 sre_parse.py:204(match)
        8    0.000    0.000    0.000    0.000 sre_parse.py:210(get)
        1    0.000    0.000    0.000    0.000 sre_parse.py:353(_parse_sub)
        2    0.000    0.000    0.000    0.000 sre_parse.py:431(_parse)
        1    0.000    0.000    0.000    0.000 sre_parse.py:69(__init__)
        1    0.000    0.000    0.000    0.000 sre_parse.py:726(fix_flags)
        1    0.000    0.000    0.000    0.000 sre_parse.py:738(parse)
        3    0.000    0.000    0.000    0.000 sre_parse.py:92(__init__)
        1    0.000    0.000    0.000    0.000 {built-in method compile}
        1    0.000    0.000    0.001    0.001 {built-in method exec}
       17    0.000    0.000    0.000    0.000 {built-in method isinstance}
    38/37    0.000    0.000    0.000    0.000 {built-in method len}
        2    0.000    0.000    0.000    0.000 {built-in method max}
        8    0.000    0.000    0.000    0.000 {built-in method min}
        6    0.000    0.000    0.000    0.000 {built-in method ord}
```

Ln: 47 Col: 4

Ανάλυση αποτελεσμάτων

- ✓ Η πρώτη γραμμή δείχνει ότι 197 κλήσεις ελέγχθηκαν. Από αυτές τις κλήσεις,οι 192 ήταν αρχέγονες(***primitive***) ,πράγμα που σημαίνει ότι η κλήση δεν προκαλείται μέσω αναδρομής.
- ✓ Η επόμενη γραμμή→***Ordered by:*** πρότυπο όνομα, δηλώνει ότι η συμβολοσειρά κειμένου στη δεξιά στήλη χρησιμοποιήθηκε για να ταξινομήσουμε την έξοδο.
- ✓ Οι επικεφαλίδες των στηλών περιλαμβάνουν:
  - ncalls → για τον αριθμό των κλήσεων
  - tottime → για το σύνολικό χρόνο που δαπανάται στην συγκεκριμένη λειτουργία (και με εξαίρεση το χρόνο που γίνονται κλήσεις προς υπο-λειτουργίες)
  - percall → είναι το πηλίκο tottime/ncalls
  - filename:lineno(function) → παρέχει τα σχετικά δεδομένα της κάθε λειτουργίας
- ✓ Όταν υπάρχουν δύο αριθμοί στην πρώτη στήλη (για παράδειγμα 3/1),σημαίνει ότι η λειτουργία είναι recursed(αναδρομική). Η δεύτερη τιμή είναι ο αριθμός των πρωτόγονων κλήσεων και η πρώτη είναι ο συνολικός αριθμός των κλήσεων. Σημειώνουμε ότι, όταν η λειτουργία δεν λειτουργεί αναδρομικά, οι δύο αυτές τιμές είναι οι ίδιες, και μόνο το ενιαίο σχήμα(single figure) εκτυπώνεται.

Αντί για την εκτύπωση της εξόδου, μπορούμε να αποθηκεύσουμε τα αποτελέσματα σε ένα αρχείο, καθορίζοντας ένα όνομα αρχείου στην run()function:

```
import cProfile

import re

cProfile.run('re.compile("foo|bar")', 'restats')
```

- ✓ Η τάξη pstats.Stats διαβάζει τα αποτελέσματα προφίλ από ένα αρχείο και τα διαμορφώνει με διάφορους τρόπους.
- ✓ Το αρχείο cProfile μπορεί επίσης να χρησιμοποιηθεί ως ένα σενάριο(script) στο προφίλ άλλου σεναρίου. Για παράδειγμα:

```
python -m cProfile [-o output_file] [-s sort_order] myscript.py
```

→ -o γράφει τα αποτελέσματα του προφίλ σε ένα αρχείο αντί για την κανονική έξοδο

→ -s καθορίζει μια από τις sort_stats() τιμές ταξινόμησης στην ταξινόμηση της εξόδου. Αυτό ισχύει μόνο όταν το -o δεν παρέχεται.

Το pstats module της Stats τάξης έχει μια ποικιλία μεθόδων για το χειρισμό και την εκτύπωση των δεδομένων που αποθηκεύονται σε ένα αρχείο αποτελεσμάτων προφίλ (profile results file):

```
import pstats

p = pstats.Stats('restats')

p.strip_dirs().sort_stats(-1).print_stats()
```

Η strip_dirs () μέθοδος αφαιρεί την εξωτερική διαδρομή από όλα τα ονόματα των modules. Η sort_stats () μέθοδος ταξινόμεί όλες τις καταχωρήσεις σύμφωνα με το πρότυπο μονάδα / σειρά / όνομα συμβολοσειράς που εκτυπώνεται. Η μέθοδος print_stats () εκτυπώνει όλα τα στατιστικά στοιχεία. Μπορείτε να δοκιμάσετε τις παρακάτω κλήσεις ταξινόμησης:

```
p.sort_stats('name')

p.print_stats()
```

Η πρώτη κλήση θα ταξινομήσει τη λίστα με βάση το όνομα της λειτουργίας-συνάρτησης και η δεύτερη κλήση θα εκτυπώσει τα στατιστικά στοιχεία. Παρακάτω θα δούμε μερικές ενδιαφέροντες κλήσεις με τις οποίες θα πειραματιστούμε:

```
p.sort_stats('cumulative').print_stats(10)
```

Αυτό ταξινομεί το προφίλ από το συνολικό χρόνο σε μια λειτουργία, και στη συνέχεια εκτυπώνει μόνο τις δέκα πιο σημαντικές γραμμές. Αν θέλετε να καταλάβετε τι αλγόριθμοι παίρνουν χρόνο, η παραπάνω γραμμή είναι αυτό που θα χρησιμοποιήσετε.

Αν ψάχνουμε να δούμε τι λειτουργίες κάνουν loop, και καταλαμβάνουν πολύ χρόνο, θα κάνουμε :

```
p.sort_stats('time').print_stats(10)
```

για να ταξινομήσετε ανάλογα με το χρόνο που δαπανήθηκε σε κάθε λειτουργία, και στη συνέχεια να εκτυπώσετε τα στατιστικά στοιχεία για τις δέκα κορυφαίες λειτουργίες.

Μπορείτε επίσης να δοκιμάσετε:

```
p.sort_stats('file').print_stats('__init__')
```

Αυτό θα ταξινομήσει όλα τα στατιστικά στοιχεία με βάση το όνομα αρχείου, και στη συνέχεια θα εκτυπώσει στατιστικά στοιχεία μόνο για την κατηγορία των μεθόδων init

(δεδομένου ότι γράφεται με __ init__ σε αυτά). Ως ένα τελευταίο παράδειγμα, θα μπορούσατε να δοκιμάσετε:

