ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Εργαστηριακός Οδηγός Γλώσσας Προγραμματισμού Python

Beautiful is better than ugly. **Explicit** is better than implicit. **Simple** is better than complex. **Complex** is better than complicated. **Flat** is better than nested. **Sparse** is better than dense. **Readability** counts. *Special cases* aren't special enough to break the rules.

Although **practicality** beats purity. *Errors* should never pass silently. Unless **explicitly** silenced. In the face of *ambiguity*, **refuse** the temptation to guess. There should be **one** — and preferably only one — obvious way to do it. Although that way may not be obvious at first *unless you're Dutch*. **Now** is better than never. Although never is **often** better than *right* now. If the implementation is *hard* to explain, it's a **bad** idea. If the implementation is *easy* to explain, it *may* be a **good** idea. **Namespaces** are one *honking great* idea — let's do more of those!

Beautiful is better than ugly. **Explicit** is better than implicit. **Simple** is better than complex. **Complex** is better than complicated. **Flat** is better than nested. **Sparse** is better than dense. **Readability** counts. *Special cases* aren't special enough to break the rules.

Although **practicality** beats purity. *Errors* should never pass silently. Unless **explicitly** silenced. In the face of *ambiguity*, **refuse** the temptation to guess. There should be **one** — and preferably only one — obvious way to do it. Although that way may not be obvious at first *unless you're Dutch*. **Now** is better than never. Although never is **often** better than *right* now. If the implementation is *hard* to explain, it's a **bad** idea. If the implementation is *easy* to explain, it *may* be a **good** idea. **Namespaces** are one *honking great* idea — let's do more of those!

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΕΙΡΑΙΑ

ΣΠΟΥΔΑΣ
ΚΟΥΛΟΥΡΑΣ ΗΛ. ΓΡΗΓΟΡΙΟΣ - ΒΑΚΕΡΛΗΣ Γ. ΔΗΜΗΤΡ
ΕΙΣΗΓΗΤΗΣ ΚΑΘΗΓΗ
Κος ΑΛΑΤΣΑΘΙΑΝΟΣ ΣΤΑΜΑΤ
ΗΛΕΚΤΡΟΝΙΚΑ ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜ

Περιεχόμενα

Ευχαριστίες

Ο παρόν οδηγός αποτελεί ένα εισαγωγικό βήμα στην εκμάθηση της συγκεκριμένης γλώσσας προγραμματισμού και φιλοδοξεί να καλύψει το κενό που υπάρχει στην ελληνική βιβλιογραφία.

Με την ευκαιρία της κατάθεσης της πτυχιακής μας εργασίας θα θέλαμε να ευχαριστήσουμε τα εξής άτομα:

- Τον υπεύθυνο επιβλέπων καθηγητή κο Αλατσαθιανό Σταμάτη για τη σημαντική του καθοδήγηση στην εργασία αυτή, τη αδιάκοπη συνεργασία του όλο το διάστημα εκπόνησης της, τη φιλική αντιμετώπιση του, καθώς και την ανάπτυξη ενδιαφέροντος για την ενασχόληση μας με τη συγκεκριμένη γλώσσα προγραμματισμού.

- Την Ελληνική Κοινότητα Προγραμματιστών Python για την αμέριστη βοήθεια με παραδείγματα, υποδείξεις και επεξηγήσεις σε όσα σημεία χρειαστήκαμε βοήθεια.

- Τους συμφοιτητές και φίλους μας για τη συντροφιά και συνεργασία όλα αυτά τα χρόνια φοίτησης μας στο Ίδρυμα.

Οι σπουδαστές

Κουλουράς Ηλ. Γρηγόριος
Βακερλής Γ. Δημήτριος

1. Εισαγωγή

Η Python είναι μια από εκείνες τις σπάνιες γλώσσες που ισχυρίζονται ότι είναι και **απλές** και **ισχυρές**. Θα εκπλαγείτε ευχάριστα από την ευκολία με την οποία θα συγκεντρώνεστε στη λύση ενός προβλήματος παρά στο συντακτικό και τη δομή της γλώσσας στην οποία προγραμματίζετε.

Η επίσημη εισαγωγή στην Python είναι:

> Η Python είναι μια εύκολη στην εκμάθηση, 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.

Θα εξηγήσω τα περισσότερα από αυτά τα χαρακτηριστικά λεπτομερώς στις επόμενες ενότητες.

Σημείωση

> Ο Guido van Rossum, ο δημιουργός της γλώσσας Python, ονόμασε τη γλώσσα από την εκπομπή "Monty Python's Flying Circus" του BBC. Δεν του αρέσουν ιδιαίτερα τα φίδια τα οποία σκοτώνουν άλλα ζώα για φαγητό τυλίγοντας το σώμα τους γύρω τους και συντρίβοντάς τα.

1.1. 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

> Όπως θα δείτε, είναι εξαιρετικά απλό να ξεκινήσετε με την Python. Η Python έχει μια ασυνήθιστα απλή σύνταξη, όπως έχει ήδη αναφερθεί.

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.

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.

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 !

Διερμηνευόμενη

Εδώ χρειάζονται μερικές εξηγήσεις.

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!

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.

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.

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 [1] , η Twisted [2], η Python Imaging Library [3] 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 όχι Perl;

Αν δεν το ξέρατε ήδη, η Perl είναι μια ακόμα εξαιρετικά ισχυρή, δημοφιλής, ανοικτού κώδικα, διερμηνεύσιμη γλώσσα προγραμματισμού.

Αν είχατε προσπαθήσει ποτέ να γράψετε ένα μεγάλο πρόγραμμα σε Perl, θα είχατε απαντήσει αυτή την ερώτηση από μόνοι σας! Με άλλα λόγια, 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 [4] -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 [5].

Γιατί όχι Ruby;

Αν δεν το ξέρατε ήδη, η Ruby είναι μια ακόμα δημοφιλής, ανοικτού κώδικα, διερμηνεύσιμη γλώσσα προγραμματισμού.

Αν χρησιμοποιείτε ήδη τη Ruby και σας αρέσει, τότε χωρίς αμφιβολία σας προτείνω να συνεχίσετε να τη χρησιμοποιείτε.

Για τους υπόλοιπους ανθρώπους που δεν την έχουν χρησιμοποιήσει ακόμα και προσπαθούν να αποφασίσουν αν θα μάθουν Python ή Ruby, τότε σας προτείνω την Python, καθαρά από την άποψη της ευκολίας εκμάθησης. Προσωπικά το βρήκα δύσκολο να κατανοήσω τη γλώσσα Ruby, αλλά όλοι οι άνθρωποι που την καταλαβαίνουν εξυμνούν την ομορφιά της γλώσσας. Δυστυχώς, εγώ δεν είμαι τόσο τυχερός.

1.2. Τι λένε οι προγραμματιστές

Ίσως να σας ενδιαφέρει να διαβάσετε τι λένε οι σπουδαίοι hackers όπως ο ESR 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 [6] (άρθρο στα αγγλικά). Αυτό το άρθρο ήταν η πραγματική έμπνευση πίσω από την πρώτη μου επαφή με την Python.
- 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. Διαβάστε

την πλήρη συνέντευξή του [7] (στα αγγλικά) για περισσότερες λεπτομέρειες.

- Ο **Peter Norvig** είναι ένας γνωστός συγγραφέας της Lisp και Διευθυντής ποιότητας αναζητήσεων στο Google (ευχαριστώ τον Guido van Rossum που μου το ανέφερε αυτό). Λέει ότι η Python πάντα ήταν ένα εσωτερικό κομμάτι του Google. Μπορείτε να το επιβεβαιώσετε αυτό κοιτώντας τις αγγελίες Google Jobs [8] στις οποίες αναγράφεται η γνώση της Python ως απαιτούμενη για μηχανικούς λογισμικού.
-

1.3. Περί της Python 3.0

Η Python 3.0 είναι η νέα έκδοση της γλώσσας. Μερικές φορές θα τη δείτε ως Python 3000 ή Py3K.

Ο κύριος λόγος για μια μεγάλη νέα έκδοση της Python είναι να απομακρυνθούν όλα τα μικροπροβλήματα και ελαττώματα που συσσωρεύθηκαν με τα χρόνια, και να γίνει η γλώσσα ακόμα πιο καθαρή.

Αν ήδη έχετε αρκετό κώδικα σε Python 2.x, τότε υπάρχει μια εφαρμογή που θα σας βοηθήσει να μετατρέψετε τον κώδικα από 2.x σε 3.x [9] (σελίδα στα Αγγλικά).

Περισσότερες λεπτομέρειες στα (όλοι οι σύνδεσμοι στα Αγγλικά):

- Εισαγωγή του Guido van Rossum [10]
- Τι είναι καινούριο στην Python 2.6 [11] (χαρακτηριστικά που είναι πολύ διαφορετικά από προηγούμενες εκδόσεις Python 2.x και που πιθανότατα θα συμπεριληφθούν στην Python 3.0)
- Τι είναι καινούριο στην Python 3.0 [12]
- Χρονοδιάγραμμα έκδοσης της Python 2.6 και της 3.0 [13]
- Python 3000 (η επίσημη λίστα από προτεινόμενες αλλαγές) [14]
- Διάφορα σχέδια για την Python 3.0 [15]
- Νέα της Python (λεπτομερής λίστα αλλαγών) [16]

2 Εγκατάσταση

Αν έχετε την Python 2.x ήδη εγκατεστημένη, **δε χρειάζεται** να την απεγκαταστήσετε για να εγκαταστήσετε την Python 3.0. Μπορείτε να τις έχετε και τις δύο εγκατεστημένες ταυτόχρονα.

2.1. Για χρήστες Linux και BSD

Αν χρησιμοποιείτε μια διανομή Linux όπως το Ubuntu, το Fedora, το OpenSUSE ή {εισάγετε την επιλογή σας εδώ}, 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
```

Σημείωση

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.

- Μπορείτε να μεταγλωττίσετε την Python από τον πηγαίο κώδικα [1] και να την εγκαταστήσετε. Οι οδηγίες μεταγλώττισης παρέχονται στον ιστότοπο.
- 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.

2.2. Για χρήστες Windows

Επισκεφθείτε το http:/ / www. python. org/ download/ releases/ 3. 1/ και κατεβάστε την πιο πρόσφατη έκδοση από αυτόν τον ιστότοπο, η οποία ήταν η 3.1 για 32 bit [2] και η 3.1 για 64 bit [3] όταν γράφτηκε αυτό το κείμενο.
Αυτά τα αρχεία είναι μόνο 13 MB το οποίο είναι πολύ μικρό σε σχέση με τις περισσότερες γλώσσες προγραμματισμού ή λογισμικά. Η εγκατάσταση είναι όπως οποιοδήποτε άλλο λογισμικό για Windows.
Προσοχή

> Όταν σας δοθεί η επιλογή να αποεπιλέξετε οποιαδήποτε "προαιρετικά" συστατικά, μην αποεπιλέξετε κανένα! Μερικά από αυτά τα συστατικά μπορεί να σας φανούν χρήσιμα, και ειδικά το IDLE.

Ένα ενδιαφέρον γεγονός είναι ότι η πλειοψηφία των λήψεων της Python από τον ιστότοπο γίνεται από χρήστες των Windows. Φυσικά, αυτό δεν περιγράφει όλη την εικόνα αφού σχεδόν όλοι οι χρήστες Linux έχουν ήδη την Python εγκατεστημένη στα συστήματά τους από προεπιλογή.

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.

2.3. 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.

Σύνοψη

Για ένα σύστημα Linux, πιθανότατα έχετε ήδη την Python εγκατεστημένη στο σύστημά σας. Σε αντίθετη περίπτωση, μπορείτε να την εγκαταστήσετε από το διαχειριστή πακέτων λογισμικού που συνοδεύει τη διανομή σας. 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 έχετε την Python εγκατεστημένη το σύστημά σας.

Στη συνέχεια, θα γράψουμε το πρώτο μας πρόγραμμα σε Python.

3. Τα πρώτα βήματα

3.1. Εισαγωγή

Τώρα θα δούμε πώς μπορούμε να τρέξουμε το παραδοσιακό πρόγραμμα "Χαίρε, Κόσμε!" (Hello, World!) στην Python. Αυτό θα σας διδάξει πώς να γράφετε, να αποθηκεύετε και να τρέχετε προγράμματα Python.

Υπάρχουν δύο τρόποι για να χρησιμοποιήσετε την Python ώστε να τρέξετε το πρόγραμμά σας: χρησιμοποιώντας την κονσόλα του διαδραστικού διερμηνευτή (interactive interpreter prompt) ή χρησιμοποιώντας ένα αρχείο πηγαίου κώδικα (source code file). Τώρα θα δούμε πώς χρησιμοποιούνται οι δύο αυτές μέθοδοι.

Χρήση της κονσόλας διερμηνευτή

Εκκινήστε το διερμηνευτή (interpreter) από τη γραμμή εντολών πληκτρολογώντας την εντολή python3 .

Οι χρήστες Windows, μπορείτε να τρέξετε το διερμηνευτή σε γραμμή εντολών αν έχετε ορίσει τη μεταβλητή περιβάλλοντος PATH κατάλληλα.
Εάν χρησιμοποιείτε το IDLE, επιλέξτε Start → Programs → Python 3.1 → IDLE (Python GUI).
Τώρα εισάγετε print('Χαίρε, Κόσμε!') και πιέστε το πλήκτρο Enter. Θα πρέπει να δείτε τις λέξεις Hello World ως αποτέλεσμα.

```
Python 3.0.1+ (r301:69556, Apr 15 2009, 15:59:22) [GCC 4.3.3]
on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> print('Χαίρε, Κόσμε!') Χαίρε,
Κόσμε!
>>>
```

Παρατηρήστε ότι 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. Εδώ, της δίνουμε το κείμενο Χαίρε, Κόσμε! και αυτό τυπώνεται στην οθόνη.

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.

3.2. 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 αυτοκίνητο 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.

Εάν χρησιμοποιείτε Windows, τότε προτείνω να χρησιμοποιήσετε 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.

Εάν χρησιμοποιείτε Linux/FreeBSD, τότε έχετε πλήθος επιλογών για τον επεξεργαστή που θα χρησιμοποιήσετε. Εάν ξεκινάτε μόλις τώρα τον προγραμματισμό, πιθανόν θα θέλετε να χρησιμοποιήσετε το Geany [1]. 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.

Εάν είστε έμπειρος προγραμματιστής, τότε ήδη θα χρησιμοποιείτε το Vim ή τον Emacs. Δε χρειάζεται να πούμε ότι πρόκειται για δύο από τους πιο ισχυρούς επεξεργαστές κώδικα και θα οφεληθείτε από τη χρήση τους για τη συγγραφή των Python προγραμμάτων σας. Προσωπικά χρησιμοποιώ το Vim για τα περισσότερα προγράμματά μου. Σε περίπτωση που είστε διατεθειμένοι να αφιερώσετε χρόνο ώστε να μάθετε το Vim ή τον Emacs, τότε συνιστώ ανεπιφύλακτα να μάθετε να χρησιμοποιείτε οποιονδήποτε από τους δύο καθώς θα σας φανούν πολύ χρήσιμοι στην πορεία.

Σε αυτό το βιβλίο, θα χρησιμοποιήσουμε το **IDLE**, το IDE (Integrated Development Environment - Ολοκληρωμένο Περιβάλλον Ανάπτυξης) και επεξεργαστή κώδικα που επιλέξαμε. Το IDLE εγκαθίσταται εξ ορισμού από το πρόγραμμα εγκατάστασης της Python σε Windows και OS X. Είναι επίσης διαθέσιμο για εγκατάσταση σε Linux [2] και BSD στα αντίστοιχα αποθετήρια (repositories).
Θα δούμε πώς χρησιμοποιούμε το IDLE στο επόμενο τμήμα. Για περισσότερες πληροφορίες, παρακαλώ ανατρέξτε στην τεκμηρίωση του IDLE [3].
Αν παρ' όλα αυτά θέλετε να εξερευνήσετε άλλες επιλογές όσον αφορά τον επεξεργαστή κώδικα, δείτε τη λίστα επεξεργαστών κώδικα για Python [4] και κάντε την επιλογή σας. Μπορείτε επίσης να επιλέξετε ένα IDE (Ολοκληρωμένο Περιβάλλον Ανάπτυξης) για Python. Δείτε την ολοκληρωμένη λίστα IDE που υποστηρίζουν την Python [5] για περισσότερες λεπτομέρειες. Αφού ξεκινήσετε να γράφετε μεγάλα προγράμματα σε Python, τα ολοκληρωμένα περιβάλλοντα ανάπτυξης μπορούν να φανούν εξαιρετικά χρήσιμα.

Επαναλαμβάνω άλλη μία φορά, παρακαλώ επιλέξτε έναν κατάλληλο επεξεργαστή για Python - 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 [6].

Για τους χρήστες του Emacs

Υπάρχει μία καλή εισαγωγή στο πώς να μετατρέψετε τον Emacs σε ένα ισχυρό Python IDE από τον Ryan McGuire [7].

3.3. Χρησιμοποιώντας ένα αρχείο πηγαίου κώδικα

Ας επιστρέψουμε στον προγραμματισμό. 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 [8], είναι «το παραδοσιακό ξόρκι προς τους θεούς του προγραμματισμού για να μας βοηθήσουν να μάθουμε τη γλώσσα καλύτερα» :) .

Εκκινήστε τον επεξεργαστή κώδικα που επιλέξατε, εισάγετε το πρόγραμμα που ακολουθεί και αποθηκεύστε το ως helloworld.py

Αν χρησιμοποιείτε το IDLE, επιλέξτε File → New Window και εισάγετε το ακόλουθο πρόγραμμα. Έπειτα επιλέξτε File → Save.