```
p.sort_stats('time', 'cum').print_stats(.5, 'init')
```

Αυτή η γραμμή ταξινομεί τα στατιστικά στοιχεία με ένα πρωτεύον κλειδί του χρόνου, και ένα δευτερεύον κλειδί του συνολικού χρόνου, και μετά εκτυπώνει μερικά από τα στατιστικά. Πιο συγκεκριμένα, η λίστα είναι η πρώτη που επιλέγεται μέχρι και το 50% (re: .5) του αρχικού μεγέθους του, έπειτα μόνο οι γραμμές που περιέχουν init διατηρούνται, και αυτή η υπο-υπο-λίστα θα εκτυπωθεί.

profile and cProfile Module Reference

Τόσο το profile όσο και το cProfile module παρέχουν τις ακόλουθες λειτουργίες:

→ ***profile.run(command, filename=None, sort=-1)***

Αυτή η συνάρτηση λαμβάνει ένα μόνο argument που μπορεί να περάσει στην exec() συνάρτηση,και ένα προαιρετικό όνομα αρχείου. Σε όλες τις περιπτώσεις, αυτή η ρουτίνα εκτελεί:

```
exec(command, __main__.__dict__, __main__.__dict__)
```

και συγκεντρώνει στατιστικά στοιχεία του profiling από την εκτέλεση.Εάν δεν υπάρχει το όνομα του αρχείου είναι, τότε αυτή η λειτουργία δημιουργεί αυτόματα ένα Stats στιγμιότυπο και εκτυπώνει μια απλή έκθεση-αναφορά του profiling. Εάν η τιμή ταξινόμησης καθορίζεται,αυτό περνάει σε αυτό το Stats instance για να γίνει έλεγχος πώς ταξινομούνται τα αποτελέσματα.

→ ***profile.runctx(command, globals, locals, filename=None)***

Αυτή η συνάρτηση είναι παρόμοια με την run(), με πρόσθετα arguments για την προμήθεια των καθολικών και τοπικών λεξικών για την εντολή string. Αυτή η ρουτίνα εκτελεί:

```
exec(command, globals, locals)
```

και συγκεντρώνει στατιστικές profiling όπως η run() συνάρτηση παραπάνω.

→ ***classprofile.Profile(timer=None,timeunit=0.0,subcalls=True, builtins=True)***

Η τάξη αυτή συνήθως χρησιμοποιείται μόνο αν απαιτείται ακριβέστερος έλεγχος σχετικά με το profiling, απ’ ότι η cProfile.run() συνάρτηση παρέχει.

Ένα σύνηθες χρονόμετρο μπορεί να παρέχεται για τη μέτρηση πόσου χρόνου χρειάζεται ο κώδικας για να τρέξει μέσω του χρονομέτρου argument. Αυτό πρέπει να είναι μια συνάρτηση που επιστρέφει έναν αριθμό που αντιπροσωπεύει την τρέχουσα ώρα.Εάν ο αριθμός είναι ένας ακέραιος, ο timeunit καθορίζει έναν πολλαπλασιαστή που καθορίζει την διάρκεια της κάθε μονάδας του χρόνου. Για παράδειγμα, εάν το χρονόμετρο επιστρέφει χρόνους που μετρούνται σε χιλιάδες δευτερόλεπτα, η μονάδα χρόνου θα είναι .001.

Άμεσα χρησιμοποιώντας την Profile τάξη σας επιτρέπει να επεξεργάζεστε τα αποτελέσματα προφίλ χωρίς να γράψετε τα δεδομένα προφίλ σε ένα αρχείο:

```
import cProfile, pstats, io

pr = cProfile.Profile()

pr.enable()

... do something ...

pr.disable()

s = io.StringIO()

ps = pstats.Stats(pr, stream=s)