```
#!/usr/bin/python3 #Filename:
helloworld.py

print('Χαίρε, Κόσμε!')
```

Τρέξτε αυτό το πρόγραμμα ανοίγοντας ένα κέλυφος (τερματικό στο Linux ή γραμμή εντολών στο DOS) και εισάγοντας την εντολή python3 helloworld.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 helloworld.py.

Αν χρησιμοποιείται το IDLE, επιλέξτε από το μενού Run → Run Module ή χρησιμοποιήστε τη συντόμευση πληκτρολογίου F5.

Το αποτέλεσμα είναι αυτό :

```
$ python3 helloworld.py Χαίρε,
Κόσμε!
```

Αν πήρατε την έξοδο που φαίνεται πιο πάνω, συγχαρητήρια! Τρέξατε επιτυχώς το πρώτο σας πρόγραμμα σε Python.

Σε περίπτωση που πήρατε κάποιο μήνυμα λάθους, παρακαλώ ξαναγράψτε το παραπάνω πρόγραμμα *ακριβώς* όπως το βλέπετε πιο πάνω και τρέξτε το ξανά. Σημειώστε οτι η Python κάνει διάκριση πεζών-κεφαλαίων π.χ. το print δεν είναι το ίδιο με το Print -παρατηρήστε το πεζό p στο πρώτο και το κεφαλαίο P στο τελευταίο. Επίσης, βεβαιωθείτε οτι δεν υπάρχουν κενά ή στηλοθέτες (tabs) πριν τον πρώτο χαρακτήρα (γράμμα) σε κάθε γραμμή -θα δούμε γιατί αυτό είναι σημαντικό αργότερα.

Πώς δουλεύει

Ας εξετάσουμε τις 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. Αυτό εξηγείται λεπτομερώς στο επόμενο τμήμα. 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 helloworld.py .

Σημαντικό

> 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 'Χαίρε, Κόσμε!'. Θα μάθουμε περισσότερα σχετικά με τις συναρτήσεις σε → ένα από τα επόμενα κεφάλαια -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 'Χαίρε, Κόσμε!' το οποίο αποκαλούμε συμβολοσειρά (string) -μην ανησυχείτε, θα εξερευνήσουμε αυτές τις ορολογίες λεπτομερώς αργότερα.

3.4. 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 helloworld.py $
./helloworld.py
Χαίρε, Κόσμε!
```

Η εντολή chmod χρησιμοποιείται εδώ για να αλλάξουμε (*ch*ange) την κατάσταση (*mod*e) του αρχείου δίνοντας δικαιώματα εκτέλεσης (*e*xecute) σε όλους (*a*ll) 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).

Για να γίνουν τα πράγματα πιο διασκεδαστικά, μπορείτε να μετονομάσετε το αρχείο απλώς σε helloworld και να το εκτελείτε με ./helloworld και θα συνεχίσει να λειτουργεί εφ' όσον το σύστημα γνωρίζει οτι πρέπει να το τρέξει το πρόγραμμα χρησιμοποιώντας το διερμηνευτή του οποίου η τοποθεσία καθορίζεται στην πρώτη γραμμή του κώδικα.

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. Απλά τροποποιήστε 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 helloworld.py
/usr/local/bin/helloworld

$ helloworld
Χαίρε, Κόσμε!
```

Σημειώστε 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 σε τερματικό την εντολή PATH=$PATH:/home/swaroop/mydir όπου '/home/swaroop/mydir' είναι η διαδρομή που θέλετε να προσθέσετε.

Σημείωση

Η προσθήκη μιας διαδρομής στην $PATH με την εντολή PATH=$PATH:/home/swaroop/mydir 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.

Προσοχή

Όσο αφορά την Python ένα πρόγραμμα, ένα script ή λογισμικό όλα έχουν την ίδια σημασία.

3.5. Αναζητώντας βοήθεια

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.

Σημείωση

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.

4. Τα βασικά

Το να τυπώσετε απλά «Χαίρε, Κόσμε!» δεν είναι αρκετό, είναι; Θέλετε να δημιουργήσετε περισσότερα - θέλετε να κρατήσετε κάποια δεδομένα εισόδου, να τα χειριστείτε και να εξάγετε κάτι από αυτά. Μπορούμε να το επιτύχουμε αυτό στην Python με σταθερές και μεταβλητές.

Κυριολεκτικές σταθερές (Literal Constants)

Ένα παράδειγμα κυριολεκτικής σταθεράς είναι ένας αριθμός όπως 5, 1.23, 9.25e-3 ή μια συμβολοσειρά όπως 'Αυτή είναι μια συμβολοσειρά' ή "Μία 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.

4.1. Αριθμοί

Οι αριθμοί στην Python είναι τριών τύπων: οι ακέραιοι (integers), οι αριθμοί κινητής υποδιαστολής (floating point) και οι μιγαδικοί αριθμοί (complex numbers).

- Ένα παράδειγμα ακέραιου είναι το 2 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 (-5+4j) και (2.3 - 4.6j)

Σημείωση για έμπειρους προγραμματιστές

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

4.2. 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 [1].

Σημείωση για έμπειρους προγραμματιστές

> 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"). Για λεπτομέρειες, δείτε παρακαλώ τη σχετική συζήτηση στο Stack Overflow [2].
> Από προεπιλογή, όλες οι συμβολοσειρές είναι σε Unicode.

Μπορώ να σας εγγυηθώ ότι σχεδόν σε κάθε πρόγραμμα Python που θα γράψετε θα χρησιμοποιήσετε συμβολοσειρές, γι' αυτό προσέξτε ιδιαίτερα τα παρακάτω για τη χρήση τους στην Python.

Μονά εισαγωγικά

Μπορείτε να ορίσετε συμβολοσειρές με μονά εισαγωγικά, για παράδειγμα 'Με λένε Γιώργο και ποτέ δεν τραγουδάω'. 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 σε λένε;"

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
δεύτερη γραμμή.

"Πώς σε λένε;" το ρώτησα.
Μου είπε: 'Μποντ. Τζέμης Μποντ.'
'''
```

4.3. 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' άκουσα!. Δεν μπορείτε να το ορίσετε ως 'Δε σ' άκουσα!' 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\' άκουσα!'.

Ένας άλλος τρόπος ορισμού της συγκεκριμένης συμβολοσειράς είναι π.χ. "Δε σ' άκουσα!", 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"Newlines are indicated 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 παρακάτω στα διάφορα προγράμματα που θα εξετάσουμε.

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. Δεν υπάρχει πραγματική ανάγκη γι' αυτό και είμαι σίγουρος ότι δε θα σας λείψει.

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'.

4.4. Η μέθοδος format

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

Σημείωση

Οι μέθοδοι εξηγούνται καλύτερα σε → επόμενο κεφάλαιο. Προς το παρόν έχετε υπ' όψη ότι μετά την τελεία (.) ακολουθεί μια μέθοδος, η οποία εκτελεί μια λειτουργία σε οτιδήποτε προηγείται της τελείας. Στο επόμενο παράδειγμα, η μέθοδος format εκτελεί μια λειτουργία στη συμβολοσειρά που παρέχεται.

```
#!/usr/bin/python3
# Filename: str_format.py

age = 25
name = 'Αρνόλδος'

print('Ο {0} είναι {1} ετών.'.format(name, age))
print('Μα γιατί παίζει ο {0} με τον πύθωνα;'.format(name))
```

Έξοδος:

```
$ python str_format.py
Ο Αρνόλδος είναι 25 ετών.
Μα γιατί παίζει ο Αρνόλδος με τον πύθωνα;
```

Πώς λειτουργεί:

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: 'Ο ' + 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.

Η Python με τη μέθοδο format αντικαθιστά κάθε τιμή ορίσματος στη θέση της προδιαγραφής. Μπορούν να δηλωθούν λεπτομερέστερες προδιαγραφές όπως:

```
>>> '{0:.3}'.format(1/3) # δεκαδικός (.) ακρίβεια τριών ψηφίων
'0.333'
>>> '{0:_^11}'.format('hello') # γέμισμα με κάτω παύλες (_) με το κείμενο
κεντραρισμένο (^) σε πλάτος 11
'___hello___'

>>> 'Ο {name} έγραψε το {book}'.format(name='Swaroop', book='A Byte of Python') # βάσει
λέξεων-κλειδιών
'Ο Swaroop έγραψε το A Byte of Python'
```

Οι λεπτομέρειες αυτής της προδιαγραφής μορφοποίησης αναλύονται στο κείμενο Python Enhancement Proposal No. 3101 [3].

4.5. 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_ვძջ ... ςεζζ ... .
- 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. Στα επόμενα κεφάλαια, θα δούμε πώς μπορούμε να δημιουργήσουμε δικούς μας τύπους χρησιμοποιώντας → Κλάσεις.

4.6. Αντικείμενα

Να θυμάστε, 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.

Τώρα θα δούμε τη χρήση μεταβλητών μαζί με κυριολεκτικές σταθερές. Αποθηκεύστε το παρακάτω παράδειγμα και εκτελέστε το πρόγραμμα.

Πώς να γράφετε προγράμματα Python

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

1. Ανοίξτε τον αγαπημένο σας επεξεργαστή κώδικα.
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 χρησιμοποιήστε το IDLE για την εκτέλεση των προγραμμάτων. Μπορείτε να κάνετε επίσης τα αρχεία εκτελέσιμα όπως εξηγήθηκε προηγουμένως.

Παράδειγμα: Χρήση μεταβλητών και κυριολεκτικών σταθερών

```
# Filename : var.py

i = 5 print(i)
i = i + 1
print(i)


s = '''This is a multi-line string. This is the
second line.''' print(s)
```

Έξοδος:

```
$ python var.py 5 6


This is a multi-line string. This is the
second line.
```

Πώς λειτουργεί:

Να πώς λειτουργεί το πρόγραμμα. Πρώτα, εκχωρούμε την τιμή της κυριολεκτικής σταθεράς 5 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 5. 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 1 στην αποθηκευμένη τιμή του i και αποθηκεύεται ενημερωμένη εκ νέου. Κατόπιν την εκτυπώνουμε και όπως περιμένουμε, μας επιστρέφει την τιμή 6.
Παρομοίως, εκχωρούμε την κυριολεκτική σταθερά στη μεταβλητή s και εκτυπώνουμε στην οθόνη.

Σημείωση για 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).

4.7. 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. Για παράδειγμα, το

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

είναι ουσιαστικά ίδιο με το

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

και το ίδιο μπορεί να γραφτεί ως

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

ή ακόμη και

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

Ωστόσο, **συστήνω** να συνηθίσετε 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, *δεν* έχω χρησιμοποιήσει ποτέ, ούτε έχω δει ερωτηματικό σε ένα πρόγραμμα Python.

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

```
    s = 'This is a string. \ This continues
    the string.' print(s)
```

Η έξοδος του παραπάνω κώδικα:

```
    This is a string. This continues the string.
```

Παρομοίως,

```
    print\
    (i)
```

είναι το ίδιο με

```
    print(i)
```

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**. Θα το δείτε στην πράξη όταν θα γράψουμε προγράμματα με λίστες στα επόμενα κεφάλαια.

4.8. 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)**. Θα δούμε παραδείγματα για τη σημασία των πλοκάδων στα επόμενα κεφάλαια.

Να θυμάστε πάντα ότι οι λάθος εσοχές μπορεί να προκαλέσουν σφάλματα. Για παράδειγμα:

```
i = 5
  print('Η τιμή είναι ', i) # Σφάλμα! Προσέξτε ένα διάστημα στην αρχή
της γραμμής
print('Επαναλαμβάνω, η τιμή είναι ', i)
```

Αν εκτελέσετε τον παραπάνω κώδικα, θα προκύψει το παρακάτω σφάλμα:

```
    File "whitespace.py", line 4
      print('Η τιμή είναι ', i) # Σφάλμα! Προσέξτε ένα διάστημα στην αρχή της γραμμής
      ^
IndentationError: unexpected indent
```

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 από την προεπιλεγμένη κύρια πλοκάδα που χρησιμοποιούσατε συνέχεια, φυσικά). Οι περιπτώσεις που μπορείτε να εφαρμόσετε νέες πλοκάδες θα αναπτυχθούν στα επόμενα κεφάλαια, όπως το → κεφάλαιο για τον έλεγχο ροής.

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 για προγραμματιστές στατικών γλωσσών

Η Python θα έχει πάντα εσοχή για πλοκάδες και ποτέ αγκύλες (braces). Εκτελέστε from __future__ import braces για περισότερες πληροφορίες.

Σύνοψη

Τώρα αφού διυλίσαμε τις λεπτομέρειες, μπορούμε να προχωρήσουμε σε πιο ενδιαφέροντα θέματα όπως τις εντολές ελέγχου ροής. Βεβαιωθείτε ότι είστε άνετοι με όσα διαβάσατε σε αυτό το κεφάλαιο.

5. Τελεστές και εκφράσεις

Εισαγωγή

Οι περισσότερες εντολές που γράφετε θα περιέχουν εκφράσεις (expressions). Ένα απλό παράδειγμα μίας έκφρασης είναι 2 + 3. 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 2 και το 3.

5.1. Τελεστές

Θα ρίξουμε μία γρήγορη ματιά στους τελεστές και τη χρήση τους:

Σημειώστε ότι μπορείτε να βρείτε την τιμή των εκφράσεων που δίνονται στα παραδείγματα χρησιμοποιώντας το διερμηνευτή διαδραστικά. Για παράδειγμα, για τον έλεγχο της έκφρασης 2 + 3, χρησιμοποιήστε την κονσόλα του διαδραστικού διερμηνευτή της Python:

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

Τελεστής	Όνομα	Εξήγηση	Παραδείγματα
+	Συν	Προσθέτει δύο αντικείμενα.	Το 3 + 5 δίνει 8. Το 'a' + 'b' δίνει 'ab'.
-	Μείον	Είτε δίνει έναν αρνητικό αριθμό, ή αφαιρεί έναν αριθμό από έναν άλλο.	Το -5.2 δίνει έναν αρνητικό αριθμό. Το 50 - 24 δίνει 26.
*	Επί	Δίνει το γινόμενο δύο αριθμών ή μιά συμβολοσειρά (string) επαναλαμβανόμενη τόσες φορές.	Το 2 * 3 δίνει 6. Το 'la' * 3 δίνει 'lalala'.
**	Δύναμη	Επιστρέφει το x υψωμένο στη δύναμη y.	Το 3 ** 4 δίνει 81 (δηλαδή 3 * 3 * 3 * 3).
/	Διά	Διαιρεί το x με το y.	Το 4 / 3 δίνει 1.3333333333333333.

//	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 Ή	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 επιστρέφει την τιμή του y.	x = False; y = True; x and y επιστρέφει False 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:

Μπορείτε να γράψετε το:

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

ως:

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

Παρατηρήστε ότι η δήλωση μεταβλητή = μεταβλητή τέλεση έκφραση γίνεται μεταβλητή τέλεση = έκφραση.

5.2. Σειρά υπολογισμού (προτεραιότητα τελεστών)

Ας πούμε ότι έχουμε μία έκφραση όπως 2 + 3 * 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 ακόλουθος πίνακας 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 [1], 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. Δείτε την Αλλαγή της σειράς υπολογισμού πιο κάτω για λεπτομέρειες.

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 OXI
**	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

Οι τελεστές τους οποίους δεν έχουμε ακόμη συναντήσει θα εξηγηθούν σε επόμενα κεφάλαια.

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

Αλλαγή της σειράς υπολογισμού

Για να κάνουμε τις εκφράσεις πιο ευανάγνωστες, μπορούμε να χρησιμοποιήσουμε παρενθέσεις. Για παράδειγμα, το 2 + (3 * 4) είναι σίγουρα πιο εύκολο να το καταλάβουμε από το 2 + 3 * 4 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 + (3 + 4)).

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)

Παράδειγμα:

```
#!/usr/bin/python
# Filename: expression.py

length = 5
breadth = 2

area = length * breadth
print('Area is', area)
print('Perimeter is', 2 * (length + breadth))
```

Έξοδος:

```
$ python expression.py Area
is 10
Perimeter is 14
```

Πώς δουλεύει:

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.

Σύνοψη

Είδαμε πως χρησιμοποιούμε τους τελεστές, τους τελεστέους και τις εκφράσεις -αυτά αποτελούν τα βασικά δομικά στοιχεία κάθε προγράμματος. Στη συνέχεια, θα δούμε πώς τα χρησιμοποιούμε στα προγράμματά μας χρησιμοποιώντας εντολές.

6. Έλεγχος ροής

Εισαγωγή

Στα προγράμματα που είδαμε μέχρι τώρα, υπήρξε πάντα μια σειρά εντολών, τις οποίες εκτελούσε πιστά η Python με την ίδια σειρά. Αν θέλουμε όμως 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;

Όπως ίσως να μαντέψατε, αυτό επιτυγχάνεται χρησιμοποιώντας εντολές ελέγχου ροής. Υπάρχουν τρεις εντολές ελέγχου ροής στην Python - if, for και while.

6.1. 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.
Παράδειγμα:

```
#!/usr/bin/python
#  -*- coding: utf-8 -*-
#  Filename: if.py

number = 23
guess = int(input('Εισάγετε έναν ακέραιο αριθμό: '))

if guess == number:
        print('Συγχαρητήρια, τον μαντέψατε.') # Νέα πλοκάδα (block) ξεκινάει εδώ

        print('(Αλλά δεν κερδίζετε και κανένα βραβείο!)') # Νέα πλοκάδα τελειώνει εδώ
elif guess < number:
        print('Όχι, είναι λίγο μεγαλύτερος.') # Άλλη μια πλοκάδα
        #  Μπορείτε να κάνετε ότι θέλετε εντός μιας πλοκάδας ...
else:
        print('Όχι, είναι λίγο μικρότερος.')
        #  πρέπει να ισχύει guess > number για να φθάσετε εδώ

print('Τέλος')
# Η τελευταία αυτή εντολή εκτελείται πάντα μετά την εκτέλεση της
εντολής if
```

Έξοδος:

```
$ python if.py
Εισάγετε έναν ακέραιο αριθμό: 50 Όχι,
είναι λίγο μικρότερος. Τέλος
```

```
$ python if.py
Εισάγετε έναν ακέραιο αριθμό: 22 Όχι,
είναι λίγο μεγαλύτερος. Τέλος

$ python if.py
Εισάγετε έναν ακέραιο αριθμό: 23
Συγχαρητήρια, τον μαντέψατε.

(Αλλά δεν κερδίζετε και κανένα βραβείο!) Τέλος
```

Πώς λειτουργεί:

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. 23. Ύστερα παίρνουμε την πρόγνωση του χρήστη χρησιμοποιώντας την συνάρτηση input(). Οι συναρτήσεις είναι απλά επαναχρησιμοποιήσιμα κομμάτια προγραμμάτων. Θα μάθουμε περισσότερα γι' αυτές στο επόμενο κεφάλαιο.

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. Ελπίζω να ακολουθείτε πάντα τον κανόνα "συνεπής χρήσης εσοχών". Τον ακολουθείτε;

Παρατηρήστε ότι η 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.

Θυμηθείτε 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.

Αν κι αυτό είναι ένα πολύ απλό πρόγραμμα, έχω επισημάνει πολλά πράγματα που πρέπει να λάβετε υπόψη ακόμα και σ' ένα τόσο απλό πρόγραμμα. Όλα αυτά είναι αρκετά στρωτά (κι εκπληκτικά απλά για όσους έχουν προϊστορικό σε C/C++) και απαιτούν να τα συνειδητοποιήσετε, αλλά μετά απ' αυτό θα αποκτήσετε ευχέρεια στην χρήση τους και θα σας φανούν "φυσικά".

Σημείωση για προγραμματιστές C/C++

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).

6.2. 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.

Παράδειγμα:

```
#!/usr/bin/python
#  -*- coding: utf-8 -*-
#  Filename: while.py


number = 23
running = True


while running:
      guess = int(input('Εισάγετε έναν ακέραιο αριθμό : '))

      if guess == number:
            print('Συγχαρητήρια, τον μαντέψατε.')

            running = False # αυτό κάνει τον βρόχο while να σταματήσει εδώ elif guess <
      number:

            print('Όχι, είναι λίγο μεγαλύτερος.') else:
            print('Όχι, είναι λίγο μικρότερος.')
else:
      print('Ο βρόχος while τερματίστηκε.')
      # Μπορείτε να προσθέσετε ότι άλλο θέλετε εδώ

print('Τέλος')
```

Έξοδος:

```
$ python while.py
Εισάγετε έναν ακέραιο αριθμό : 50 Όχι, είναι
λίγο μεγαλύτερος. Εισάγετε έναν ακέραιο
αριθμό : 22 Όχι, είναι λίγο μεγαλύτερος
Εισάγετε έναν ακέραιο αριθμό : 23
Συγχαρητήρια, τον μαντέψατε.
```

```
Ο βρόχος while τερματίστηκε. Τέλος
```

Πώς λειτουργεί:

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.

Σημείωση για προγραμματιστές C/C++

Θυμηθείτε ότι μπορείτε να έχετε έναν όρο else για τον βρόχο while.

6.3. 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.

Παράδειγμα:

```
#!/usr/bin/python
#  -*- coding: utf-8 -*-
#  Filename: for.py

for   i in range(1, 5): print(i)
else:
      print('Ο βρόχος loop τερματίστηκε')
```

Έξοδος:

```
$ python for.py 1 2
3 4

Ο βρόχος loop τερματίστηκε
```

Πώς λειτουργεί:

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(1,5) δίνει την ακολουθία [1, 2, 3, 4]. 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(1,5) είναι αντίστοιχο του for i in [1, 2, 3, 4] 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! Θα εξερευνήσουμε αυτή την ιδέα λεπτομερέστερα στα επόμενα κεφάλαια.

Σημείωση για προγραμματιστές C/C++/Java/C#

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.

6.4. 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.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: break.py

while True:
    s = (input('Enter something : ')) if s == 'quit':

        break
    print('Length of the string is', len(s)) print('Done')
```

Αποτέλεσμα:

```
$ python break.py
Enter something : Programming is fun Length of
the string is 18

Enter something : When the work is done Length of
the string is 21

Enter something : if you wanna make your work also fun: Length of the
string is 37

Enter something : use Python! Length of the
string is 12

Enter something : quit Done
```

Πώς λειτουργεί:

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 .

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

Swaroop's Poetic Python

Τα εισαχθέντα δεδομένα που χρησιμοποίησα εδώ είναι ένα μικρό ποιήμα που έγραψα με τίτλο **Swaroop's Poetic Python**:

```
Programming is fun When the
work is done

if you wanna make your work also fun: use
        Python!
```

6.5. 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.
Παράδειγμα:

```
#!/usr/bin/python
#  -*- coding: utf-8 -*-
#  Filename: continue.py

while True:
    s = input('Εισάγετε κάτι : ') if s == 'quit':
        break
    if len(s) < 3: print('Πολύ
        μικρό') continue
```

```
print('Το μήκος των εισαχθέντων είναι επαρκές')
# Προσθέστε οτιδήποτε άλλο εδώ
```

Έξοδος:

```
$ python test.py
Εισάγετε κάτι : a Πολύ
μικρό Εισάγετε κάτι : 12
Πολύ μικρό

Εισάγετε κάτι : abc
Το μήκος των εισαχθέντων είναι επαρκές Εισάγετε
κάτι : quit
```

Πώς λειτουργεί:

Σ' αυτό το πρόγραμμα, αποδεχόμαστε δεδομένα εισαγωγής από τον χρήστη, αλλά τα επεξεργαζόμαστε μόνο εάν έχουν τουλάχιστον μήκος 3 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.
Σημειώστε ότι η εντολή continue δουλεύει επίσης και με το βρόχο for.

Σύνοψη

Είδαμε πώς χρησιμοποιούμε τις τρεις εντολές ελέγχου ροής -την if, την while και την for μαζί με τις συσχετιζόμενες ε
break και continue. Αυτές ανήκουν στα πιο συχνά Python και γι' αυτό είναι ουσιώδες να εξοικειωθεί
κανείς με αυτές.
Στα επόμενα θα δούμε πως να δημιουργήσουμε και να χρησιμοποιήσουμε
συναρτήσεις.

7. Συναρτήσεις

Εισαγωγή

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 σε αυτό το κεφάλαιο.

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 (:). Παρακάτω ακολουθεί ένα απλό παράδειγμα με ένα σύνολο εντολών που αποτελούν μέρος αυτής της συνάρτησης. Με αυτό το παράδειγμα θα δείτε ότι στην πραγματικότητα είναι πολύ απλό:

Παράδειγμα:

```
#!/usr/bin/python
# Filename: function1.py


def sayHello():
    print('Hello World!') # σύνολο εντολών που ανήκουν στη συνάρτηση
# Τέλος της συνάρτησης


sayHello() # κλήση της συνάρτησης sayHello() # κλήση
της συνάρτησης ξανά
```

Έξοδος:

```
$ python function1.py Hello
World!
Hello World!
```

Πώς δουλεύει:

Ορίζουμε μια συνάρτηση με το όνομα sayHello ακολουθώντας τη σύνταξη όπως εξηγήσαμε 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.

7.1. 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*.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: func_param.py

def printMax(a, b): if a >
    b:

        print(a, 'είναι το μέγιστο') elif a == b:

        print(a, 'είναι ίσο με το', b) else:
        print(b, 'είναι το μέγιστο')

printMax(3, 4) # δίνουμε απ' ευθείας κυριολεκτικές τιμές


x = 5 y
= 7


printMax(x, y) # δίνουμε μεταβλητές σαν ορίσματα
```

Έξοδος:

```
$ python func_param.py 4
είναι το μέγιστο 7 είναι το
μέγιστο
```

Πώς δουλεύει:

Εδώ 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.

7.2. 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.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: func_local.py

x = 50

def func(x):
    print('Το x είναι', x) x = 2
    print('Άλλαξα το τοπικό x σε', x)

func(x)
print('Το x είναι ακόμα', x)
```

Έξοδος:

```
$ python func_local.py Το x
είναι 50

Άλλαξα το τοπικό x σε 2 Το x
είναι ακόμα 50
```

Πώς δουλεύει:

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 2 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.

7.3. Χρήση της εντολής 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.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: func_global.py

x = 50

def func():
    global x

    print('Το x είναι', x) x = 2
    print('Άλλαξα το καθολικό x σε', x)

func()
print('Η τιμή του x είναι', x)
```

Έξοδος:

```
$ python func_global.py Το x
είναι 50

Άλλαξα το καθολικό x σε 2 Η τιμή
του x είναι 2
```

Πώς δουλεύει:

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.

7.4. Χρήση της εντολής 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.
Ας δούμε ένα παράδειγμα:

```
#!/usr/bin/python
# Filename: func_nonlocal.py

def func_outer(): x =
    2
    print('Το x είναι', x)

    def func_inner():
        nonlocal x
        x = 5
```

```
    func_inner()
    print('Το τοπικό x άλλαξε σε', x)

func_outer()
```

Έξοδος:

```
$ python func_nonlocal.py Το x
είναι 2
Το τοπικό x άλλαξε σε 5
```

Πώς δουλεύει:

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.

Προσπαθήστε να αλλάζοντας το nonlocal x σε global x, και επίσης να μετακινώντας την ίδια την εντολή, να παρατηρήσετε τη διαφορά στην συμπεριφορά σε αυτές τις δύο περιπτώσεις.

7.5. 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.

Σημειώστε ότι η προεπιλεγμένη τιμή ορίσματος πρέπει να είναι μια σταθερά. Για την ακρίβεια η προεπιλεγμένη τιμή ορίσματος πρέπει να είναι αμετάβλητη -αυτό εξηγείται με λεπτομέρειες στα επόμενα κεφάλαια. Προς το παρόν, είναι αρκετό να το θυμάστε.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: func_default.py


def say(message, times = 1):
    print(message * times)



say('Hello')
say('World', 5)
```

Έξοδος:

```
$ python func_default.py Hello
WorldWorldWorldWorldWorld
```

Πώς δουλεύει:

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.

Σημαντικό

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.

7.6. 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*, πρώτον η χρήση της συνάρτησης γίνεται ευκολότερη επειδή δεν χρειάζεται να ανησυχούμε για τη διάταξη των ορισμάτων. Δεύτερον, μπορούμε να δώσουμε τιμές μόνο σε εκείνες τις παραμέτρους που θέλουμε, προνοώντας ότι οι άλλες παράμετροι έχουν τις προεπιλεγμένες τιμές ορίσματος.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: func_key.py

def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)

func(3, 7) func(25,
c=24) func(c=50,
a=100)
```

Έξοδος:

```
$ python func_key.py
a is 3 and b is 7 and c is 10 a is 25 and
b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

Πώς δουλεύει:

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(3, 7) η παράμετρος a παίρνει την τιμή 3, η παράμετρος b παίρνει την τιμή 7 και η c παίρνει την προεπιλεγμένη του 10.

Στην δεύτερη χρήση func(25, c=24), η μεταβλητή a παίρνει την τιμή του 25 εξαιτίας της θέσης του ορίσματος. Κατόπιν η παράμετρος c παίρνει την τιμή του 24 εξαιτίας της ονομασίας δηλ. ορίσματα με λέξεις κλειδιά. Η μεταβλητή b παίρνει την προεπιλεγμένη τιμή του 5.

Στην τρίτη χρήση func(c=50, a=100) ,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.

7.7. Παράμετροι VarArgs

Σημείωση

Ίσως θα έπρεπε να γράψουμε για αυτό σε επόμενα κεφάλαια επειδή δεν έχουμε μιλήσει ακόμα για τις λίστες και τα λεξικά.

Μερικές φορές ίσως θέλουμε να ορίσουμε μια συνάρτηση η οποία μπορεί να λάβει οποιοδήποτε αριθμό παραμέτρων και αυτό μπορεί να επιτευχθεί χρησιμοποιώντας το σύμβολο αστερίσκο *.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: total.py

def total(initial=5, *numbers, **keywords): count = initial

    for number in numbers:
        count += number
    for key in keywords:
        count += keywords[key]
    return count

print(total(10, 1, 2, 3, vegetables=50, fruits=100))
```

Έξοδος:

```
$ python total.py 166
```

Πώς δουλεύει:

Όταν δηλώνουμε μια παράμετρο με αστερίσκο, όπως το *param, τότε όλα τα ορίσματα των οποίων η θέση βρίσκεται από αυτό το σημείο μέχρι το τέλος, συγκεντρώνονται σαν λίστα που ονομάζεται 'param'. Παρόμοια όταν δηλώνουμε παράμετρο διπλό αστερίσκο όπως το **param, τότε όλα τα ορίσματα με λέξεις κλειδιά, από αυτό το σημείο μέχρι το τέλος συγκεντρώνονται σαν λεξικό που ονομάζεται 'param'.

Θα εξερευνήσουμε τις λίστες και τα λεξικά σε ένα → επόμενο κεφάλαιο.

7.8. 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.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: keyword_only.py

def total(initial=5, *numbers, vegetables): count = initial
```

```
    for number in numbers: count
        += number
    count += vegetables
    return count


print(total(10, 1, 2, 3, vegetables=50)) print(total(10,
1, 2, 3))

# Raises error because we have not supplied a default argument value for 'vegetables'
```

Έξοδος:

```
$ python keyword_only.py 66
Traceback (most recent call last):
  File "keyword_only.py", line 12, in <module> total(10, 1, 2,
3)
TypeError: total() needs keyword-only argument vegetables
```

Πώς δουλεύει:

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).

7.9. 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.
Παράδειγμα:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# Filename: func_return.py


def maximum(x, y): if x
    > y:

        return x
    elif x == y:
        return 'Οι αριθμοί είναι ίσοι' else:
        return y

print(maximum(2, 3))
```

Έξοδος:

```
$ python func_return.py 3
```

Πώς δουλεύει:

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*.

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.
Σημείωση

Υπάρχει μία ενσωματωμένη συνάρτηση που ονομάζεται max η οποία ήδη εφαρμόζει την λειτουργία εύρεσης του μέγιστου, οπότε χρησιμοποιείστε αυτή την ενσωματωμένη λειτουργία όποτε είναι δυνατόν.

7.10. 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!
Παράδειγμα:

```
#!/usr/bin/python
# Filename: func_doc.py

def printMax(x, y):
    '''Prints the maximum of two numbers.

    Οι δύο τιμές πρέπει να είναι ακέραιοι αριθμοί.'''
    x = int(x) # μετέτρεψε σε ακέραιους αριθμούς αν είναι δυνατόν y = int(y)


    if x > y:
        print(x, 'is maximum') else:

        print(y, 'is maximum')

printMax(3, 5)
print(printMax.__doc__)
```

Έξοδος:

```
$ python func_doc.py 5 is
maximum
Prints the maximum of two numbers.

    Οι δύο τιμές πρέπει να είναι ακέραιοι αριθμοί.
```

Πώς δουλεύει:

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), όπως θα μάθουμε στα αντίστοιχα κεφάλαια.

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. Θα μάθουμε περισσότερα για τα αντικείμενα στο κεφάλαιο για τις → κλάσεις.

Εάν έχετε χρησιμοποιήσει την help() στην Python, τότε έχετε δει ήδη την 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. Μπορείτε να το δοκιμάσετε στην παραπάνω συνάρτηση αρκεί να συμπεριλάβουμε την help(printMax) στο πρόγραμμά μας. Θυμηθείτε να πατήσετε το q για κλείσετε τη help.

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.

7.11. Σχόλια (Annotations)

Οι συναρτήσεις έχουν ακόμα ένα προχωρημένο χαρακτηριστικό που ονομάζεται σχόλια, τα οποία είναι ένας γρήγορος τρόπος να επισυνάψουμε επιπρόσθετες πληροφορίες για κάθε μια από τις παραμέτρους καθώς και την τιμή επιστροφής. Η Python δεν διερμηνεύει απο μόνη της αυτά τα σχόλια με οποιοδήποτε τρόπο (αυτή η λειτουργία έχει αφεθεί στις βιβλιοθήκες τρίτων (third-party libraries), να τα διερμηνεύουν με όποιο τρόπο θέλουν), γι' αυτό θα παραλείψουμε αυτό το χαρακτηριστικό από τη συζήτησή μας. Αν παρόλα αυτά ενδιαφέρεστε να μάθετε για τα σχόλια, μπορείτε να δείτε τις σχετικές αναφορές στο Python Enhancement Proposal No. 3107 [1].

Σύνοψη

Έχουμε ήδη δει πολλές πλευρές των συναρτήσεων αλλα δεν έχουμε καλύψει όλες τις πλευρές τους. Πάντως έχουμε ήδη καλύψει τις περισσότερες από αυτές που θα χρησιμοποιήσουμε με την Python σε καθημερινή βάση. Στο επόμενο κεφάλαιο θα δούμε πως θα χρησιμοποιούμε και πώς θα δημιουργούμε αρθρώματα (modules) Python.

8. Αρθρώματα

Εισαγωγή

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 [1] 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 κάνετε χρήση της λειτουργικότητάς του. Έτσι μπορείτε να χρησιμοποιήσετε επίσης την πρότυπη βιβλιοθήκη της Python. Αρχικά θα δείτε πως να χρησιμοποιείτε τα αρθρώματα της πρότυπης βιβλιοθήκης.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: using_sys.py

import sys

print('The command line arguments are:') for i in
sys.argv:
    print(i)

print('\n\nThe PYTHONPATH is', sys.path, '\n')
```

Έξοδος:

```
$ python using_sys.py we are arguments The
command line arguments are: using_sys.py

we
```

```
are
arguments


The PYTHONPATH is ['', 'C:\\Windows\\system32\\python30.zip',
'C:\\Python30\\DLLs', 'C:\\Python30\\lib', 'C:\\Python30\\lib\\plat-win',
'C:\\Python30', 'C:\\Python30\\lib\\site-packages']
```

Πώς λειτουργεί:

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 (οι λίστες

εξηγούνται σε επόμενο κεφάλαιο. 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) για να γράψετε και να τρέξετε αυτά τα προγράμματα, ψάξε 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.

8.1. 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 (θυμηθείτε το → εισαγωγικό τμήμα του βιβλίου για το πώς λειτουργεί η Python). 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).

Σημείωση

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.

8.2. 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.

8.3. 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.

Παράδειγμα:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# Filename: using_name.py if
__name__ == '__main__':
    print('Αυτό το πρόγραμμα τρέχει από μόνο του') else:
    print('Έχω εισαχθεί από ένα άλλο άρθρωμα')
```

Έξοδος:

```
$ python using_name.py
Αυτό το πρόγραμμα τρέχει από μόνο του

$ python
>>> import using_name
Έχω εισαχθεί από ένα άλλο άρθρωμα
>>>
```

Πώς λειτουργεί:

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.

Για να φτιάχνετε τα δικά σας αρθρώματα

Η δημιουργία των δικών σας αρθρωμάτων είναι εύκολη, το έχετε κάνει ήδη, κι αυτό διότι κάθε πρόγραμμα στη Python είναι επίσης κι ένα άρθρωμα. Για το μόνο που πρέπει να είστε σίγουροι είναι να έχει μια επέκταση .py. Το ακόλουθο παράδειγμα θα το κάνει πιο ξεκάθαρο.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: mymodule.py

def sayhi():
    print('Hi, this is mymodule speaking.')