ps.print_results()
```

- enable() : ξεκινάει τη συλλογή δεδομένων προφιλ (profiling data)
- disable() : σταματά τη συλλογή δεδομένων προφιλ
- create_stats() : σταματά τη συλλογή δεδομένων προφιλ και καταγράφει τα αποτελέσματα εσωτερικά ως το τρέχον προφίλ
- print_stats() : Δημιουργεί ένα αντικείμενο Stats με βάση το τρέχον προφιλ και εκτυπώνει τα αποτελέσματα στην έξοδο
- dump_stats() : γράφει τα αποτελέσματα του τρέχοντος προφιλ στο όνομα αρχείου
- run(cmd) : προφιλ του cmd μέσω exec()
- runctx(cmd,globals,locals) : προφίλ του cmd μέσω exec() με το καθορισμένο καθολικό και τοπικό περιβάλλον
- runcall(func, *args, **kwargs) : profile func((*args, **kwargs)

Αποσφαλμάτωση

Δεν είναι λίγες οι φορές που καλούμαστε να αντιμετωπίσουμε μια ιδιάζουσα περίπτωση στο πρόγραμμα μας, η οποία όμως ξεφεύγει από τον αυστηρό στόχο όσον αφορά το πρόβλημα που έχουμε να λύσουμε και έχει να κάνει κυρίως με την διαχειριστική επίλυση περιπτώσεων όπως η απουσία ενός αρχείου.

Είδη σφαλμάτων

Συντακτικά σφάλματα

Το συντακτικό αναφέρεται στη δομή ενός προγράμματος και στους κανόνες αυτής. ΄Ενα πρόγραμμα που δεν τηρεί το συντακτικό της γλώσσας δεν μπορεί να εκτελεστεί. Αν προσπαθήσουμε να εκτελέσουμε ένα πρόγραμμα που παραβιάζει αυτούς τους κανόνες, ο διερμηνές ή ο μεταφραστής θα παραπονεθούν για συντακτικά λάθη. Αυτό το είδος προβλημάτων είναι και τα πιο απλά σχετικά και με τις τρεις κατηγορίες, γιατί ο διερμηνευτής ή ο μεταγλωττιστής μπορεί να μας ενημερώσει σε ποια γραμμή ακριβώς εντοπίζεται το σφάλμα.

Σφάλματα χρόνου εκτέλεσης

Τα σφάλματα χρόνου εκτέλεσης (run time errors) εμφανίζονται μόνο στην εκτέλεση του προγράμματος μας γιατί έχει συμβεί κάτι εξαιρετικό (τελείωσε η μνήμη, δεν έγινε σωστός χειρισμός μιας εξαίρεσης κοκ).

Λογικά σφάλματα

Αυτά είναι τα σφάλματα που ενώ εκτελείται το πρόγραμμα μας, παράγονται διαφορετικά από τα επιθυμητά αποτελέσματα. Αποτελούν την δυσκολότερη κατηγορία σφαλμάτων. Αν ο υπολογιστής μας μπορούσε να τα διορθώσει, θα μπορούσε να προγραμματίσει και μόνος τον εαυτό του!

Python Debugger

΄Ενα από τα πιο χρήσιμα εργαλεία για την συγγραφή ενός προγράμματος μεσαίου ή μεγαλύτερου μεγέθους είναι ένας **debugger**. Η Python έχει έναν debugger ο οποίος είναι διαθέσιμος με την μορφή ενός module, του pdb. Συνήθως, οι μόνες στιγμές που χρησιμοποιούμε κάποιον debugger είναι όταν κάτι πηγαίνει στραβά και οι συνηθισμένες μας τακτικές (print, log messages) αδυνατούν να βρουν τη ρίζα του προβλήματος. Ευτυχώς για εμάς, η Python περιλαμβάνει στην βασική της βιβλιοθήκη έναν διαδραστικό αποσφαλματωτή (debugger).

Για να χρησιμοποιήσουμε τον debugger, πρέπει να εισάγουμε το κατάλληλο άρθρωμα :

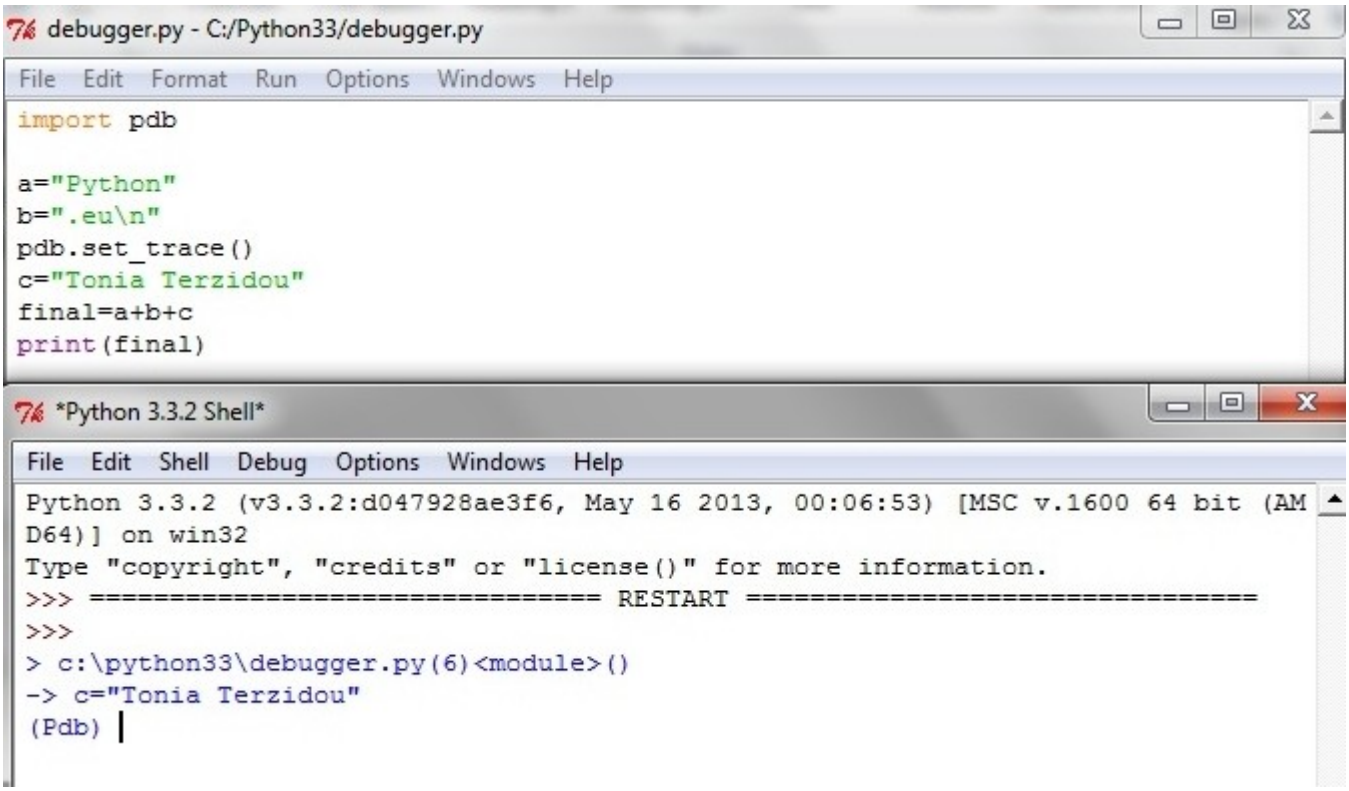
```
import pdb
```

Ο διαδραστικός debugger εκτελεί τον κώδικα με έναν ελεγχόμενο τρόπο. Επιτρέπει ανάμεσα σε άλλα να :

- ✓ Εξετάζουμε ένα κομμάτι κώδικα ανά γραμμή κάθε φορά.
- ✓ Καλούμε και να επιστρέφουμε από συναρτήσεις .
- ✓ Ορίζουμε σημεία διακοπής ροής του προγράμματος (breakpoints) .
- ✓ Χρησιμοποιούμε απευθείας την δύναμη του κελύφους της Python.

Βηματική εκτέλεση προγράμματος

Για την βηματική εκτέλεση του προγράμματος, χρησιμοποιούμε την συνάρτηση `pdb.set_trace()`:



Από το σημείο όπου χρησιμοποιείται η pdb.set_trace(), εκτελούμε την επόμενη δήλωση μέσω της εντολής ‘‘n’’ στο διαδραστικό κέλυφος που παρουσιάζεται μετά την εκτέλεση του προγράμματος. Μετά την πρώτη φορά που χρησιμοποιούμε το ‘‘n’’, μπορούμε να εκτελούμε κάθε φορά την επόμενη δήλωση πατώντας το πλήκτρο enter καθώς έτσι επαναλαμβάνεται η τελευταία εντολή αποσφαλμάτωσης.

Για επιστρέψουμε από το περιβάλλον του debugger χρησιμοποιούμε το ‘‘q’’ που τερματίζει τον debugger ή το ‘‘c’’ που επιτρέπει την κανονική συνέχιση της εκτέλεσης του προγράμματος.

Συναρτήσεις

΄Οποτε προχωράμε προς την επόμενη προς εκτέλεση εντολή με το ‘‘n’’, τότε ο debugger συμπεριφέρεται σε κάθε δήλωση με τον ίδιο τρόπο, ανεξάρτητα αν καλεί κάποια συνάρτηση ή όχι. Αν νομίζουμε πως το πρόβλημα βρίσκεται εντός κάποιας συνάρτησης τότε μπορούμε να προχωράμε με το ***‘‘s’’. Η συμπεριφορά του είναι ακριβώς η ίδια με το ‘‘n’’***, με την εξαίρεση ότι μόλις συναντά μια συνάρτηση που έχουμε γράψει εμείς, τότε συνεχίζεται η βηματική εκτέλεση για κάθε γραμμή του κώδικα αυτής της συνάρτησης.