__version__ = '0.1'

# End of mymodule.py
```

Το ανωτέρω είναι ένα δείγμα *'αρθρώματος*. Όπως μπορείτε να δείτε δεν υπάρχει καμμία ιδιαιτερότητα συγκρινόμενο με ένα συνηθισμένο πρόγραμμα σε Python. Κατόπιν θα δούμε πώς να χρησιμοποιούμε αυτό το άρθρωμα στα δικά μας προγράμματα 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.
Παράδειγμα:

```
#!/usr/bin/python
# Filename: mymodule_demo.py

import mymodule

mymodule.sayhi()
print ('Version', mymodule.__version__)
```

Έξοδος:

```
$ python mymodule_demo.py
Hi, this is mymodule speaking. Version
0.1
```

Παρατηρήστε 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 μαθαίνετε καινούργιους τρόπους για να κάνετε πράγματα.

Να μια εκδοχή όπου γίνεται χρήση της σύνταξης from..import:

```
#!/usr/bin/python
# Filename: mymodule_demo2.py

from mymodule import sayhi, __version__

sayhi()
print('Version', __version__)
```

Το αποτέλεσμα του mymodule_demo2.py είναι το ίδιο με το αποτέλεσμα του mymodule_demo.py.

Παρατηρήστε ότι εάν ήταν ήδη δηλωμένο το όνομα της έκδοσης __version__ στο άρθρωμα το οποίο εισάγει το mymodule, 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.

Το Zen της Python

Μια από τις αρχές καθοδήγησης της Python είναι ότι " Οι ρητές εντολές είναι καλύτερες από αυτές που εξυπακούονται" (Explicit is better than Implicit). Τρέξτε import this για να μάθετε περισσότερα και δείτε αυτή τη συζήτηση [2] που παραθέτει παραδείγματα για κάθε μια από τις αρχές της Python.

8.4. 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.

Παράδειγμα:

```
$ python

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

>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__', '__name__', '__package__',
'__s

tderr__', '__stdin__', '__stdout__', '_clear_type_cache', '_compact_freelists',
```

```
'_current_frames', '_getframe', 'api_version', 'argv',
'builtin_module_names', '
byteorder', 'call_tracing', 'callstats', 'copyright', 'displayhook', 'dllhandle'

, 'dont_write_bytecode', 'exc_info', 'excepthook', 'exec_prefix', 'executable',

'exit', 'flags', 'float_info', 'getcheckinterval', 'getdefaultencoding',
'getfil
esystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof',

'gettrace', 'getwindowsversion', 'hexversion', 'intern', 'maxsize', 'maxunicode

', 'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache', 'platfor

m', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setprofile', 'setrecursionlimit

', 'settrace', 'stderr', 'stdin', 'stdout', 'subversion', 'version', 'version_in

fo', 'warnoptions', 'winver']

>>> dir() # δίνει λίστα ιδιοχαρακτηριστικών για το τρέχον άρθρωμα
['__builtins__', '__doc__', '__name__', '__package__', 'sys']

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

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'a', 'sys']

>>> del a # διαγράφει ή αφαιρεί ένα όνομα

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys']

>>>
```

Πώς λειτουργεί:

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.

Ένα σχόλιο στην del: αυτή η εντολή 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.

Σημειώστε επίσης ότι 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.

8.5. Πακέτα (Packages)

Μέχρι τώρα, πρέπει να έχετε αρχίσει να παρατηρείτε την ιεραρχία με την οποία οργανώνονται τα προγράμματά σας. 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 πούμε ότι θέλετε να δημιουργήσετε ένα πακέτο που ονομάζεται 'world' με υποπακέτα που ονομάζονται 'asia', 'africa', κ.τ.λ. και αυτά τα υποπακέτα με τη σειρά τους περιέχουν αρθρώματα όπως 'india', 'madagascar', κ.τ.λ.

Ένα παράδειγμα για το πώς θα δομούσατε τους φακέλους:

```
- <some folder present in the sys.path>/
    - world/
        - __init__.py
        - asia/
            - __init__.py
            - india/
                - __init__.py
                - foo.py
        - africa/
            - __init__.py
            - madagascar/
                - __init__.py
                - bar.py
```

Τα πακέτα είναι μια ευκολία στην ιεραρχική οργάνωση των αρθρωμάτων. Θα δείτε πολλά τέτοια παραδείγματα στην → πρότυπη βιβλιοθήκη.

Σύνοψη

Όπως ακριβώς και οι συναρτήσεις είναι επαναχρησιμοποιούμενα μέρη προγραμμάτων, τα αρθρώματα είναι επαναχρησιμοποιούμενα προγράμματα. Τα πακέτα είναι μια άλλη ιεραρχία για να οργανώνετε αρθρώματα. Η πρότυπη βιβλιοθήκη που συνοδεύει την Python είναι ένα παράδειγμα τέτοιων πακέτων και αρθρωμάτων.

Έχουμε δει πώς να χρησιμοποιούμε αυτά τα αρθρώματα και πώς να δημιουργούμε δικά μας αρθρώματα.

Κατόπιν θα μάθουμε μερικές ενδιαφέρουσες έννοιες που ονομάζονται δομές δεδομένων.

9. Δομές δεδομένων

Εισαγωγή

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, οι λίστες, οι πλειάδες, τα λεξικά και τα σύνολα. Θα δούμε πώς να τα χρησιμοποιούμε και πώς μας κάνουν τη ζωή ευκολότερη.

9.1. 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.

Γρήγορη εισαγωγή στα αντικείμενα (objects) και τις κλάσεις (classes)

Αν και έχουμε αναβάλει μέχρι τώρα τη συζήτηση για τα αντικείμενα και τις κλάσεις, μια μικρή επεξήγηση απαιτείται ακριβώς τώρα για να καταλάβετε καλύτερα τις λίστες. Θα εξερευνήσουμε αυτό το θέμα αργότερα σε → δικό του κεφάλαιο.

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.

Παράδειγμα:

```
#!/usr/bin/python
#  -*- coding: utf-8 -*-
#  Filename: using_list.py

#  Αυτή είναι η λίστα αγορών μου
shoplist = ['μήλο', 'μάνγκο', 'καρότο', 'μπανάνα']

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


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


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


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


print('Το πρώτο πράγμα που θ\' αγοράσω είναι', shoplist[0]) olditem =
shoplist[0]

del shoplist[0] print('Αγόρασα το',
olditem)
print('Η λίστα αγορών μου τώρα είναι', shoplist)
```

Έξοδος:

```
$ python using_list.py
Πρέπει ν' αγοράσω 4 πράγματα.

Τα πράγματα αυτά είναι: μήλο μάνγκο καρότο μπανάνα Πρέπει
επίσης ν' αγοράσω ρύζι.

Η  λίστα αγορών μου τώρα είναι: ['μήλο', 'μάνγκο', 'καρότο', 'μπανάνα', 'ρύζι']

Θα ταξινομήσω τη λίστα μου τώρα

Η  ταξινομημένη λίστα μου είναι ['καρότο', 'μάνγκο', 'μήλο', 'μπανάνα', 'ρύζι']

Το πρώτο πράγμα που θ' αγοράσω είναι καρότο Αγόρασα
το καρότο

Η  λίστα αγορών μου τώρα είναι ['μάνγκο', 'μήλο', 'μπανάνα', 'ρύζι']
```

Πώς δουλεύει:

Η μεταβλητή shoplist είναι μια λίστα αγορών για κάποιον που πηγαίνει στην αγορά. Στη shoplist 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. Η ιδιαιτερότητα των ακολουθιών θα συζητηθεί σε επόμενη ενότητα.

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, όπως ήδη συζητήσαμε. 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 από τη λαϊκή, 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.

9.2. Πλειάδα

Οι πλειάδες χρησιμοποιούνται για να συγκρατήσουν μαζί πολλαπλά αντικείμενα. 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 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.

Παράδειγμα:

```
#!/usr/bin/python
#  -*- coding: utf-8 -*-
#  Filename: using_tuple.py


zoo = ('πύθωνας', 'ελέφαντας', 'πιγκουίνος') # θυμηθείτε οι παρενθέσεις είναι προαιρετικές
print('Ο αριθμός των ζώων στο ζωολογικό κήπο είναι', len(zoo))

new_zoo = ('μαϊμού', 'καμήλα', zoo)
```

```
print('Ο αριθμός των κλουβιών στο νέο ζωολογικό κήπο είναι', len(new_zoo))

print('Όλα τα ζώα στο νέο ζωολογικό κήπο είναι', new_zoo) print('Όλα τα ζώα που
έφεραν από τον παλιό ζωολογικό κήπο είναι', new_zoo[2])

print('Το τελευταίο ζώο που έφεραν από τον παλιό ζωολογικό κήπο είναι', new_zoo[2][2])

print('Ο αριθμός των ζώων που είναι στο νέο ζωολογικό κήπο', len(new_zoo)-
1+len(new_zoo[2]))
```

Έξοδος:

```
$ python using_tuple.py
Ο αριθμός των ζώων στο ζωολογικό κήπο είναι 3
Ο αριθμός των κλουβιών στο νέο ζωολογικό κήπο είναι 3

Όλα τα ζώα στο νέο ζωολογικό κήπο είναι ('μαϊμού', 'καμήλα', ('πύθωνας',
'ελέφαντας', 'πιγκουίνος'))

Όλα τα ζώα που έφεραν από τον παλιό ζωολογικό κήπο είναι ('πύθωνας', 'ελέφαντας',
'πιγκουίνος')

Το τελευταίο ζώο που έφεραν από τον παλιό ζωολογικό κήπο είναι πιγκουίνος
Ο αριθμός των ζώων που είναι στο νέο ζωολογικό κήπο 5
```

Πώς δουλεύει:

Η μεταβλητή zoo 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.

Τώρα μετακινούμε αυτά τα ζώα σε ένα νέο ζωολογικό κήπο επειδή ο παλιός κλείνει. Συνεπώς, η πλειάδα the new_zoo περιέχει κάποια ζώα που βρίσκονται ήδη εκεί, μαζί με τα ζώα που έφεραν από τον παλιό ζωολογικό κήπο. Πίσω στην πραγματικότητα, σημειώστε 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_zoo καθορίζοντας new_zoo[2] και παίρνουμε το τρίτο στοιχείο μέσα στο τρίτο στοιχείο στην πλειάδα new_zoo καθορίζοντας new_zoo[2][2]. Αυτό είναι πολύ απλό άπαξ και έχετε καταλάβει το ιδίωμα.

Παρενθέσεις

Αν και οι παρενθέσεις είναι προαιρετικές, εγώ προτιμώ πάντα να τις έχω για να κάνω 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.

Σημείωση για τους προγραμματιστές της Perl

Μια λίστα μέσα σε μια λίστα δεν χάνει την ταυτότητά της δηλ. οι λίστες δεν ισοπεδώνονται όπως στην Perl. Το ίδιο ισχύει για μια πλειάδα μέσα σε μια πλειάδα, ή για μια πλειάδα μέσα σε μια λίστα, ή για μια λίστα μέσα σε μια πλειάδα, κ.τ.λ. Σε ό,τι αφορά την Python, αυτά είναι μόνο αντικείμενα που αποθηκεύονται χρησιμοποιώντας ένα άλλο αντικείμενο, κι αυτό είναι όλο.

9.3. Λεξικό

Ένα λεξικό είναι σαν ένας τηλεφωνικός κατάλογος όπου μπορείτε να βρείτε τη διεύθυνση ή άλλα στοιχεία επικοινωνίας για ένα άτομο, γνωρίζοντας μόνο το όνομά του/της, δηλαδή συσχετίζουμε **κλειδιά** (ονομασία) με **τιμές** (λεπτομέρειες). Σημειώστε ότι το κλειδί πρέπει να είναι μοναδικό, με τον ίδιο τρόπο που δε θα μπορείτε να βρείτε τα σωστά στοιχεία επικοινωνίας κάποιου αν έχετε δύο άτομα με το ίδιο όνομα.

Σημειώστε ότι μπορείτε να χρησιμοποιήσετε μόνο αμετάβλητα αντικείμενα (όπως συμβολοσειρές) για τα κλειδιά του λεξικού, αλλά μπορείτε να χρησιμοποιήσετε είτε αμετάβλητα είτε μεταβλητά αντικείμενα για τις τιμές του λεξικού. Αυτό ουσιαστικά σημαίνει ότι πρέπει να χρησιμοποιείτε μόνο απλά αντικείμενα για κλειδιά.

Ζευγάρια κλειδιών και τιμών καθορίζονται στο λεξικό χρησιμοποιώντας το συμβολισμό d = {key1 : value1, key2 : value2 }. Παρατηρήστε ότι τα ζευγάρια κλειδί-τιμή διαχωρίζονται με διπλή τελεία και τα ζευγάρια διαχωρίζονται μεταξύ τους με κόμματα και όλα αυτά περικλείονται σε ένα ζευγάρι άγκιστρων.

Θυμηθείτε ότι τα ζευγάρια κλειδί-τιμή σε ένα λεξικό δεν ταξινομούνται με κανένα τρόπο. Αν θέλετε μια ειδική σειρά ταξινόμησης, τότε πρέπει να τα ταξινομήσετε από μόνοι σας πριν τα χρησιμοποιήσετε.
Τα λεξικά που θα χρησιμοποιείτε είναι υποστάσεις/αντικείμενα της κλάσης dict.
Παράδειγμα:

```
#!/usr/bin/python
#  Filename: using_dict.py

#  'ab' is short for 'a'ddress'b'ook

ab = {  'Swaroop'   : 'swaroop@swaroopch.com',
            'Larry'          : 'larry@wall.org',
            'Matsumoto' : 'matz@ruby-lang.org',
            'Spammer'     : 'spammer@hotmail.com'
        }

print("Swaroop's address is", ab['Swaroop'])

# Deleting a key-value pair del
ab['Spammer']

print('\nThere are {0} contacts in the address-book\n'.format(len(ab)))

for name, address in ab.items():
    print('Contact {0} at {1}'.format(name, address))

# Adding a key-value pair ab['Guido'] =
'guido@python.org'
```

```
if 'Guido' in ab:
    print("\nGuido's address is", ab['Guido'])
```

Έξοδος:

```
$ python using_dict.py
Swaroop's address is swaroop@swaroopch.com

There are 3 contacts in the address-book

Contact Swaroop at swaroop@swaroopch.com
Contact Matsumoto at matz@ruby-lang.org
Contact Larry at larry@wall.org

Guido's address is guido@python.org
```

Πώς δουλεύει:

Δημιουργούμε το λεξικό ab χρησιμοποιώντας το συμβολισμό που ήδη συζητήσαμε. Τότε εισάγουμε ζευγάρια κλειδί-τιμή καθορίζοντας το κλειδί, χρησιμοποιώντας τον τελεστή ευρετηρίασης (indexing operator) όπως συζητήθηκε στο απόσπασμα των λιστών και πλειάδων. Παρατηρήστε την απλή σύνταξη.

Μπορούμε να διαγράψουμε ζευγάρια κλειδί-τιμή χρησιμοποιώντας τον παλιό μας φίλο, την εντολή del. Εμείς απλά καθορίζουμε το λεξικό και τον τελεστή ευρετηρίασης για το κλειδί που θα αφαιρεθεί και τα περνάμε στην εντολή del. Δεν είναι αναγκαίο να γνωρίζετε την τιμή που αντιστοιχεί στο κλειδί για αυτή τη λειτουργία.

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 Guido 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 τη help(dict).

Ορίσματα με λέξεις κλειδιά και λεξικά

Σε μια διαφορετική νότα, εάν έχετε χρησιμοποιήσει ορίσματα με λέξεις κλειδιά στις συναρτήσεις σας, τότε έχετε ήδη χρησιμοποιήσει λεξικά! Σκεφτείτε μόνο αυτό, το ζευγάρι κλειδί-τιμή καθορίζεται από εσάς στη λίστα παραμέτρων του ορισμού της συνάρτησης και όταν ζητάτε πρόσβαση σε μεταβλητές μέσα στη συνάρτησή σας, αυτό είναι απλά πρόσβαση σε ένα κλειδι ενός λεξικού (που ονομάζεται *συμβολοπίνακας* στην ορολογία του σχεδίασης μεταγλωττιστών).

9.4. 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.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: seq.py

shoplist = ['apple', 'mango', 'carrot', 'banana'] name = 'swaroop'


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



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


# 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[:])
```

Έξοδος:

```
$ python seq.py Item 0
is apple Item 1 is
mango Item 2 is carrot
Item 3 is banana Item
-1 is banana Item -2 is
carrot Character 0 is s
```

```
Item 1 to 3 is ['mango', 'carrot'] Item 2 to end is
['carrot', 'banana'] Item 1 to -1 is ['mango', 'carrot']

Item start to end is ['apple', 'mango', 'carrot', 'banana'] characters 1 to 3 is wa

characters 2 to end is aroop characters 1 to -
1 is waroo characters start to end is swaroop
```

Πώς λειτουργεί:

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 shoplist[0] κάνει μετάκληση του πρώτου στοιχείου και η shoplist[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 shoplist[-1] αναφέρεται στο τελευταίο στοιχείο στην ακολουθία και η shoplist[-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.

Σημειώστε ότι 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 shoplist[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 shoplist[:] 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 shoplist[:-1] θα επιστρέψει ένα τμήμα της ακολουθίας η οποία παραλείπει το τελευταίο στοιχείο της ακολουθίας, αλλά περιέχει κάθε άλλο.

Μπορείτε επίσης να δώσετε ένα τρίτο όρισμα για το τμήμα, το οποίο είναι το *βήμα* (step) για τον τεμαχισμό (από προεπιλογή το μέγεθος βήματος είναι 1):

```
>>> shoplist = ['apple', 'mango', 'carrot', 'banana']
>>> shoplist[::1]
['apple', 'mango', 'carrot', 'banana']
>>> shoplist[::2] ['apple',
'carrot']
>>> shoplist[::3] ['apple',
'banana']
>>> shoplist[::-1]
```

```
['banana', 'carrot', 'mango', 'apple']
```

Παρατηρήστε ότι όταν το βήμα είναι 2, παίρνουμε τα στοιχεία με θέση 0, 2, ... Όταν το μέγεθος βήματος είναι 3, παίρνουμε τα στοιχεία με θέση 0, 3, κ.τ.λ.

Δοκιμάστε διάφορους συνδυασμούς από τέτοιους προσδιορισμούς τμημάτων, χρησιμοποιώντας το διερμηνευτή Python αλληλεπιδραστικά, δηλ. την προτροπή (prompt) έτσι ώστε να μπορείτε να δείτε τα αποτελέσματα αμέσως. Το σπουδαίο με τις ακολουθίες είναι ότι μπορείτε να έχετε πρόσβαση σε πλειάδες, λίστες και συμβολοσειρές όλες με τον ίδιο τρόπο.

9.5. Σύνολο (Set)

Τα σύνολα είναι *μη ταξινομημένες* συλλογές απλών αντικειμένων. 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.

```
>>> bri = set(['brazil', 'russia', 'india'])
>>> 'india' in bri
True
>>> 'usa'    in    bri
False
>>> bric = bri.copy()
>>> bric.add('china')
>>> bric.issuperset(bri)
True
>>> bri.remove('russia')
>>> bri & bric # OR bri.intersection(bric)
{'brazil', 'india'}
```

Πώς δουλεύει:

Το παράδειγμα εξηγείται από μόνο του, διότι περιλαμβάνει βασική θεωρία μαθηματικών συνόλων που διδάσκεται στο σχολείο.

9.6. 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.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: reference.py

print('Simple Assignment')
shoplist = ['apple', 'mango', 'carrot', 'banana']
mylist = shoplist # mylist is just another name pointing to the same
```

```
object!

del shoplist[0] # I purchased the first item, so I remove it from the list


print('shoplist is', shoplist) print('mylist
is', mylist)
#  notice that both shoplist and mylist both print the same list without
#  the 'apple' confirming that they point to the same object

print('Copy by making a full slice')
mylist = shoplist[:] # make a copy by doing a full slice del mylist[0] # remove
first item

print('shoplist is', shoplist) print('mylist
is', mylist)
# notice that now the two lists are different
```

Έξοδος:

```
$ python reference.py
Simple Assignment
shoplist is ['mango', 'carrot', 'banana'] mylist is ['mango',
'carrot', 'banana'] Copy by making a full slice


shoplist is ['mango', 'carrot', 'banana'] mylist is ['carrot',
'banana']
```

Πώς δουλεύει:
Το μεγαλύτερο μέρος της εξήγησης είναι διαθέσιμο στα σχόλια.

Θυμηθείτε ότι εάν θέλετε να φτιάξετε ένα αντίγραφο μιας λίστας, ή τέτοιου είδους ακολουθίες, ή σύμπλοκα αντικείμενα (όχι απλά *αντικείμενα* όπως ακέραιους αριθμούς), τότε πρέπει να χρησιμοποιήσετε τη λειτουργία τεμαχισμού για να φτιάξετε αντίγραφο. Εάν εκχωρήσετε 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 σας, εάν δεν είστε προσεκτικοί.

Σημείωση για τους προγραμματιστές της Perl

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

9.7. Περισσότερα για τις συμβολοσειρές

Έχουμε ήδη συζητήσει νωρίτερα για τις συμβολοσειρές (strings) προηγουμένως. Τι περισσότερο μπορούμε να μάθουμε; Λοιπόν, γνωρίζατε ότι 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, κοιτάξτε τη help(str).

Παράδειγμα:

```
#!/usr/bin/python
# Filename: str_methods.py

name = 'Swaroop' # This is a string object

if name.startswith('Swa'):
    print('Yes, the string starts with "Swa"')

if 'a' in name:
    print('Yes, it contains the string "a"')

if name.find('war') != -1:
    print('Yes, it contains the string "war"')

delimiter = '_*_'
mylist = ['Brazil', 'Russia', 'India', 'China']
print(delimiter.join(mylist))
```

Έξοδος:

```
$ python str_methods.py
Yes, the string starts with "Swa" Yes, it
contains the string "a" Yes, it contains the
string "war"
Brazil_*_Russia_*_India_*_China
```

Πώς δουλεύει:

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 ωραία 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.

Σύνοψη

Έχουμε διερευνήσει τις διάφορες ενσωματωμένες δομές δεδομένων της Python με λεπτομέρεια. Αυτές οι δομές δεδομένων θα είναι απαραίτητες για τη συγγραφή προγραμμάτων σε κάποιο υπολογίσιμο μέγεθος. Τώρα που έχουμε δει πολλά από τα βασικά της Python, θα δούμε πώς να σχεδιάζουμε και να γράφουμε ένα πραγματικό πρόγραμμα Python.

10. Επίλυση προβλημάτων

Έχουμε εξερευνήσει διάφορες λειτουργίες της Python και τώρα θα δούμε πως να τις ταιριάξουμε όλες μαζί με το σχεδιασμό και τη συγγραφή ενός προγράμματος που *κάνει* κάτι χρήσιμο. Ο σκοπός είναι να μάθουμε πώς γράφουμε το δικό μας πρόγραμμα σε Python.

10.1. Το πρόβλημα

Το πρόβλημα είναι *"Θέλω ένα πρόγραμμα που δημιουργεί αντίγραφα ασφαλείας όλων των σημαντικών μου αρχείων"*.

Αν και είναι ένα απλό πρόβλημα, δεν έχουμε αρκετές πληροφορίες για να ξεκινήσουμε με την επίλυσή του. Απαιτείται καλύτερη **ανάλυση**. Για παράδειγμα, πώς θα προσδιορίσουμε *ποια* αρχεία θέλουμε για τη δημιουργία αντιγράφων ασφαλείας; *Πώς* θα αποθηκεύονται; *Πού* θα αποθηκεύονται;

Αφού αναλύσουμε σωστά το πρόβλημα, **σχεδιάζουμε** το πρόγραμμά μας. Δημιουργούμε μια λίστα με τη δομή και λειτουργία του προγράμματος. Στην περίπτωσή μας, έχω δημιουργήσει την ακόλουθη λίστα για το πώς θέλω *εγώ* να δουλεύει. Στο δικό σας σχεδιασμό, μπορεί να μην καταλήξετε στην ίδια ανάλυση, μιας και ο καθένας έχει το δικό του, ιδιαίτερο τρόπο που ενεργεί, και αυτό είναι απολύτως αποδεκτό.

1. Τα προς δημιουργία αντιγράφων ασφαλείας αρχεία και φάκελοι καθορίζονται σε μια λίστα.
2. Τα αντίγραφα ασφαλείας πρέπει να αποθηκευτούν σε ένα κεντρικό φάκελο.
3. Τα αντίγραφα ασφαλείας αποθηκεύονται σε ένα συμπιεσμένο αρχείο.
4. Το όνομα του συμπιεσμένου αρχείου είναι η τρέχουσα ημερομηνία και ώρα.
5. Χρησιμοποιούμε τη γνωστή εντολή zip που είναι διαθέσιμη σε κάθε σύγχρονη διανομή Linux/Unix. Οι χρήστες Windows μπορούν να την εγκαταστήσουν [1] από τη σελίδα του έργου [2] και να προσθέσουν τη διαδρομή C:\Program Files\GnuWin32\bin στο σύστημά τους κατάλληλα ως μεταβλητή περιβάλλοντος, όπως έχουμε ήδη δει για την αναγνώριση του διερμηνευτή Python. Σημειώστε ότι μπορείτε να χρησιμοποιήσετε κάθε εφαρμογή αρχειοθέτησης, αρκεί να διαθέτει γραμμή εντολών και να μπορεί να δεχτεί εντολές από το πρόγραμμά μας.

10.2. Η λύση

Αφού ο σχεδιασμός του προγράμματος σταθεροποιήθηκε, μπορούμε να γράψουμε τον κώδικα που αποτελεί την **υλοποίηση** της λύσης μας.

```
#!/usr/bin/python
# Filename: backup_ver1.py

import os
import time
```

```
# 1. Τα προς δημιουργία αντιγράφων ασφαλείας αρχεία και φάκελοι
καθορίζονται σε μια λίστα
source = ['"C:\\My Documents"', 'C:\\Code']
# Προσέξτε τη χρήση διπλών εισαγωγικών εντός της συμβολοσειράς για
ονόματα αρχείων με κενούς χαρακτήρες

# 2. Τα αντίγραφα ασφαλείας πρέπει να αποθηκευτούν σε ένα κεντρικό
φάκελο
target_dir = 'E:\\Backup' # Θυμηθείτε να αλλάξετε τον φάκελο προορισμού
  σε αυτόν που θα χρησιμοποιήσετε

# 3. Τα αντίγραφα ασφαλείας αποθηκεύονται σε ένα συμπιεσμένο αρχείο
# 4. Το όνομα του συμπιεσμένου αρχείου είναι η τρέχουσα ημερομηνία και
ώρα
target = target_dir + os.sep + time.strftime('%Y%m%d%H%M%S') + '.zip'

# 5. Εκτελούμε την εντολή zip για την εισαγωγή των αρχείων στο
συμπιεσμένο αρχείο
zip_command = "zip -qr {0} {1}".format(target, ' '.join(source))

# Εκτέλεση του προγράμματος
if os.system(zip_command) == 0: print('Successful
      backup to', target)
else:
      print('Backup FAILED')
```

Έξοδος:

```
$ python backup_ver1.py
Successful backup to E:\Backup\20080702185040.zip
```

Τώρα βρισκόμαστε στη **δοκιμαστική** φάση και ελέγχουμε αν το πρόγραμμά μας λειτουργεί σωστά. Αν δε λειτουργεί όπως θα θέλαμε, οφείλουμε να **αποσφαλματώσουμε** (debug) το πρόγραμμα, δηλαδή να διορθώσουμε τα *σφάλματα* (bugs) του προγράμματος.

Αν το παραπάνω πρόγραμμα δε λειτουργεί, προσθέστε μια εντολή print(zip_command) πριν την κλήση του αρθρώματος os.system και εκτελέστε το. Τώρα, αντιγράψτε την εκτέλεση της zip_command και επικολλήστε την στο τερματικό για να δείτε αν εκτελείται μόνη της σωστά. Αν αποτύχει, δείτε το εγχειρίδιο της zip εντολής για πιθανά σφάλματα. Αν επιτύχει, ελέγξτε τον πηγαίο κώδικα αν ταιριάζει απόλυτα με τον κώδικα του προγράμματος παραπάνω.

Πώς λειτουργεί:

Θα παρατηρήσετε πώς έχουμε μετατρέψει τη *σχεδίαση* σε *κώδικα* σταδιακά.

Πρώτα χρησιμοποιούμε τα αρθρώματα os και time με την εισαγωγή τους. Στη συνέχεια, ορίζουμε τα αρχεία και τους φακέλους προς αρχειοθέτηση στη λίστα source. Ο φάκελος προορισμού, όπου αποθηκεύονται τα αντίγραφα ασφαλείας, ορίζεται στη μεταβλητή target_dir. Το όνομα του συμπιεσμένου αρχείου που θα δημιουργηθεί είναι η τρέχουσα ημερομηνία και ώρα και παράγεται αυτόματα από τη συνάρτηση time.strftime(). Το αρχείο θα έχει επίσης την κατάληξη .zip και θα αποθηκευτεί στο φάκελο target_dir.

Παρατηρήστε τη χρήση της μεταβλητής os.sep - ορίζει το διαχωριστή καταλόγου δηλαδή θα είναι '/' για Linux και Unix, '\\' για Windows και ':' για Mac OS. Η χρήση της os.sep μεταβλητής στη θέση αυτών των χαρακτήρων επιτρέπει τη φορητότητα του προγράμματός μας μεταξύ όλων αυτών των λειτουργικών συστημάτων.

Η συνάρτηση time.strftime() δέχεται προδιαγραφές όπως αυτές που έχουμε χρησιμοποιήσει στο παραπάνω πρόγραμμα. Η προδιαγραφή %Y αντικαθίσταται από το έτος μαζί με τον αιώνα. Η προδιαγραφή %m θα αντικατασταθεί από το μήνα ως δεκαδικός αριθμός μεταξύ 01 και 12 και ούτω καθ' εξής. Η πλήρης λίστα αυτών των προδιαγραφών βρίσκεται στο εγχειρίδιο αναφοράς Python [3].
Δημιουργούμε το όνομα του συμπιεσμένου αρχείου προορισμού με τη χρήση του τελεστή πρόσθεσης (+), ο οποίος *συνενώνει* τις συμβολοσειρές, δηλαδή ενώνει τις δύο συμβολοσειρές και επιστρέφει μια νέα. Στη συνέχεια, δημιουργούμε τη συμβολοσειρά zip_command και η οποία περιέχει την εντολή που θέλουμε να εκτελεστεί. Μπορείτε να ελέγξετε αν λειτουργεί η εντολή εκτελώντας την στο κέλυφος (Τερματικό Linux ή γραμμή εντολών DOS).

Η χρήση της εντολής zip δέχεται κάποιες επιλογές και παραμέτρους. Η επιλογή -q δηλώνει ότι πρέπει να εκτελεστεί αθόρυβα (**q**uietly). Η επιλογή -r προσδιορίζει ότι η εντολή πρέπει να εκτελεστεί αναδρομικά (**r**ecursively) για καταλόγους, δηλαδή να συμπεριλάβει όλους τους υποφακέλους και τα αρχεία. Οι δύο επιλογές συνδυάζονται και προσδιορίζονται με τη συντόμευση -qr. Μετά τις επιλογές αυτές ακολουθεί το όνομα του συμπιεσμένου αρχείου που θα δημιουργηθεί και έπεται η λίστα αρχείων και φακέλων προς αρχειοθέτηση. Μετατρέπουμε τη λίστα source σε μια συμβολοσειρά με τη χρήση της μεθόδου join των συμβολοσειρών. Έχουμε ήδη αναφερθεί στον τρόπο χρήσης της.

Τέλος, *εκτελούμε* την εντολή με τη χρήση της συνάρτησης os.system και η οποία τρέχει την εντολή σαν να είχε εκτελεστεί από το ίδιο το *σύστημα*, δηλαδή επιστρέφει στο κέλυφος 0 αν η εκτέλεση ήταν επιτυχής, ειδάλλως επιστρέφει κάποιον αριθμό σφάλματος.

Ανάλογα με το αποτέλεσμα της εντολής, εκτυπώνουμε στην οθόνη το αντίστοιχο μήνυμα επιτυχίας ή αποτυχίας της δημιουργίας του αντιγράφου ασφαλείας.

Αυτό ήταν, δημιουργήσαμε ένα σενάριο εντολών για να παίρνουμε αντίγραφα ασφαλείας των σημαντικών μας αρχείων!

Σημείωση για χρήστες Windows

> Αντί για διπλούς αριστερούς πλάγιους καθέτους ως ακολουθίες διαφυγής, μπορείτε επίσης να χρησιμοποιήσετε ανεπεξέργαστες συμβολοσειρές. Για παράδειγμα, χρησιμοποιήστε τη σύνταξη 'C:\\Documents' ή r'C:\Documents'. Ωστόσο, **μη** χρησιμοποιήσετε τη σύνταξη 'C:\Documents', διότι θα καταλήξετε με μια άγνωστη ακολουθία διαφυγής \D.

Τώρα που το σενάριο είναι πλήρως λειτουργικό, μπορούμε να δημιουργήσουμε όποτε θέλουμε ένα αντίγραφο ασφαλείας των αρχείων μας. Για χρήστες Linux/Unix συστήνεται η μέθοδος εκτελέσιμων προγραμμάτων, όπως συζητήθηκε πριν, ώστε να εκτελεστεί το σενάριο δημιουργίας αντιγράφων ασφαλείας οποτεδήποτε, οπουδήποτε. Αποκαλείται στάδιο **λειτουργίας** της **ανάπτυξης** λογισμικού.

Το παραπάνω πρόγραμμα λειτουργεί κανονικά, αλλά (συνήθως) τα πρώτα στάδια δεν λειτουργούν όπως ακριβώς θα θέλαμε. Για παράδειγμα, ίσως παρουσιαστούν προβλήματα αν δεν σχεδιάσατε σωστά το πρόγραμμα ή έχετε κάποιο συντακτικό λάθος στον πηγαίο κώδικα. Θα χρειαστεί να επιστρέψετε πίσω στο στάδιο σχεδιασμού ή αποσφαλμάτωσης.

10.3. Δεύτερη έκδοση

Η πρώτη έκδοση του προγράμματός μας λειτουργεί. Ωστόσο, μπορούμε να κάνουμε κάποιες αλλαγές σε αυτό, ώστε να βελτιωθεί η χρήση του σε καθημερινή βάση. Αυτό ονομάζεται στάδιο **συντήρησης** του λογισμικού.

Μια από τις βελτιώσεις που θεώρησα χρήσιμη είναι ο καλύτερος μηχανισμός ονομασίας των αρχείων -με τη χρήση της *ώρας* για το όνομα του αρχείου εντός ενός φακέλου και με την τρέχουσα *ημερομηνία* ως όνομα φακέλου εντός του κεντρικού φακέλου αντιγράφων ασφαλείας. Το πρώτο πλεονέκτημα είναι ότι τα αντίγραφα ασφαλείας αποθηκεύονται σε ιεραρχική δομή και επομένως είναι πολύ πιο εύκολο να τα διαχειριστείτε. Το δεύτερο πλεονέκτημα είναι ότι τα ονόματα αρχείων είναι αισθητά μικρότερα. Το τρίτο πλεονέκτημα είναι ότι οι ξεχωριστοί φάκελοι θα σας βοηθήσουν να ελέγξετε εάν έχετε δημιουργήσει ένα αντίγραφο ασφαλείας για κάθε ημέρα, δεδομένου ότι ο φάκελος θα δημιουργηθεί μόνο αν υπάρχει ήδη ένα αντίγραφο ασφαλείας για την ημέρα εκείνη.