Αν θέλουμε να επιστρέψουμε από μια συνάρτηση που κοιτάμε βηματικά μέσω της ‘‘s’’, τότε μπορούμε να χρησιμοποιήσουμε την ‘‘r’’. Με αυτή, συνεχίζεται η εκτέλεση του κώδικα χωρίς διακοπή μέχρι την δήλωση return της συγκεκριμένης συνάρτησης (ή μέχρι το τέλος της αν αυτή η δήλωση απουσιάζει).

Πολύ χρήσιμες είναι οι **εντολές ‘‘l’’ και ‘‘p’’**. Με την “l” μπορούμε να δούμε τις επόμενες γραμμές του κώδικα από το σημείο στο οποίο βρισκόμαστε, ενώ με την ‘‘p’’ ακολουθούμενη από το όνομα μιας μεταβλητής, μπορούμε να τυπώσουμε τα περιεχόμενα της.

Επιπρόσθετα σε αυτά, ***παρέχεται η δυνατότητα να γράψουμε απευθείας κώδικα σε Python μέσα από το περιβάλλον του αποσφαλματωτή***. ΄Ετσι, μπορούμε γρήγορα να θέσουμε τιμές σε μεταβλητές ώστε να δούμε αν όντως αυτές ευθύνονται για κάποιο σφάλμα ώστε να το διορθώσουμε στην συνέχεια, ή μπορούμε να δοκιμάσουμε απευθείας την λύση που νομίζουμε ότι θα λύσει το πρόβλημα χωρίς να χρειάζεται να ανατρέχουμε στο αρχείο του πηγαίου κώδικα για λίγες γραμμές μόνο και μόνο για να δοκιμάσουμε την λύση μας.

Αν τύχει και έχετε ονομάσει κάποια μεταβλητή σας με ένα όνομα που αντιστοιχεί σε κάποια εντολή, τότε μάλλον θα αντιμετωπίσετε προβλήματα με την συγγραφή κώδικα απευθείας σε Python. Για να το αποφύγετε λοιπόν, μπορείτε να αρχίσετε τις εντολές σας με ένα θαυμαστικό (!) υποδηλώνοντας έτσι στον debugger ότι η προς εκτέλεση εντολή είναι Python και όχι άμεση εντολή προς τον ίδιο.

Ο πίνακας που παρατίθεται παρακάτω παρουσιάζει συνοπτικά τις βασικές λειτουργίες που μπορορύμε να επιτελέσουμε στον debugger.

Εντολή	Λειτουργικότητα
Import pdb	Εισαγωγή αρθρώματος debugger
pdb.set_trace()	Εκκίνηση αποσφαλματωτή
<Enter>	Επανάληψη τελευταίας εντολής
Q	Έξοδος (quit)
P	Εκτύπωση τιμής μεταβλητής (print)
C	Συνέχιση κανονικής εκτέλεσης (continue)
L	Εκτύπωση προς εκτέλεση κώδικα(listing)
S	Βηματική εκτέλεση υπορουτίνας(step into)
R	Επιστροφή από υπορουτίνα (return)

GCI Programming

Η Common Interface Gateway, ή CGI, είναι ένα σύνολο κανόνων που καθορίζει τον τρόπο με τον οποίο ανταλλάσσονται πληροφορίες μεταξύ του web server και ένα σύνηθους σεναρίου.

Τα CGI specs σήμερα συντηρούνται από την NCSA και η NCSA ορίζει τη CGI έχει ως εξής:

Η Common Interface Gateway, ή CGI, είναι ένα πρότυπο για εξωτερικές πύλες προγραμμάτων για τη διασύνδεση με servers πληροφοριών, όπως οι HTTP servers.

Web Browsing

Για να κατανοήσουμε την έννοια της CGI, ας δούμε τι συμβαίνει όταν

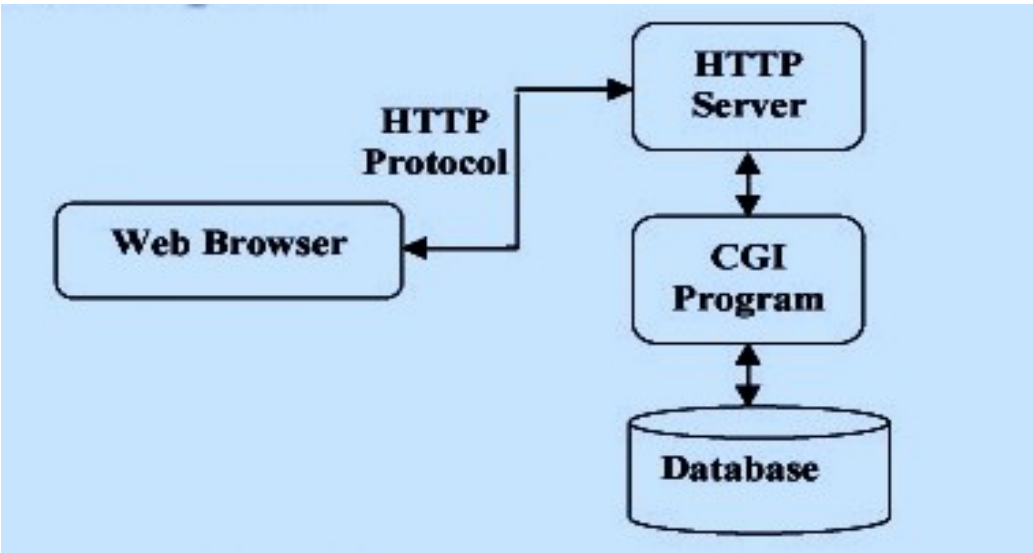
κάνουμε κλικ σε έναν υπερσύνδεσμο για να αναζητήσουμε μια

συγκεκριμένη ιστοσελίδα ή URL.

- Ο περιηγητής (browser)μας επικοινωνεί με τον HTTP web server και ζητά το URL ie.filename.
- Ο διακομιστής Web θα αναλύσει τη διεύθυνση URL και θα ψάξει για το όνομα του αρχείου, εάν εντοπίσει αυτό το αρχείο,το στέλνει πίσω στον browser διαφορετικά στέλνει ένα μήνυμα σφάλματος που υποδεικνύει ότι έχουμε ζήτήσει ένα λάθος αρχείο.
- Ο Web Browser παίρνει απάντηση από τον web server και εμφανίζει είτε το αρχείο που έχει ληφθεί είτε μήνυμα λάθους.

Ωστόσο, είναι δυνατό να εγκαθιδρύσετε τον HTTP server, έτσι ώστε κάθε φορά που ένα αρχείο σε ένα συγκεκριμένο κατάλογο κάνει αίτημα ότι το αρχείο δεν έχει αποσταλεί πίσω. Αντ'αυτού εκτελείται ως πρόγραμμα και όποια κι αν είναι η έξοδος του προγράμματος, αποστέλλεται πίσω στον browser μας για να εμφανιστεί . Αυτή η λειτουργία ονομάζεται Common Gateway Interface ή CGI και τα προγράμματα ονομάζονται CGI scripts. Αυτά τα CGI προγράμματα μπορεί να είναι ένα Python script, ένα Perl Script, Shell Script, C ή C + + πρόγραμμα κλπ.

Διάγραμμα CGI Αρχιτεκτονικής (CGI Architecture Diagram)



Υποστήριξη και Διαμόρφωση Web Server (Web Server Support & Configuration)

Πριν προχωρήσουμε με τον CGI προγραμματισμό, πρέπει να βεβαιωθούμε ότι ο Web Server μας υποστηρίζει CGI και έχει ρυθμιστεί για να χειρίζεται προγράμματα CGI. Όλα τα CGI προγράμματα για να εκτελεστούν από τον εξυπηρετητή HTTP, φυλάσσονται σε ένα προρυθμισμένο κατάλογο. Αυτός ο κατάλογος ονομάζεται CGI Directory και κατά συνθήκη, αυτό ονομάζεται ως / var / www / cgi-bin. Κατά σύμβαση τα αρχεία CGI θα έχουν ως επέκταση .cgi, αλλά μπορούμε να κρατήσουμε τα αρχεία μας με την επέκταση python (.py).

Από προεπιλογή, το Linux server έχει ρυθμιστεί να τρέχει μόνον τα σενάρια στον κατάλογο cgi-bin in / var / www. Εάν θέλουμε να καθορίσουμε οποιοδήποτε άλλο κατάλογο για να τρέξουν CGI scripts μας, θα πρέπει να σχολιάσουμε τις ακόλουθες γραμμές στο αρχείο httpd.conf:

```
<Directory "/var/www/cgi-bin">
   AllowOverride None
   Options ExecCGI
   Order allow,deny
   Allow from all
</Directory>

<Directory "/var/www/cgi-bin">
Options All
</Directory>
```

Παραδείγματα Python CGI

Ορισμένες εφαρμογές που τρέχουν σε ένα διακομιστή Web ανάκτηση και επεξεργασία δεδομένων από ένα χρήστη , τότε να δημιουργήσει και να μεταφέρετε δεδομένα πίσω στο πρόγραμμα περιήγησης στο Web του χρήστη . Αυτές οι εφαρμογές βασίζονται σε CGI , ή Common- Πύλη- Interface , πρωτόκολλα να κάνουμε πίσω από τα παρασκήνια έργο τους . CGI είναι ένα World Wide Web πρότυπο που καθορίζει το πώς server εφαρμογών μπορούν να διασυνδεθούν με το διακομιστή Web . Η Python είναι μια γενικής χρήσης , scripting γλώσσα που μπορεί να χρησιμοποιηθεί για τον προγραμματισμό εφαρμογών CGI.

Η γλώσσα προγραμματισμού Python περιλαμβάνει ένα " cgi " ενότητα της βιβλιοθήκης που καθιστά δυνατή για να αποσπάσουν πληροφορίες από HTML - HyperText Markup Language - έντυπα , να αναλύσει και να δημιουργήσουν νέο κώδικα HTML και το παρόν αυτός ο κώδικας για τον web server . CGI υποστήριξη της Python είναι τόσο πλήρης , μπορείτε να δημιουργήσετε το δικό σας απλό διακομιστή Web με μόλις επτά γραμμές κώδικα Python . Ο κώδικας που ακολουθεί από την ιστοσελίδα Python.org υλοποιεί ένα πολύ βασικό διακομιστή Web σε Python .

Η ενότητα " cgi " Python μας δίνει τη δυνατότητα να εξάγει τα δεδομένα που αποστέλλονται από μια φόρμα Web που αποστέλλονται χρησιμοποιώντας η "GET " ή τη μέθοδο " POST" . Αυτή η ενότητα περιέχει τη μέθοδο " cgi.FieldStorage ( ) , " η οποία εξάγει όλες τις πληροφορίες που αποστέλλονται από τη φόρμα. Μπορείτε να αναζητήσετε αυτά τα δεδομένα για πληροφορίες ή μπορείτε να έχετε πρόσβαση άμεσα την τιμή που αποδίδεται σε έναν συγκεκριμένο τομέα με βάση το όνομα . Για παράδειγμα, ο κώδικας Python μπορεί να πάρει τα δεδομένα από τα πεδία της φόρμας που ονομάζεται " firstname " και " επώνυμο " και να επιστρέψει ένα προσωπικό μήνυμα στον browser του χρήστη . Η " Python.org " ιστοσελίδα διαθέτει πολλές επιτυχημένες εφαρμογές που βασίζονται Python . Η " Journyx Timesheet " είναι ένα παράδειγμα ενός online εφαρμογή που ανακτά και επεξεργάζεται πολύπλοκα δεδομένα φόρμας Web .

Η Python κώδικα CGI μπορεί να να χρησιμοποιηθεί για την ανάκτηση στατικές ιστοσελίδες είναι αποθηκευμένα ως αρχεία σε ένα διακομιστή και να περάσει τους μαζί με το διακομιστή Web για την εκπομπή . Μπορείτε επίσης να δημιουργήσουν νέες , δυναμικές ιστοσελίδες . Για παράδειγμα , ένα σενάριο Python μπορεί να διαβάσει τα cookies του προγράμματος περιήγησης που αποθηκεύονται στον υπολογιστή σας και να δημιουργήσετε μια ιστοσελίδα που φαίνεται να έχουν δημιουργηθεί ειδικά για εσάς - με το όνομά σας , την τελευταία σελίδα που επισκεφτήκατε , μια λίστα με αντικείμενα που αγοράσατε τελευταία από την ιστοσελίδα ή τα στοιχεία που θα θέλατε . Python μπορεί να χρησιμοποιηθεί για να δημιουργήσει ολόκληρη ιστοσελίδες , blogs ή να διατηρήσουν εσωτερικά έγγραφα ενός οργανισμού . Python.org αναφέρει το Σύστημα Διαχείρισης Περιεχομένου EZRO ως παράδειγμα μιας Python -based εφαρμογή που χρησιμοποιείται για τη δημιουργία , τη διανομή και τη διατήρηση των εγγράφων στην τοπική ή δίκτυα ευρείας περιοχής .

Με την προσθήκη μιας ελεύθερης , ενότητα open-source που ονομάζεται MySQLdb ,μπορείτε να δημιουργήσετε Python scripts CGI που έχουν πρόσβαση και να χειριστούν τις βάσεις δεδομένων . Τα αντικείμενα και οι μέθοδοι στη μονάδα MySQLdb δώσει Python προγραμματιστές τα άγκιστρα που απαιτούνται για να συνδεθείτε , το ερώτημα , τη δημιουργία , ανάγνωση και εγγραφή δεδομένων σε βάσεις δεδομένων MySQL . Ως ένα εύκολο στη χρήση , γενική γλώσσα σκοπός , Python επεκτείνει τη δύναμη του λογισμικού βάσεων δεδομένων . Η ιστοσελίδα Python.org παραθέτει το Web της εταιρείας Gusto ως παράδειγμα ένα online εταιρεία που χρησιμοποιεί με επιτυχία την ικανότητα επεξεργασίας δεδομένων της Python . Gusto επικαλείται Python με τη δική μου για πληροφορίες στις βάσεις δεδομένων της που μπορούν να βοηθήσουν την εταιρεία την καλύτερη εξυπηρέτηση των μελών του ταξιδιού - κοινωνική υπηρεσία του δικτύου της .