```
#!/usr/bin/python
# Filename: backup_ver2.py

import os
import time

# 1. Τα προς δημιουργία αντιγράφων ασφάλειας αρχεία και φάκελοι
καθορίζονται σε μια λίστα
source = ['"C:\\My Documents"', 'C:\\Code']
# Προσέξτε τη χρήση διπλών εισαγωγικών εντός της συμβολοσειράς για
ονόματα αρχείων με κενούς χαρακτήρες

# 2. Τα αντίγραφα ασφαλείας πρέπει να αποθηκευτούν σε ένα κεντρικό
φάκελο
target_dir = 'E:\\Backup' # Θυμηθείτε να αλλάξετε τον φάκελο προορισμού
  σε αυτόν που θα χρησιμοποιήσετε

# 3. Τα αντίγραφα ασφαλείας αποθηκεύονται σε ένα συμπιεσμένο αρχείο
# 4. Η τρέχουσα ημέρα είναι το όνομα του υποφακέλου στον κεντρικό
φάκελο
today = target_dir + os.sep + time.strftime('%Y%m%d')
# Το όνομα του συμπιεσμένου αρχείου είναι η τρέχουσα ώρα
now = time.strftime('%H%M%S')


# Δημιουργία του υποφακέλου αν δεν υπάρχει ήδη if not
os.path.exists(today):

    os.mkdir(today) # make directory print('Successfully created
    directory', today)


# Το όνομα του συμπιεσμένου αρχείου target =
today + os.sep + now + '.zip'


# 5. Εκτελούμε την εντολή zip για την εισαγωγή των αρχείων στο
συμπιεσμένο αρχείο
zip_command = "zip -qr {0} {1}".format(target, ' '.join(source))
```

```
# Εκτέλεση του προγράμματος
if os.system(zip_command) == 0: print('Successful
    backup to', target)
else:
    print('Backup FAILED')
```

Έξοδος:

```
$ python backup_ver2.py
Successfully created directory E:\Backup\20080702 Successful backup to
E:\Backup\20080702\202311.zip


$ python backup_ver2.py
Successful backup to E:\Backup\20080702\202325.zip
```

Πώς λειτουργεί:

Το μεγαλύτερο μέρος του προγράμματος παραμένει ίδιο. Οι αλλαγές είναι ότι ελέγχουμε αν υπάρχει φάκελος με όνομα την τρέχουσα ημέρα εντός του κεντρικού φακέλου των αντιγράφων ασφαλείας με τη χρήση της συνάρτησης os.path.exists. Αν δεν υπάρχει, το δημιουργούμε με τη συνάρτηση os.mkdir.

10.4. Τρίτη έκδοση

Η δεύτερη έκδοση λειτουργεί άψογα όταν κάνω πολλά αντίγραφα ασφαλείας, αλλά όταν υπάρχει πληθώρα αντιγράφων, μου είναι δύσκολο να ξεχωρίσω το καθένα από αυτά. Για παράδειγμα, μπορεί να έγιναν κάποιες σημαντικές αλλαγές σε ένα πρόγραμμα ή μια παρουσίαση και θέλω να συσχετίσω τις αλλαγές αυτές με το όνομα του συμπιεσμένου αρχείου. Αυτό μπορεί εύκολα να επιτευχθεί επισυνάπτοντας ένα σχόλιο ορισμένο από το χρήστη στο όνομα του συμπιεσμένου αρχείου.

Σημείωση

Το παρακάτω πρόγραμμα δεν λειτουργεί, οπότε μην ανησυχήσετε, παρακαλώ παρακολουθήστε γιατί υπάρχει ένα μάθημα εδώ.

```
#!/usr/bin/python
# Filename: backup_ver3.py


import os
import time

# 1. Τα προς δημιουργία αντιγράφων ασφάλειας αρχεία και φάκελοι
καθορίζονται σε μια λίστα
source = ['"C:\\My Documents"', 'C:\\Code']
# Προσέξτε τη χρήση διπλών εισαγωγικών εντός της συμβολοσειράς για
ονόματα αρχείων με κενούς χαρακτήρες

# 2. Τα αντίγραφα ασφαλείας πρέπει να αποθηκευτούν σε ένα κεντρικό
φάκελο
target_dir = 'E:\\Backup' # Θυμηθείτε να αλλάξετε τον φάκελο προορισμού
 σε αυτόν που θα χρησιμοποιήσετε
```

```
# 3. Τα αντίγραφα ασφαλείας αποθηκεύονται σε ένα συμπιεσμένο αρχείο
# 4. Η τρέχουσα ημέρα είναι το όνομα του υποφακέλου στον κεντρικό
φάκελο
today = target_dir + os.sep + time.strftime('%Y%m%d')
# The current time is the name of the zip archive
now = time.strftime('%H%M%S')

# Ζητήστε από το χρήστη ένα σχόλιο για την επισύναψή του στο όνομα του
συμπιεσμένου αρχείου
comment = input('Enter a comment --> ')
if len(comment) == 0: # Έλεγχος για την προσθήκη του σχολίου target = today +
    os.sep + now + '.zip'
else:
    target = today + os.sep + now + '_' +
        comment.replace(' ', '_') + '.zip'

# Δημιουργία του υποφακέλου αν δεν υπάρχει ήδη
if not os.path.exists(today): os.mkdir(today) #
    make directory
    print('Successfully created directory', today)

# 5. Εκτελούμε την εντολή zip για την εισαγωγή των αρχείων στο
συμπιεσμένο αρχείο
zip_command = "zip -qr {0} {1}".format(target, ' '.join(source))

# Εκτέλεση του προγράμματος
if os.system(zip_command) == 0: print('Successful
    backup to', target)
else:
    print('Backup FAILED')
```

Έξοδος:

```
$ python backup_ver3.py
  File "backup_ver3.py", line 25
    target = today + os.sep + now + '_' +
                                          ^
SyntaxError: invalid syntax
```

Πώς (δεν) λειτουργεί:

Το πρόγραμμα αυτό δεν λειτουργεί! Η Python αναφέρει ότι υπάρχει συντακτικό σφάλμα, που σημαίνει ότι το σενάριο δεν πληροί την αναμενόμενη δομή της Python. Αν παρατηρήσουμε το σφάλμα που αναφέρεται, θα δούμε και πού εντοπίστηκε. Ξεκινούμε λοιπόν την *αποσφαλμάτωση* του προγράμματος από αυτή τη γραμμή.

Με προσεκτική παρατήρηση, θα δούμε ότι η μονή λογική γραμμή έχει χωριστεί σε δύο φυσικές γραμμές χωρίς να ορίσουμε ότι αυτές οι δύο πάνε μαζί. Βασικά, η Python βρήκε τον τελεστή πρόσθεσης (+) χωρίς κάποιον τελεστέο σ'αυτή τη λογική γραμμή και κατά συνέπεια δεν ξέρει πώς να συνεχίσει. Θυμηθείτε ότι μπορούμε να δηλώσουμε τη συνέχιση της λογικής γραμμής στην επόμενη φυσική γραμμή με τη χρήση μιας δεξιάς πλάγιας καθέτου στο τέλος της φυσικής γραμμής. Έτσι διορθώνουμε το πρόγραμμά μας. Αυτή η διόρθωση, όταν

εντοπίζουμε σφάλματα, αποκαλείται **διόρθωση σφαλμάτων**.

10.5. Τέταρτη έκδοση

```
#!/usr/bin/python
# Filename: backup_ver4.py


import os
import time

# 1. Τα προς δημιουργία αντιγράφων ασφάλειας αρχεία και φάκελοι
καθορίζονται σε μια λίστα
source = ['"C:\\My Documents"', 'C:\\Code']
# Προσέξτε τη χρήση διπλών εισαγωγικών εντός της συμβολοσειράς για
ονόματα αρχείων με κενούς χαρακτήρες

# 2. Τα αντίγραφα ασφαλείας πρέπει να αποθηκευτούν σε ένα κεντρικό
φάκελο
target_dir = 'E:\\Backup' # Θυμηθείτε να αλλάξετε τον φάκελο προορισμού
  σε αυτόν που θα χρησιμοποιήσετε

# 3. Τα αντίγραφα ασφαλείας αποθηκεύονται σε ένα συμπιεσμένο αρχείο
# 4. Η τρέχουσα ημέρα είναι το όνομα του υποφακέλου στον κεντρικό
φάκελο
today = target_dir + os.sep + time.strftime('%Y%m%d')
# The current time is the name of the zip archive
now = time.strftime('%H%M%S')

# Ζητήστε από το χρήστη ένα σχόλιο για την επισύναψή του στο όνομα του
συμπιεσμένου αρχείου
comment = input('Enter a comment --> ')

if len(comment) == 0: # check if a comment was entered target = today +
    os.sep + now + '.zip'

else:

    target = today + os.sep + now + '_' + \
        comment.replace(' ', '_') + '.zip'


# Δημιουργία του υποφακέλου αν δεν υπάρχει ήδη

if not os.path.exists(today): os.mkdir(today) #
    make directory
    print('Successfully created directory', today)

# 5. Εκτελούμε την εντολή zip για την εισαγωγή των αρχείων στο
συμπιεσμένο αρχείο
zip_command = "zip -qr {0} {1}".format(target, ' '.join(source))

# Εκτέλεση του προγράμματος
if os.system(zip_command) == 0:
```

```
    print('Successful backup to', target) else:

    print('Backup FAILED')
```

Έξοδος:

```
$ python backup_ver4.py
Enter a comment --> added new examples
Successful backup to E:\Backup\20080702\202836_added_new_examples.zip

$ python backup_ver4.py
Enter a comment -->
Successful backup to E:\Backup\20080702\202839.zip
```

Πώς λειτουργεί:

Λειτουργεί τώρα! Ας δούμε τις ουσιαστικές βελτιώσεις που κάναμε στην τρίτη έκδοση. Επισυνάπτουμε τα σχόλια χρήστη με τη συνάρτηση input και κατόπιν ελέγχουμε αν ο χρήστης πράγματι πληκτρολόγησε κάτι ελέγχοντας το μέγεθος της πληκτρολόγησης με τη χρήση της len συνάρτησης. Αν ο χρήστης πληκτρολόγησε μόνο enter και τίποτα άλλο (ίσως πρόκειται απλά για ένα αντίγραφο ρουτίνας ή δεν υπήρξαν κάποιες ιδιαίτερες αλλαγές), τότε θα προχωρήσουμε με τον προηγούμενο τρόπο.

Ωστόσο, αν πληκτρολογήσαμε ένα σχόλιο, επισυνάπτεται στο όνομα του συμπιεσμένου αρχείου ακριβώς πριν την κατάληξη .zip. Παρατηρήστε ότι οι κενοί χαρακτήρες αντικαθίστανται με κάτω παύλες (underscores), διότι η διαχείριση ονομάτων αρχείων χωρίς κενούς χαρακτήρες είναι ευκολότερη.

10.6. Περισσότερες βελτιώσεις

Η τέταρτη έκδοση του σεναρίου λειτουργεί ικανοποιητικά για τους περισσότερους χρήστες, αλλά υπάρχουν πάντα περιθώρια βελτίωσης. Για παράδειγμα, μπορείτε να συμπεριλάβετε ένα *λεκτικό* επίπεδο στο πρόγραμμα σας, όπου η παράμετρος -v θα το κάνει πιο ομιλητικό.

Μια άλλη πιθανή βελτίωση θα ήταν να επιτραπεί σε επιπλέον αρχεία και φακέλους η πρόσβασή τους στο σενάριο από τη γραμμή εντολών. Μπορούμε να εξάγουμε τα ονόματα αυτά με τη λίστα sys.argv και να τα προσθέσουμε στη λίστα source με τη χρήση της μεθόδου extend της κλάσης list.

Η σημαντικότερη βελτίωση θα ήταν να μην χρησιμοποιούσαμε το άρθρωμα os.system για τη δημιουργία αρχείων, αλλά αντ' αυτού τα ενσωματωμένα αρθρώματα zipfile ή tarfile. Είναι μέρος της πρότυπης βιβλιοθήκης και διαθέσιμα άμεσα προς χρήση, χωρίς εξωτερικές εξαρτήσεις, όπως η διαθεσιμότητα του προγράμματος συμπίεσης στον υπολογιστή μας.

Ωστόσο, χρησιμοποίησα για τη δημιουργία αντιγράφου ασφαλείας τον τρόπο με το άρθρωμα os.system στα παραπάνω παραδείγματα καθαρά για παιδαγωγικούς σκοπούς, ώστε να γίνει κατανοητό στον καθένα και να είναι χρηστικό παράλληλα.

Θα δοκιμάσετε την συγγραφή της πέμπτης έκδοσης με τη χρήση του αρθρώματος zipfile [4] αντί για την κλήση του os.system;

10.7. Η διαδικασία ανάπτυξης λογισμικού

Έχουμε ολοκληρώσει τα διάφορα **στάδια** της διαδικασίας συγγραφής λογισμικού. Ακολουθεί μια σύνοψη παρακάτω:

1. Τι (Ανάλυση)
2. Πώς (Σχεδιασμός)
3. Συγγραφή (Υλοποίηση)
4. Έλεγχος (Δοκιμές και αποσφαλμάτωση)
5. Χρήση (Λειτουργία ή ανάπτυξη)
6. Συντήρηση (Βελτίωση)

Ο συνιστώμενος τρόπος δημιουργίας προγραμμάτων είναι η διαδικασία που ακολουθήσαμε για τη συγγραφή του σεναρίου δημιουργίας αντιγράφων ασφάλειας: Αναλύστε και σχεδιάστε. Ξεκινήστε την εφαρμογή με μια απλή έκδοση. Δοκιμάστε και ελέγξτε για σφάλματα. Χρησιμοποιήστε την εφαρμογή για να δείτε αν λειτουργεί όπως θα θέλατε. Προσθέστε νέα χαρακτηριστικά και συνεχίστε επαναλαμβάνοντας τον κύκλο Συγγραφή-Έλεγχος-Χρήση όσες φορές απαιτείται. Να θυμάστε ότι **το λογισμικό καλλιεργείται, δεν κατασκευάζεται**.

Σύνοψη

Έχουμε δει τον τρόπο δημιουργίας των δικών μας προγραμμάτων/σεναρίων Python και τα διάφορα στάδια εξέλιξης για τη συγγραφή τους. Θα σας φανεί χρήσιμο να δημιουργήσετε τα δικά σας προγράμματα με τη μεθοδολογία που ακολουθήσαμε σε αυτό το κεφάλαιο, έτσι ώστε να αποκτήσετε ευχέρεια με την Python και την επίλυση προβλημάτων.

Στη συνέχεια θα συζητήσουμε για τον αντικειμενοστρεφή προγραμματισμό.

11. Αντικειμενοστρεφής προγραμματισμός

Εισαγωγή

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.

Σημείωση για προγραμματιστές στατικών γλωσσών

> Προσέξτε ότι 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). Δείτε την εντολή help(int) για περισσότερες λεπτομέρειες σχετικά με την κλάση.
>
> Οι προγραμματιστές C# και Java 1.5 θα βρουν ομοιότητες με την *boxing and unboxing* έννοια.

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.

11.1. 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.

Σημείωση για προγραμματιστές C++/Java/C#

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

Θα αναρωτιέστε πώς η Python ορίζει την τιμή για τη self και γιατί σας απαλλάσσει από τον χειροκίνητο ορισμό της. Ένα 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.

11.2. Κλάσεις

Η απλούστερη κλάση φαίνεται στο παρακάτω παράδειγμα.

```
#!/usr/bin/python
# Filename: simplestclass.py

class Person:
    pass # Ένα κενό μπλοκ


p = Person()
print(p)
```

Έξοδος:

```
$ python simplestclass.py <__main__.Person
object at 0x019F85F0>
```

Πώς δουλεύει:

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. (Θα μάθουμε περισσότερα για τη δημιουργία υποστάσεων στην επόμενη ενότητα). 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.

11.3. Μέθοδοι αντικειμένου

Έχουμε ήδη συζητήσει πως οι κλάσεις/αντικείμενα δέχονται μεθόδους όπως και συναρτήσεις, εκτός της πρόσθετης μεταβλητής self. Ας δούμε ένα παράδειγμα.

```
#!/usr/bin/python
# Filename: method.py

class Person:
    def sayHi(self):
```

```
        print('Hello, how are you?')

p = Person()
p.sayHi()

# Αυτό το σύντομο παράδειγμα θα μπορούσε να γραφτεί και ως
Person().sayHi()
```

Έξοδος:

```
$ python method.py
Hello, how are you?
```

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

11.4. 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.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: class_init.py

class Person:
    def __init__(self, name):
        self.name = name
    def sayHi(self):
        print('Hello, my name is', self.name)

p = Person('Swaroop')
p.sayHi()

# Αυτό το σύντομο παράδειγμα μπορεί να γραφτεί και ως
Person('Swaroop').sayHi()
```

Έξοδος:

```
$ python class_init.py Hello, my
name is Swaroop
```

Πώς δουλεύει:

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 sayHi.

11.5. Μεταβλητές κλάσης και αντικειμένου

Έχουμε ήδη συζητήσει τη λειτουργικότητα των κλάσεων και αντικειμένων (δηλαδή μεθόδων), ας μάθουμε τώρα για 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:

```
#!/usr/bin/python
# Filename: objvar.py

class Robot:
    '''Represents a robot, with a name.'''


    # Μια μεταβλητή κλάσης, που μετρά τον αριθμό των ρομπότ population = 0


    def __init__(self, name):
        '''Initializes the data.''' self.name =
        name
        print('(Initializing {0})'.format(self.name))

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

    def __del__(self):
        '''I am dying.'''
        print('{0} is being destroyed!'.format(self.name))
```

```
            Robot.population -= 1

            if Robot.population == 0:
                print('{0} was the last one.'.format(self.name)) else:

                print('There are still {0:d} robots
working.'.format(Robot.population))

    def sayHi(self):
        '''Greeting by the robot.

        Yeah, they can do that.'''
                                print('Greetings, my masters call me {0}.'.format(self.name))

    def howMany():
        '''Prints the current population.'''
        print('We have {0:d} robots.'.format(Robot.population)) howMany =
    staticmethod(howMany)

droid1 = Robot('R2-D2')
droid1.sayHi()
Robot.howMany()

droid2 = Robot('C-3PO')
droid2.sayHi()
Robot.howMany()

print("\nRobots can do some work here.\n")

print("Robots have finished their work. So let's destroy them.") del droid1

del droid2

Robot.howMany()
```

Έξοδος:

```
(Initializing R2-D2)
Greetings, my masters call me R2-D2. We have
1 robots.
(Initializing C-3PO)
Greetings, my masters call me C-3PO. We have
2 robots.

Robots can do some work here.

Robots have finished their work. So let's destroy them.
```

```
R2-D2 is being destroyed!
There are still 1 robots working.
C-3PO is being destroyed!
C-3PO was the last one.
We have 0 robots.
```

Πώς δουλεύει:

Είναι μεγάλο το παράδειγμα αλλά βοηθά στην κατανόηση της φύσης των μεταβλητών κλάσεων και αντικειμένων. 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. Να θυμάστε αυτή την απλή διαφορά μεταξύ των μεταβλητών κλάσεων και αντικειμένων. Σημειώστε επίσης ότι 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) [1].

```
@staticmethod
def howMany():
    '''Prints the current population.'''
    print('We have {0:d} robots.'.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 κάνει εδώ.

Σημείωση για προγραμματιστές C++/Java/C#

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).

11.6. 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.

Μια καλύτερη λύση θα ήταν να δημιουργηθεί μια κοινή κλάση με το όνομα SchoolMember και κατόπιν οι κλάσεις καθηγητές και μαθητές 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 SchoolMember, αυτομάτως ενημερώνονται και οι υποτύποι. Για παράδειγμα, ένα πεδίο αριθμού ταυτότητας για τους μαθητές και τους καθηγητές είναι εφικτό εύκολα με την προσθήκη του στην κλάση SchoolMember. 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 μαθητή ως αντικείμενο SchoolMember, θα μας φανεί χρήσιμο σε καταστάσεις για τον υπολογισμό του συνολικού αριθμού των μελών του κολλεγίου. 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 SchoolMember σε αυτή την περίπτωση ονομάζεται *βασική κλάση (base class)* ή *υπερκλάση (superclass)*. Οι κλάσεις Teacher και Student αποκαλούνται *παράγωγες κλάσεις (derived classes)* ή *υποκλάσεις (subclasses)*.

Θα δούμε το παράδειγμα αυτό τώρα σε ένα πρόγραμμα.

```
#!/usr/bin/python
# Filename: inherit.py

class SchoolMember:
    '''Represents any school member.''' def
    __init__(self, name, age):
```

```
            self.name = name
            self.age = age
            print('(Initialized SchoolMember: {0})'.format(self.name))

        def tell(self):
            '''Tell my details.'''
                                print('Name:"{0}" Age:"{1}"'.format(self.name, self.age), end="
")

class Teacher(SchoolMember):
    '''Represents a teacher.'''
        def __init__(self, name, age, salary):
            SchoolMember.__init__(self, name, age) self.salary
            = salary
            print('(Initialized Teacher: {0})'.format(self.name))

        def tell(self):
            SchoolMember.tell(self)
            print('Salary: "{0:d}"'.format(self.salary))

class Student(SchoolMember):
    '''Represents a student.'''
        def __init__(self, name, age, marks):
            SchoolMember.__init__(self, name, age)
            self.marks = marks
            print('(Initialized Student: {0})'.format(self.name))

        def tell(self):
            SchoolMember.tell(self)
            print('Marks: "{0:d}"'.format(self.marks))

t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('Swaroop', 25, 75)

print() # εκτυπώνει μια κενή γραμμή

members = [t, s]
for member in members:
    member.tell() # works for both Teachers and Students
```

Έξοδος:

```
$ python inherit.py
(Initialized SchoolMember: Mrs. Shrividya) (Initialized
Teacher: Mrs. Shrividya) (Initialized SchoolMember:
Swaroop) (Initialized Student: Swaroop)
```

```
Name:"Mrs. Shrividya" Age:"40" Salary: "30000"
Name:"Swaroop" Age:"25" Marks: "75"
```

Πώς δουλεύει:

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 Teacher ή Student ως απλές υποστάσεις της κλάσης SchoolMember, όταν χρησιμοποιούμε τη μέθοδο tell της κλάσης SchoolMember.

Επίσης, παρατηρήστε ότι καλείται η μέθοδος tell του υποτύπου και όχι η μέθοδος tell της κλάσης SchoolMember. 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*.

Σύνοψη

Έχουμε ως τώρα εξερευνήσει τις διάφορες πτυχές των κλάσεων και αντικειμένων, όπως επίσης και τις σχετικές ορολογίες. Έχουμε εξετάσει επίσης τα ωφέλη και τις παγίδες του αντικειμενοστρεφούς προγραμματισμού. Η Python είναι ιδιαίτερα αντικειμενοστρεφής και η προσεκτική κατανόηση αυτών των εννοιών θα σας ωφελήσει μακροπρόθεσμα.

Στη συνέχεια θα μάθουμε πώς να χειριστούμε την είσοδο/έξοδο και πώς την προσπέλαση αρχείων στην Python.

12. Είσοδος έξοδος

Εισαγωγή

Θα υπάρξουν καταστάσεις όπου το πρόγραμμά σας πρέπει να αλληλεπιδράσει με το χρήστη. Για παράδειγμα, θα θέλετε να πάρετε είσοδο από το χρήστη και μετά να τυπώσετε 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, μπορείτε να χρησιμοποιήσετε τη μέθοδο rjust για να πάρετε μια συμβολοσειρά, η οποία είναι δεξιά στοιχισμένη (right justified) σε ένα καθορισμένο εύρος. Κοιτάξτε τη 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 εξερευνήσουμε αυτή την πτυχή σε αυτό το κεφάλαιο.

12.1. Είσοδος από το χρήστη

```
#!/usr/bin/python
# user_input.py

def  reverse(text): return
      text[::-1]

def is_palindrome(text):
      return text == reverse(text)

something = input('Enter text: ') if
(is_palindrome(something)):
      print("Yes, it is a palindrome") else:

      print("No, it is not a palindrome")
```

Έξοδος:

```
$ python user_input.py Enter
text: sir
No, it is not a palindrome

$ python user_input.py Enter
text: madam
Yes, it is a palindrome

$ python user_input.py Enter
text: racecar Yes, it is a
palindrome
```

Πώς δουλεύει:

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 [1].

Εργασία για το σπίτι:

Ο έλεγχος εάν ένα κείμενο είναι παλίνδρομο θα έπρεπε επίσης να αγνοεί τη στίξη, τα διαστήματα και τα κεφαλαία. Για παράδειγμα, το κείμενο "Rise to vote, sir" είναι παλίνδρομο, αλλά το τρέχον πρόγραμμά μας δεν το λέει. Μπορείτε να αναπτύξετε το ανωτέρο πρόγραμμα για να το αναγνωρίσει σαν παλίνδρομο;

12.2. 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.

Παράδειγμα:

```
#!/usr/bin/python
# Filename: using_file.py

poem = '''\ Programming is
fun When the work is done

if you wanna make your work also fun: use
    Python!
'''

f = open('poem.txt', 'w') # open for 'w'riting f.write(poem) #
write text to file
f.close() # close the file

f = open('poem.txt') # if no mode is specified, 'r'ead mode is assumed by default

while True:
    line = f.readline()
    if len(line) == 0: # Zero length indicates EOF break

    print(line, end='') f.close() #
close the file
```

Έξοδος:

```
$ python using_file.py
Programming is fun When
the work is done
if you wanna make your work also fun: use
        Python!
```

Πώς δουλεύει:

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.

Στο δικό μας 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.

Τώρα, ελέγξτε τα περιεχόμενα του αρχείου poem.txt, για να επιβεβαιώσετε ότι το πρόγραμμα έχει πραγματικά γραφτεί και διαβαστεί από αυτό το αρχείο.

12.3. 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).

Παράδειγμα:

```
#!/usr/bin/python
# Filename: pickling.py

import pickle

# the name of the file where we will store the object shoplistfile =
'shoplist.data'
# the list of things to buy
shoplist = ['apple', 'mango', 'carrot']

# Write to the file
f = open(shoplistfile, 'wb')
```

```
pickle.dump(shoplist, f) # dump the object to a file f.close()


del shoplist # destroy the shoplist variable

# Read back from the storage f =
open(shoplistfile, 'rb')
storedlist = pickle.load(f) # load the object from the file print(storedlist)
```

Έξοδος:

```
$ python pickling.py ['apple', 'mango',
'carrot']
```

Πώς δουλεύει:

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*.

Σύνοψη

Έχουμε συζητήσει διάφορους τύπους εισόδου/εξόδου, καθώς επίσης διαχείριση αρχείου και χρήση του αρθρώματος pickle.

Στη συνέχεια θα εξερευνήσουμε την έννοια των εξαιρέσεων (exceptions).

13.Εξαιρέσεις

13.1. Εξαιρέσεις

Οι εξαιρέσεις εμφανίζονται όταν ορισμένες *εξαιρετικές* καταστάσεις συμβαίνουν στο πρόγραμμά σας. 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**.

Σφάλματα

Σκεφτείτε μια απλή κλήση της συνάρτησης print. Τι συμβαίνει αν γράψαμε ανορθόγραφα 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).

```
>>> Print('Hello World')
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module> Print('Hello
    World')
NameError: name 'Print' is not defined
>>> print('Hello World')
Hello World
```

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.

Εξαιρέσεις

Θα **δοκιμάσουμε** (try) να διαβάσουμε είσοδο από το χρήστη. Πιέστε ctrl-d και κοιτάξτε τι συμβαίνει.

```
>>> s = input('Enter something --> ') Enter
something -->
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.

13.2. Χειρισμοί εξαιρέσεων

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.

```
#!/usr/bin/python
# Filename: try_except.py

try:
    text = input('Enter something --> ') except
EOFError:
    print('Why did you do an EOF on me?') except
KeyboardInterrupt:
    print('You cancelled the operation.') else:

    print('You entered {0}'.format(text))
```

Έξοδος:

```
$ python try_except.py
Enter something --> # Press ctrl-d Why did you do
an EOF on me?

$ python try_except.py
Enter something --> # Press ctrl-c You cancelled
the operation.

$ python try_except.py
Enter something --> no exceptions You
entered no exceptions
```

Πώς δουλεύει:

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. Το έχουμε δει ήδη σε ενέργεια παραπάνω.

Μπορείτε επίσης να έχετε μια πρόταση else συνδεδεμένη με μια πλοκάδα try..except. Η πρόταση else εκτελείται εάν δεν συμβαίνει καμμία εξαίρεση.

Στο επόμενο παράδειγμα θα δούμε επίσης, πώς να παίρνουμε το αντικείμενο της εξαίρεσης, έτσι ώστε να μπορούμε να ανακτούμε επιπρόσθετες πληροφορίες.

13.3. 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.

```
#!/usr/bin/python
# Filename: raising.py

class ShortInputException(Exception):
    '''A user-defined exception class.''' def
    __init__(self, length, atleast):
        Exception.__init__(self)
        self.length = length self.atleast =
        atleast

try:
    text = input('Enter something --> ') if len(text) <
    3:
        raise ShortInputException(len(text), 3)
    # Other work can continue as usual here except
EOFError:
    print('Why did you do an EOF on me?') except
ShortInputException as ex:
    print('ShortInputException: The input was {0} long, expected at least {1}'\

          .format(ex.length, ex.atleast))
else:
    print('No exception was raised.')
```

Έξοδος:

```
$ python raising.py Enter
something --> a
ShortInputException: The input was 1 long, expected at least 3

$ python raising.py Enter
something --> abc No exception
was raised.
```

Πώς δουλεύει:

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.

13.4. 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. Σημειώστε ότι 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.

```
#!/usr/bin/python
# Filename: finally.py

import time

try:
    f = open('poem.txt')
    while True: # our usual file-reading idiom line =
        f.readline()
        if len(line) == 0: break

        print(line, end='')
        time.sleep(2) # To make sure it runs for a while except
KeyboardInterrupt:
    print('!! You cancelled the reading from the file.') finally:

    f.close()
    print('(Cleaning up: Closed the file)')
```

Έξοδος:

```
$ python finally.py
Programming is fun When
the work is done
if you wanna make your work also fun:
!! You cancelled the reading from the file. (Cleaning up:
Closed the file)
```

Πώς δουλεύει:

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.

13.5. 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:

```
#!/usr/bin/python
# Filename: using_with.py

with open("poem.txt") as f: for line
    in f:
        print(line, end='')
```

Πώς δουλεύει:

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 " thefile" 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.

Περισσότερη συζήτηση για αυτό το θέμα είναι πέρα από το πεδίο αυτού του βιβλίου, έτσι παρακαλώ αναφερθείτε στο PEP343 [1] για πλήρη επεξήγηση.

Σύνοψη

Έχουμε συζητήσει την χρήση των εντολών try..except και try..finally. Έχουμε δει πως να δημιουργούμε τους δικούς μας τύπους εξαιρέσεων και πως να αναδεικνύουμε, επίσης, εξαιρέσεις.

Κατόπιν θα εξερευνήσουμε την πρότυπη βιβλιοθήκη της Python (Python Standard Library).

14.Πρότυπη βιβλιοθήκη

Εισαγωγή

Η πρότυπη βιβλιοθήκη της Python περιέχει έναν τεράστιο αριθμό χρήσιμων αρθρωμάτων --~~~~Insert non-formatted text here--~~~~~~~~~~~~~~~~~~~~ ---- ---- ---- ---- [[[http://www.example.com link title]]]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 εξερευνήσουμε μερικά από τα πιο συνηθισμένα αρθρώματα σε αυτή τη βιβλιοθήκη. Μπορείτε να βρείτε ολοκληρωμένες λεπτομέρειες για όλα τα αρθρώματα της πρότυπης βιβλιοθήκης, στο 'Library Reference' section [1] της τεκμηρίωσης που συνοδεύει κάθε εγκατάσταση της Python.
Ας εξερευνήσουμε μερικά χρήσιμα αρθρώματα.

Σημείωση

> Εάν βρείτε τα θέματα σε αυτό το κεφάλαιο πολύ προχωρημένα (advanced) μπορείτε να παραλείψετε αυτό το κεφάλαιο. Πάντως, εγώ σας συνιστώ να επιστρέψετε σε αυτό το κεφάλαιο, όταν νιώθετε πιο άνετα προγραμματίζοντας με Python.

14.1. 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, έτσι ώστε να επιβεβαιώσουμε ότι χρησιμοποιούμε τουλάχιστον την έκδοση 3. Το άρθρωμα sys μας δίνει αυτή τη λειτουργία.

```
>>> import sys
>>> sys.version_info    (3,
0, 0, 'beta', 2)
>>> sys.version_info[0] >= 3
True
```

Πώς δουλεύει:

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 3.0:

```
#!/usr/bin/python
# Filename: versioncheck.py import
sys, warnings
if sys.version_info[0] < 3:
    warnings.warn("Need Python 3.0 for this program to run",
        RuntimeWarning)
else:
    print('Proceed as normal')
```

Έξοδος:

```
$ python2.5 versioncheck.py
versioncheck.py:6: RuntimeWarning: Need Python 3.0 for this program to run

  RuntimeWarning)

$ python3 versioncheck.py
Proceed as normal
```

Πώς δουλεύει:

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.

14.2. 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:

```
#!/usr/bin/python
# Filename: use_logging.py import
os, platform, logging

if platform.platform().startswith('Windows'): logging_file =
    os.path.join(os.getenv('HOMEDRIVE'),
os.getenv('HOMEPATH'), 'test.log') else:

    logging_file = os.path.join(os.getenv('HOME'), 'test.log')

print("Logging to", logging_file)

logging.basicConfig( level=logging.DEBUG,

    format='%(asctime)s : %(levelname)s : %(message)s', filename =
    logging_file,
    filemode = 'w',
)

logging.debug("Start of the program")
logging.info("Doing something")
logging.warning("Dying now")
```

Έξοδος:

```
$python use_logging.py
Logging to C:\Users\swaroop\test.log
```

Αν ελέγξουμε τα περιεχόμενα του αρχείου test.log, θα μοιάζουν με τα εξής:

```
2008-09-03 13:18:16,233 : DEBUG : Start of the program 2008-09-03
13:18:16,233 : INFO : Doing something
```

```
2008-09-03 13:18:16,233 : WARNING : Dying now
```

Πώς δουλεύει:

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 καμμία πληροφορία δεν εμφανίστηκε στον χρήστη που τρέχει το πρόγραμμα.

14.3. Τα αρθρώματα urllib και json

Δε θα ήταν φοβερό εάν μπορούσαμε να γράψουμε το δικό μας πρόγραμμα που θα έψαχνε για αποτελέσματα στο διαδίκτυο; Ας το εξερευνήσουμε τώρα.

Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας μερικά αρθρώματα. Αρχικά είναι το άρθρωμα urllib, το οποίο μπορούμε να χρησιμοποιήσουμε για να "πιάσουμε" οποιαδήποτε ιστοσελίδα από το διαδίκτυο. Θα χρησιμοποιήσουμε το Yahoo! Search για να πάρουμε τα αποτελέσματα της αναζήτησης και ευτυχώς μπορεί να μας δώσει τα αποτελέσματα σε ένα μορφότυπο που ονομάζεται JSON, ο οποίος είναι εύκολος για εμάς να αναλύσουμε, επειδή υπάρχει το ενσωματωμένο άρθρωμα json στην πρότυπη βιβλιοθήκη.

TODO

Αυτό το πρόγραμμα δεν δουλεύει ακόμα και φαίνεται ότι είναι σφάλμα (bug) στην Python 3.0 beta 2 [2].

```
#!/usr/bin/python
# Filename: yahoo_search.py

import sys
if sys.version_info[0] != 3:
    sys.exit('This program needs Python 3.0')

import json
import urllib, urllib.parse, urllib.request, urllib.response

# Get your own APP ID at http://developer.yahoo.com/wsregapp/
YAHOO_APP_ID =
```

```
'jl22psvV34HELWhdfUJbfDQzlJ2B57KFS_qs4I8D0Wz5U5_yCI1Awv8.IBSfPhwr'
SEARCH_BASE = 'http://search.yahooapis.com/WebSearchService/V1/webSearch'


class YahooSearchError(Exception): pass


# Taken from http://developer.yahoo.com/python/python-json.html def search(query,
results=20, start=1, **kwargs):
    kwargs.update({
        'appid': YAHOO_APP_ID,
        'query': query, 'results':
        results, 'start': start, 'output':
        'json'

    })
    url = SEARCH_BASE + '?' + urllib.parse.urlencode(kwargs) result =
    json.load(urllib.request.urlopen(url))
    if 'Error' in result:
        raise YahooSearchError(result['Error']) return
    result['ResultSet']

query = input('What do you want to search for? ') for result in
search(query)['Result']:
    print("{0} : {1}".format(result['Title'], result['Url']))
```

Έξοδος:

TODO

Πώς δουλεύει:

Μπορούμε να πάρουμε τα αποτελέσματα της αναζήτησης από ένα συγκεκριμένο δικτυακό τόπο, δίνοντας το κείμενο που ψάχνουμε σε ένα ιδιαίτερο μορφότυπο. Πρέπει να καθορίσουμε πολλές επιλογές, τις οποίες συνδυάζουμε χρησιμοποιώντας το μορφότυπο key1=value1&key2=value2, ο οποίος χειρίζεται από τη συνάρτηση urllib.parse.urlencode().

Έτσι για παράδειγμα, ανοίξτε αυτό το σύνδεσμο στο φυλλομετρητή σας [3] και θα δείτε 20 αποτελέσματα, αρχίζοντας από το πρώτο αποτέλεσμα, για τις λέξεις "byte of python", και ζητάμε την έξοδο στο μορφότυπο JSON.