Πώς να μετατρέψω UTF8 Python σε Unicode

Η Python είναι μια ερμηνευτική , γλώσσα προγραμματισμού ανοικτού κώδικα που χρησιμοποιεί μια διαισθητική object-oriented πλαίσιο . Κατά τον προγραμματισμό σε Python , μπορεί να χρειαστεί να αλλάξετε το κείμενο από το UTF - 8 , μια μορφή κωδικοποίησης 8 -bit , σε Unicode , μια μορφή κωδικοποίησης 16 - bit . Για παράδειγμα, πολλοί χρειάζονται να μετατρέψουν UTF - 8 κείμενο, το οποίο υποστηρίζει μόνο αγγλικούς χαρακτήρες , σε Unicode που μπορεί να χειριστεί γαλλική τόνους ή της Ρωσίας με την κυριλλική γραφή.

Ανοίγω το πρόγραμμα επεξεργασίας Python.

Δημιουργώ μια μορφή UTF - 8 Unicode , πληκτρολογώντας τα εξής : .

Unicode (« stringtext » , ' utf - 8')

Πατάω "Enter ".Η Python επιστρέφει τα εξής :

u'stringtext «

Σε αυτό το παράδειγμα , η Python μετατρέπει το UTF - 8 συμβολοσειράς κειμένου σε Unicode . Σε αυτό το σημείο , μπορείτε να προσθέσετε τόνους και άλλους χαρακτήρες στη συμβολοσειρά Unicode .

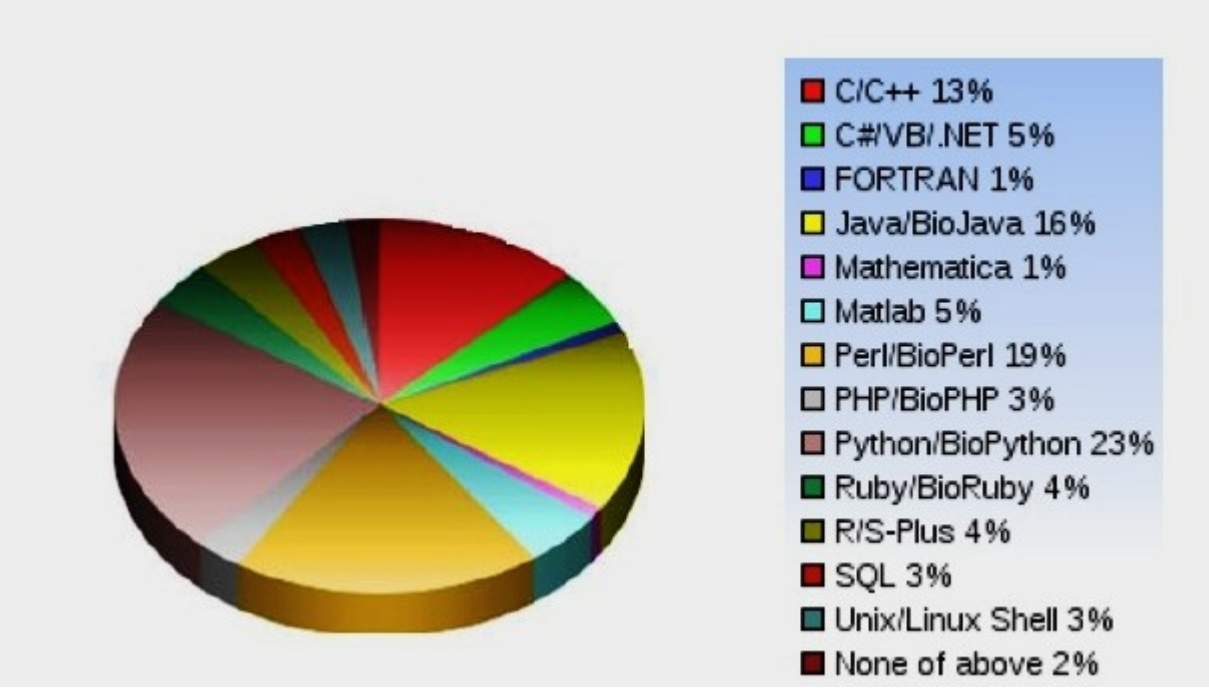
Bioinformatics Programming in Python

Σε αυτό το σημείο θα εξετάσουμε μια υποκατηγορία της Python, την Biopython με την οποία θέλησα να μελετήσω μαθαίνοντας Python διότι την θεωρώ άκρως ενδιαφέρουσα και σημαντική για όσους επιθυμούν να ασχοληθουν με Biomedical Engineering ή Bioinformatics. Στο παρον project θα κάνουμε μια εισαγωγή, όποιος επιθυμεί να ασχοληθεί εκτενέστερα στο διαδίκτυο υπάρχει μια πληθώρα πηγών για εντρύφιση στο συγκεκριμένο κλαδο.

Η Biopython διευκολύνει τους μηχανικούς που ασχολούνται με Bioinformatics γιατί δεν ανησυχείς για τα παρακάτω :

→ Παράξενα σύμβολα (~=, <>, eq, '\n', {}...)
→ Εναλλακτική σύνταξη για να κάνει την ίδια λειτουργία
→ Ορισμός τύπος μεταβλητών
→ Διαχείριση μνήμης
→ IO, call by reference/value κτλ…

Στο διάγραμμα που ακολουθεί βλέπουμε την αναλογία των γλωσσών προγραμματισμού που χρησιμοποιούνται στον κλάδο της Βιοπληροφορικής



Η BioPython είναι μια συλλογή τυποποιημένων libraries σε python για τη βιοπληροφορική .

Χαρακτηριστικά :

- Ανοιχτού κώδικα (Open Source,)
- Cross platform : Linux, Windows, Mac OS X
- Συναφή projects: BioPerl, BioRuby, BioJava

Πλεονεκτήματα χρήσης open sourcelibraries :

⇨ Αναπαραγωγιμότητα
⇨ Ευκολία σύγκρισης αποτελεσμάτων
⇨ Λιγότερα λάθη
⇨ Λιγότερος χρόνος υλοποίησης

Εφαρμογές της Biopython

1. Διαχείριση και επεξεργασία ακολουθιών
2. BLAST (τοπική και online)

3. Web databases (NCBI's EUtils)
4. Επιλογήcommand lineδιεπαφών (e.g. clustalw)
5. Ομαδοποίηση (Bio.Cluster)
6. Φυλογενετική(Bio.Nexus)
7. Δομή Πρωτεϊνών(Bio.PDB)
8. Υποστήριξη βάσεων (Bio.SQL)
9. Γενετική Πληθυσμού (Bio.PopGen)

Επιπλέον modules

NumPy

- ✓ N-dimensional μητρώα
- ✓ Συναρτήσεις γραμμικής άλγεβρας
- ✓ Μετασχηματισμούς Fourier
- ✓ Γεννήτορες τυχαίων αριθμών

SciPy

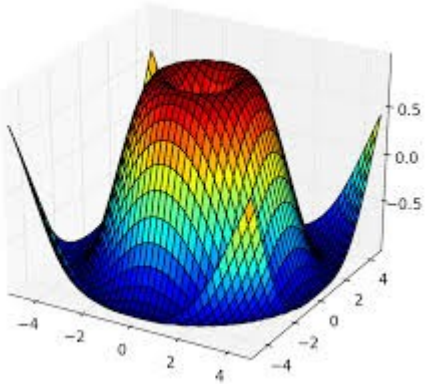
- ✓ Στατιστικά πακέτα
- ✓ Αριθμητική ολοκλήρωση
- ✓ Γραμμική άλγεβρα
- ✓ Επεξεργασία σημάτων
- ✓ Επεξεργασία εικόνας
- ✓ Γενετικούς αλγόριθμους
- ✓ Επιλυτές Διαφορικών εξισώσεων

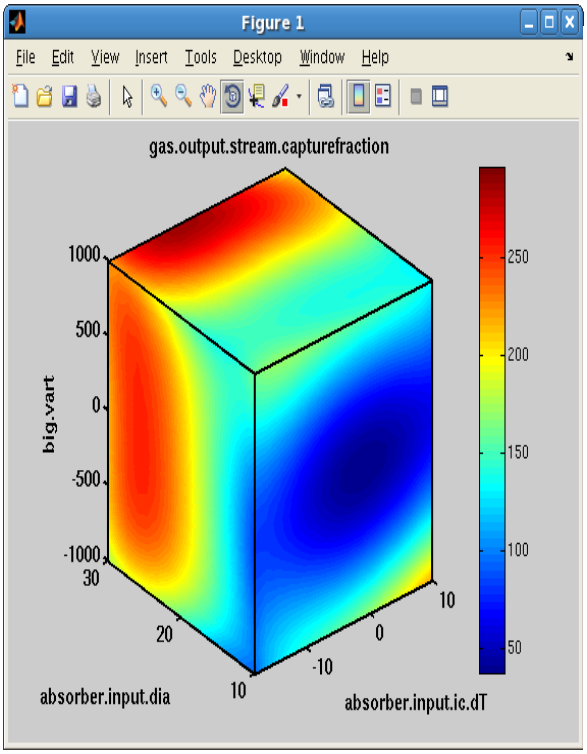
Matplotlib

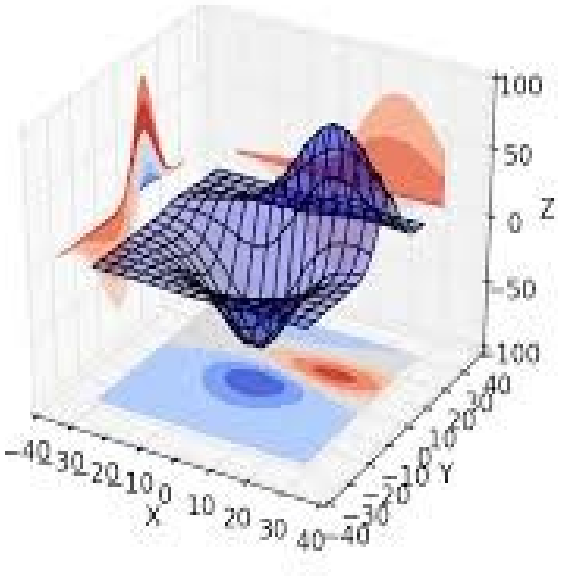
Είναι βιβλιοθήκη για το σχεδιασμό 2D και 3Dδιαγραμμάτων.

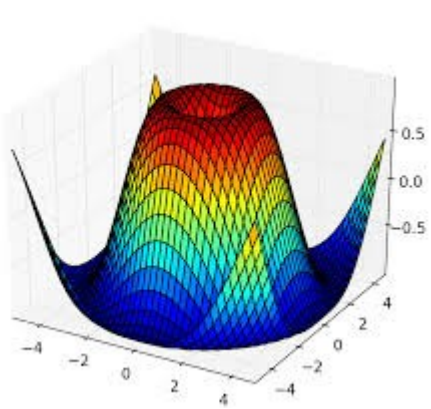
<u>Πλεονεκτήματα</u>

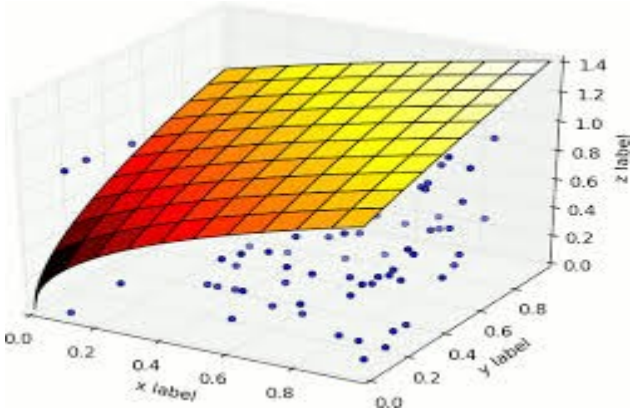
- ✓ Ευκολία χρήσης
- ✓ Documentationκαι tutorials
- ✓ Αποδοτικό visualization.











Άλλες εφαρμογές και βιβλιοθήκες

- Django (Web frameworks)
- Plone (Content ManagementSystem)
- ReportLab (PDF generation)
- MPI for Python (Παράλληλος Προγραμματισμός)
- SymPy (Συμβολικά Μαθηματικά)
- Python/R interface (στατιστική ανάλυση)
- SWIG(Simplified Wrapper and Interface Generator)
- Pygr (βάση δεδομένων γραφικών)
- PysCeS (Προσομοίωση των κυτταρικών συστημάτων)
- SloppyCell (Προσομοίωση βιομοριακών δικτύων)

Biopython –Sequence objects

```
>>> from Bio.Seq import Seq
>>> my_seq = Seq("ACTAGACTGGT")
>>> my_seq
Seq('ACTAGACTGGT', Alphabet())
>>> print my_seq
AGTACACTGGT
>>> my_seq.alphabet
Alphabet()
```

******Λειτουργούν ως strings αλλά έχουν περισσότερες ιδιότητες

Βασικές συναρτήσεις (Biopython –Seq Functions)

complement() :συμπληρωματική

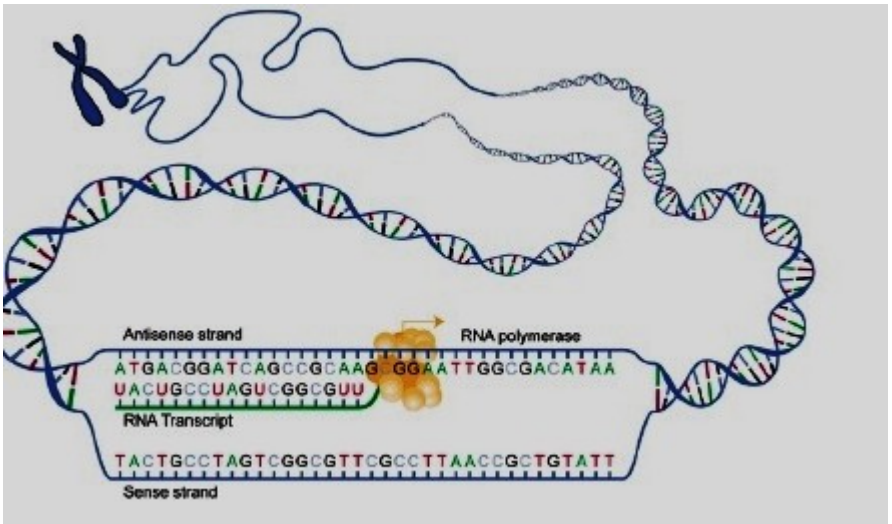
reverse_complement(): αντίστροφη συμπληρωματική

transcribe() : DNA to RNA

back_transcribe() : RNA to DNA

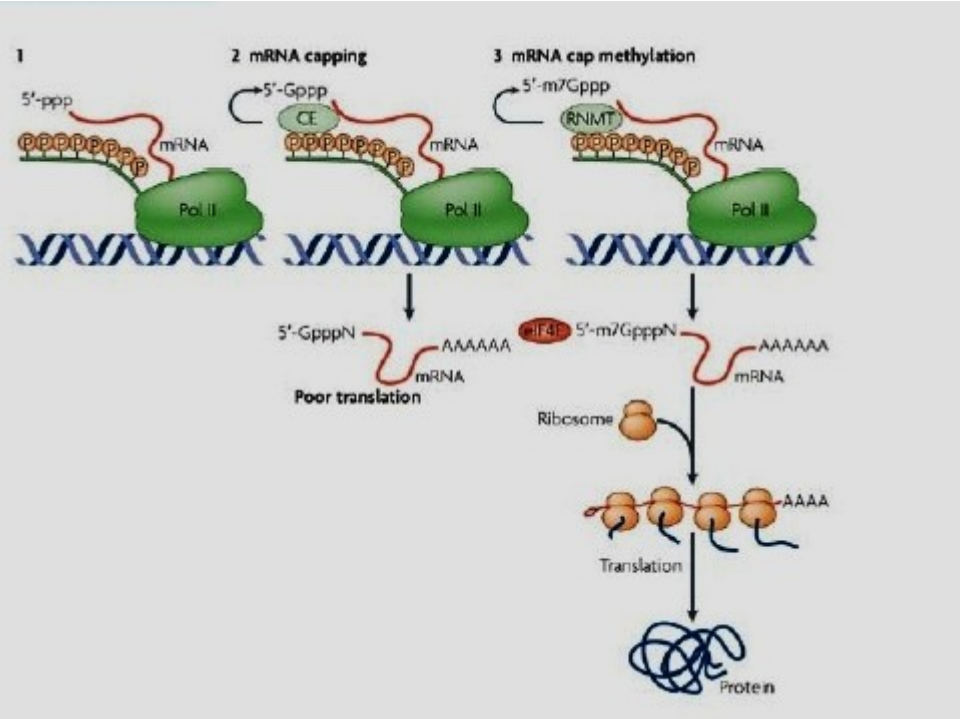
translate() : DNA to protein

Transcription



```
>>> from Bio.Seq import Seq

>>> from Bio.Alphabet import IUPAC

>>> coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG",
IUPAC.unambiguous_dna)

>>> coding_dna

Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG',
IUPACUnambiguousDNA())

>>> messenger_rna = coding_dna.transcribe()

>>> messenger_rna

Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG',
IUPACUnambiguousRNA())
```

Translation




```
>>> from Bio.Seq import Seq

>>> from Bio.Alphabet import IUPAC

>>> messenger_rna = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG",
IUPAC.unambiguous_rna)

>>> messenger_rna
```

```
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG',
IUPACUnambiguousRNA())

>>> messenger_rna.translate()

Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

Βασικές λειτουργίες

parse : όλων των στοιχείων ενός βιολογικού αρχείου

read: διάβασμα ενός στοιχείου

write :εγγραφή στοιχείων στο αρχείο

convert : μετατροπή αρχείου από την μια μορφή στην άλλη

Parsing & read από αρχείο

```
from Bio import SeqIO

handle = open("ls_orchid.fasta")

for seq_record in SeqIO.parse(handle, "fasta"):

print seq_record.id

print repr(seq_record.seq)

print len(seq_record)

handle.close()
```

Parsing & read από αρχείο με iterator

```
from Bio import SeqIO

handle = open("ls_orchid.fasta")

record_iterator = SeqIO.parse(handle, "fasta")

first_record = record_iterator.next()

print first_record.id

print first_record.description
```

```
second_record = record_iterator.next()

print second_record.id

print second_record.description
```

Parsing & read από το διαδίκτυο

```
from Bio import Entrez

from Bio import SeqIO

Entrez.email = "A.N.Other@example.com"

handle = Entrez.efetch(db="nucleotide", rettype="fasta",
id="6273291")

seq_record = SeqIO.read(handle, "fasta")

handle.close()

print "%s with %i features" % (seq_record.id,
len(seq_record.features))
```

Μετατροπή αρχείων διαφορετικό format

```
from Bio import SeqIO

from StringIO import StringIO

handle1 =open(“my_example.fasta")

handle2 =open(“ls_orchid.gbk")

count = SeqIO.convert(handle2, "genbank", handle1, "fasta")

handle1.close()

handle2.close()
```

Biopython –NCBI's Entrez

Entrez → Σύστημα ανάκτησης πληροφορίας από τις βάσεις δεδομένων της NCBI.

```
from Bio import Entrez

Entrez.email = "A.N.Other@example.com"

handle = Entrez.einfo()

record = Entrez.read(handle)

print record["DbList"]
```

Περιεχόμενα βάσης :

['pubmed', 'protein', 'nucleotide', 'nuccore', 'nucgss', 'nucest',

'structure', 'genome', 'books', 'cancerchromosomes', 'cdd', 'gap',

'domains', 'gene', 'genomeprj', 'gensat', 'geo', 'gds', 'homologene',

'journals', 'mesh', 'ncbisearch', 'nlmcatalog', 'omia', 'omim', 'pmc',

'popset', 'probe', 'proteinclusters', 'pcassay', 'pccompound',

'pcsubstance', 'snp', 'taxonomy', 'toolkit', 'unigene', 'unists']

Αναζήτηση στην βάση

```
from Bio import Entrez

Entrez.email = "A.N.Other@example.com"

handle = Entrez.esearch(db="nucleotide",term="Cypripedioideae[Orgn]
AND matK[Gene]")

record = Entrez.read(handle)

print record["Count"]

print record["IdList"]
```

Βιβλιογραφία

- ✓ http://www.pythonware.com
- ✓ http://www.python.org/
- ✓ Dive into Python, by Mark Pilgrim, Guide to Python 3
- ✓ A byte of Python, by Swaroop
  http://www.ibiblio.org/g2swap/byteofpython/files/120/byteofpython_120.pdf
- ✓ http://en.wikipedia.org/wiki/Python_(programming_language)
- ✓ http://learnpythonthehardway.org/book/
- ✓ http://en.wikibooks.org/wiki/Python_Programming
- ✓ http://www.codecademy.com/tracks/python
- ✓ http://www.youtube.com/playlist?list=PLEA1FEF17E1E5C0DA
- ✓ http://swaroopch.com/notes/python_el-Τα_βασικά/
- ✓ www.biopython.org
- ✓ http://taspython.eu/
- ✓ http://biopython.org/wiki/Main_Page
- ✓ http://biopython.org/DIST/docs/tutorial/Tutorial.html
- ✓ http://www.pasteur.fr/recherche/unites/sis/formation/python/index.html