Κάνουμε μια σύνδεση σε αυτή τη διεύθυνση (URL) χρησιμοποιώντας τη συνάρτηση urllib.request.urlopen() και περνούμε αυτό το αρχείο να το χειριστεί η json.load(), η οποία θα διαβάσει το περιεχόμενο και ταυτόχρονα θα το μετατρέψει σε αντικείμενο της Python. Τότε κάνουμε βρόχο μέσω αυτών των αποτελεσμάτων και το παρουσιάζουμε στον τελικό χρήστη.

Το άρθρωμα της εβδομάδας

Υπάρχουν πολλά περισσότερα για να εξερευνηθούν στην πρότυπη βιβλιοθήκη όπως η εκσφαλμάτωση [4], ο χειρισμός επιλογών της γραμμής εντολών [5], οι κανονικές εκφράσεις [6] κ.τ.λ.

Ο καλύτερος τρόπος για να εξερευνήσετε περισσότερο την πρότυπη βιβλιοθήκη είναι, διαβάζοντας την υπέροχη σειρά του Doug Hellmann's, Python Module of the Week [7].

Σύνοψη

Έχουμε εξερευνήσει μερικές λειτουργίες από πολλά αρθρώματα της πρότυπης βιβλιοθήκης της Python. Συνιστάται να πλοηγηθείτε στην τεκμηρίωση της Python Standard Library [1], για να πάρετε μια ιδέα από όλα τα αρθρώματα που είναι διαθέσιμα.

Κατόπιν θα καλύψουμε διάφορες πτυχές της Python, που θα κάνουν το ταξίδι μας στην Python πιο *ολοκληρωμένο*.

15.Περισσότερα

Εισαγωγή

Μέχρι στιγμής έχουμε καλύψει την πλειοψηφία από διάφορες πτυχές της Python που θα χρησιμοποιήσετε. Σε αυτό το κεφάλαιο, θα καλύψουμε μερικές ακόμα που θα κάνουν τη γνώση σας για την Python πιο σφαιρική.

15.1. Πέρα δώθε οι πλειάδες

Θέλατε ποτέ να επιστρέψετε δύο διαφορετικές τιμές από μια συνάρτηση; Μπορείτε. Το μόνο που χρειάζεται είναι να χρησιμοποιήσετε μια πλειάδα.

```
>>> def get_error_details():
...         return (2, 'second error details')
...
>>> errnum, errstr = get_error_details()
>>> errnum
2
>>> errstr
'second error details'
```

Παρατηρήστε 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:

```
>>> a, *b = [1, 2, 3, 4]
>>> a
1
>>> b
[2, 3, 4]
```

Αυτό σημαίνει επίσης ότι ο πιο γρήγορος τρόπος για να ανταλλάξετε τις τιμές δύο μεταβλητων στην Python είναι:

```
>>> a = 5; b = 8
>>> a, b = b, a
>>> a, b
(8, 5)
```

15.2. Ειδικές μέθοδοι

Υπάρχουν μερικές μέθοδοι όπως η __init__ και η __del__ οι οποίες έχουν ειδική σημασία στις κλάσεις.

Οι ειδικές μέθοδοι μιμούνται μερικές συμπεριφορές των ενσωματωμένων τύπων. Για παράδειγμα, αν θέλετε να χρησιμοποιήσετε τη λειτουργία ευρετηρίασης x[key] για την κλάση σας (όπως τη χρησιμοποιείτε για τις λίστες και τις πλειάδες), τότε το μόνο που χρειάζεται να κάνετε είναι να υλοποιήσετε τη μέθοδο __getitem__() και η δουλειά σας έγινε. Αν το καλοσκεφτείτε, αυτό είναι που κάνει η Python για την ίδια την κλάση list!

Στον ακόλουθο πίνακα εμφανίζονται μερικές χρήσιμες ειδικές μέθοδοι. Αν θέλετε τα μάθετε τα πάντα για τις ειδικές μεθόδους δείτε το εγχειρίδιο [1].

Όνομα	Εξήγηση
__init__(self, ...)	Αυτή η μέθοδος καλείται αμέσως πριν επιστραφεί για χρήση το αντικείμενο που μόλις δημιουργήθηκε.
__del__(self)	Καλείται αμέσως πριν καταστραφεί το αντικείμενο
__str__(self)	Καλείται όταν χρησιμοποιούμε τη συνάρτηση print ή όταν χρησιμοποιείται η str().
__lt__(self, other)	Καλείται όταν χρησιμοποιείται ο τελεστής *μικρότερο από* (<). Ομοίως, υπάρχουν ειδικές μέθοδοι για όλους τους τελεστές (+, >, κ.λπ.).
__getitem__(self, key)	Καλείται όταν χρησιμοποιείται ο τελεστής ευρετηρίασης x[key].
__len__(self)	Καλείται όταν χρησιμοποιείται η ενσωματωμένη συνάρτηση len() για το αντικείμενο ακολουθίας.

15.3. Πλοκάδες μοναδικών εντολών

Είδαμε ότι κάθε πλοκάδα εντολών ξεχωρίζεται από τις υπόλοιπες από το δικό της επίπεδο εσοχής του κώδικα. Λοιπόν, 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. Το πιο κάτω παράδειγμα θα πρέπει να ξεκαθαρίσει κάπως τα πράγματα:

```
>>> flag = True
>>> if flag: print 'Yes'
...
Yes
```

Παρατηρήστε 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", σας συνιστώ 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.

15.4. Lambda Forms

Η εντολή lambda χρησιμοποιείται για να δημιουργήσει νέα αντικείμενα συναρτήσεων και να τα επιστρέψει κατά την εκτέλεση.

```
#!/usr/bin/python
# Filename: lambda.py

def make_repeater(n): return
    lambda s: s * n

twice = make_repeater(2)

print(twice('word'))
print(twice(5))
```

Έξοδος:

```
$ python lambda.py
wordword
10
```

Πώς δουλεύει:

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.

TODO

Μπορούμε να κάνουμε ένα list.sort() παρέχοντας μια συνάρτηση σύγκρισης δημιουργημένη από μια lambda;

```
points = [ { 'x' : 2, 'y' : 3 }, { 'x' : 4, 'y' : 1 } ]
# points.sort(lambda a, b : cmp(a['x'], b['x']))
```

15.5. Κατανόηση λιστών

Η κατανόηση λιστών (List comprehension) χρησιμοποιείται για να εξάγουμε μια νέα λίστα από μια υφιστάμενη. Ας πούμε ότι έχετε μια λίστα από αριθμούς και θέλετε να πάρετε μια λίστα με τους αριθμούς της υφιστάμενης λίστα πολλαπλασιασμένους επί 2 μόνο όταν ο ίδιος ο αριθμός είναι μεγαλύτερος από 2. Η κατανόηση λίστας είναι ιδανική για τέτοιες περιπτώσεις.

```
#!/usr/bin/python
# Filename: list_comprehension.py

listone = [2, 3, 4]
listtwo = [2*i for i in listone if i > 2] print(listtwo)
```

Έξοδος:

```
$ python list_comprehension.py [6, 8]
```

Πώς δουλεύει:

Εδώ, έχουμε εξάγει μια νέα λίστα ορίζοντας ένα χειρισμό (2*i) όταν ικανοποιείται μια συνθήκη (if i > 2). Παρατηρήστε ότι η αρχική λίστα δεν τροποποιείται.

Το πλεονέκτημα της χρήσης κατανόησης λιστών είναι ότι μειώνει την ποσότητα του επαναλαμβανόμενου κώδικα που απαιτείται όταν χρησιμοποιούμε βρόχους για να επεξεργαστούμε κάθε ένα στοιχεία μιας λίστας και να το αποθηκεύσουμε σε μια νέα λίστα.

15.6. Αποστολή πλειάδων και λεξικών σε συναρτήσεις

Υπάρχει ένα ειδικός τρόπος αποδοχής παραμέτρων σε μια συνάρτηση ως πλειάδα ή λεξικό χρησιμοποιώντας το πρόθεμα * ή το ** αντίστοιχα. Αυτό χρησιμεύει για να παίρνουμε ένα μεταβλητό αριθμό ορισμάτων σε μια συνάρτηση.

```
>>> def powersum(power, *args):
...     '''Return the sum of each argument raised to specified power.'''
...     total = 0
...     for i in args:
```

```
...             total += pow(i, power)
...         return total
...
>>> powersum(2, 3, 4)
25

>>> powersum(2, 10)
100
```

Επειδή έχουμε το πρόθεμα * στη μεταβλητή args, όλα τα επιπλέον ορίσματα που περνούν στη συνάρτηση αποθηκεύονται στην args ως πλειάδα. Αν είχε χρησιμοποιηθεί ένα πρόθεμα **, οι επιπλέον παράμετροι θα θεωρούνται ζεύγη κλειδιών-τιμών ενός λεξικού.

15.7. exec και 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:

```
>>> exec('print("Hello World")') Hello
World
```

Με παρόμοιο τρόπο, η συνάρτηση eval χρησιμοποιείται για να αποτιμήσει έγκυρες εκφράσεις της Python οι οποίες αποθηκεύονται σε μια συμβολοσειρά. Ένα από παράδειγμα φαίνεται παρακάτω.

```
>>> eval('2*3') 6
```

15.8. 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.

```
>>> mylist = ['item']
>>> assert len(mylist) >= 1
>>> mylist.pop()
'item'
>>> mylist
[]
>>> assert len(mylist) >= 1 Traceback (most
recent call last):
  File "<stdin>", line 1, in <module>
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.

15.9. 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.

```
>>> i = []
>>> i.append('item')
>>> repr(i)
"['item']"
>>> eval(repr(i))
['item']
>>> eval(repr(i)) == i True
```

Βασικά, 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.

Σύνοψη

Καλύψαμε μερικές ακόμα δυνατότητες της Python σ' αυτό το κεφάλαιο, αλλά και πάλι δεν τις έχουμε καλύψει όλες. Ωστόσο, σ' αυτό το στάδιο έχουμε μιλήσει τα περισσότερα από τα χαρακτηριστικά που θα χρειαστείτε στην πράξη. Αυτά είναι αρκετά για να ξεκινήσετε με οποιαδήποτε προγράμματα σκοπεύετε να δημιουργήσετε.

16. Ασκήσεις Κατανόησης

- Εγκαταστήστε την Python 3k και γράψτε ένα πρόγραμμα που να τυπώνει στην οθόνη σας το Hello World. Προαιρετικά : Γράψτε έναν οδηγό για το πως κάνατε την εγκατάσταση και σε ποιο σύστημα.

- Γράψτε ένα απλό μενού εστιατορίου, όπου παρουσιάζεται ένα μενού στον πελάτη, όπως για παράδειγμα :

  1. Κολοκυθάκια γιαχνί
  2. Πατάτες στα κάρβουνα
  3. Κοτόπουλο γεμιστό
  4. Σαλιγκάρια

  Τι θα επιθυμούσατε ; Και ανάλογα με την επιλογή που δίνει ο χρήστης να εμφανίζεται ένα κατάλληλο μήνυμα, ή μήνυμα λάθος αν δεν ανήκει στο εύρος αποδεκτών τιμών.

- Υπολογίστε τις λύσεις μια εξίσωσης δευτέρου βαθμού, ζητώντας από τον χρήστη τους συντελεστές α, β, γ. Υπολογίστε και μιγαδικές ρίζες αν η διακρίνουσα είναι αρνητική.

- Βρείτε τους 20 πρώτους αριθμούς Fibonacci.

- Προσθέστε όλους του φυσικούς αριθμούς που είναι κάτω του 1000 και είναι πολλαπλάσια του 3 και του 7.

- Δημιουργείστε ένα πρόγραμμα που δέχεται ως είσοδο μια πρόταση και την αντι- στρέφει.

- Δημιουργείστε ένα πρόγραμμα που κόβει ένα αλφαριθμητικό σε συγκεκριμένους χαρακτήρες

- Δημιουργείστε ένα πρόγραμμα που :

  1. Μετράει πόσοι χαρακτήρες υπάρχουν σε ένα αρχείο
  2. Πόσες λέξεις υπάρχουν
  3. Πόσες προτάσεις υπάρχουν
  4. Πόσοι χαρακτήρες νέας γραμμής

- Βρείτε όλους τους πρώτους αριθμούς που βρίσκονται ανάμεσα στο 1 έως το 100. Υπόδειξη: Μπορείτε να ψάχνετε αν ένας αριθμός έχεις διαιρέτες μέχρι τον προς τα κάτω ακέραιο της τετραγωνικής του ρίζας.

- Γράψτε ένα απλό πρόγραμμα το οποίο ταξινομεί μια λίστα με ακεραίους. Επιβε- βαιώστε την ορθή λειτουργία του με έναν ελεγκτή ορθότητας .
  Bonus: Κάντε το παραπάνω με δύο τρόπους. Οι πιο απλοί τρόποι ίσως να είναι insertion sort, bublesort, quicksort.
  Υπόδειξη : Ως ελεγκτής ορθότητας μπορεί να χρησιμοποιηθεί ένα προγραμμα που κάνει iterate πάνω από τα ταξινομένα στοιχεία και επιβεβαιώνει ότι κάθε στοιχείο i είναι μικρότερο ή ίσο από το i + 1 συγκρίνοντας τα.

- Υλοποιήστε τον αλγόριθμο binary search. Ο αλγόριθμος αυτός δέχεται ως είσοδο ένα ταξινομένο πίνακα (ή λίστα) και ελέγχει πάντα το μεσαίο στοιχείο των (υπολειπόμενων) στοιχείων. Αναλόγως με το αποτέλεσμα της σύγκρισης συνεχίζεται ο έλεγχος για την αναζήτηση της τιμής στόχου μέχρι να βρεθεί η επιθυμητή τιμή.

- Δημιουργήστε δυο λίστες με n στοιχεία. Η πρώτη θα έχει n ονόματα ενώ η δεύτερη θα έχει n τηλέφωνα. Στην συνέχεια, γράψτε μια συνάρτηση που συνδιάζει αυτές τις δύο λίστες και παράγει μέσω μιας δομής dictionary έναν τηλεφωνικό κατάλογο.

- Ζητήστε από τον χρήστη να εισάγει ένα όνομα μαθήματος και το βαθμό που πήρε σε αυτό. Ύστερα με την χρήση λεξικού, θεωρήστε ως :

  • Κλειδί (key) το όνομα του μαθήματος
  • Τιμή (value) το βαθμό σε αυτό το μάθημα και αποθηκεύστε τα στην δομή.

  Στην συνέχεια, αφού τυπώσετε ως επιβεβαίωση τα δεδομένα που εισήγαγε ο χρήστης διατρέχοντας το λεξικό, υπολογίστε τον μέσο όρο (Μ.0.) με ακρίβεια δυο ψηφίων και τυπώστε τον.

- Μέσω του hashing μπορούμε να αντιστοιχίσουμε σε ένα σύνολο τιμών, ένα μι- κρότερο όσον αφορά την πληθικότητα σύνολο τιμών. Η συνάρτηση hash() της Python παίρνει ως όρισμα ένα αλφαριθμητικό (string) και επιστρέφει έναν αριθμό.
  Θα χρησιμοποιήσουμε την hash καθώς και το τελεστή υπόλοιπο διαίρεσης (modulo) σε αυτό το πρόβλημα. Θα έχουμε μια μυστική λέξη (password), στην οποία θα κάνουμε hash και στην συνέχεια modulo κατά έναν τριψήφιο ή τε- τραψήφιο ακέραιο αριθμό. Αυτός ο αριθμός (K EY), όπως και η μυστική λέξη (password), θα ορίζονται στην αρχή του προγράμματος. Στην συνέχεια θα δίνονται 3 ευκαιρίες στον χρήστη να μαντέψει την μυστική λέξη προκειμένου να του εμφανίσει ένα μήνυμα. Στην λέξη που θα εισάγεται από τον χρήστη, θα γίνεται hashing και modulokαι στην συνέχεια θα συγκρίνεται με την τιμή που προέκυψε από την πραγματική μυστική λέξη. Αν οι δυο τιμές ται- γιάζουν, τότε με μεγάλη πιθανότητα έχει εισαχθεί η μυστική λέξη. Προαιρετικά, στην συνέχεια μπορείτε να εξακριβώνεται αν όντως έχει εισαχθεί η σωστή λέξη.

- Υπάρχουν δυο παίκτες. Το παιχνίδι αρχίζει με 100 σπίρτα. Ο κάθε ένας παίκτης, μπορεί να τραβήξει από [1, 5] σπίρτα. Νικητής είναι αυτός που τραβάει τραβάει τα τελευταία σπίρτα. Πρώτα παίζει ο παίκτης Α, και ύστερα ο παίκτης Β, εναλλάξ.
  Υπόδειξη : Υλοποιήστε ένα διπλά εμφολευμένο βρόγχο, όπου στον εξωτερικό βρόγχο, συνεχίζεται το παιχνίδι όσο απομένουν σπίρτα και γίνεται εναλλαγή των παικτών, ενώ στον εσωτερικό, \ητείται έγκυρη είσοδος από τον χρήστη.

- Δημιουργείστε ένα πρόγραμμα το οποίο να παραθέτει τα ονόματα όλων των αρχείων που βρίσκονται κάτω από μια διαδρομή και όλους τους υποφακέλους της.
  Υπόδειξη : Χρησιμοποιείστε την βιβλιοθήκη os. Μπορείτε να χρησιμοποιείτε κάθε φορά αναδρομικά μια συνάρτηση που για κάθε φάκελο που βρίσκει καλεί τον εαυτό της και τυπώνει και όλα τα αρχεία του τρέχοντος φακέλου.

- Θα μελετήσουμε τις διαφορές ανάμεσα σε αλφαριθμητικά που απεικονίζουν την αλληλουχία βάσεων σε DNA με σκοπό να βρούμε :

  1. Την απόσταση δυο αλυσίδων DNA μεταξύ τους.
  2. Το ζεύγος των εγγύτερων χρωμοσωμάτων, δηλαδή ποιες αλυσίδες χρωμοσωμάτων διαφέρουν στις λιγότερες βάσεις μεταξύ τους.
  3. Ομοίως για τα αλυσίδες DNA που απέχουν την μεγαλύτερη απόσταση.

  Η μελέτη και η εφαρμογή των μεθόδων που θα αναπτύξετε, θα εφαρμοστεί στις παρακάτω αλληλουχίες, οι οποίες και θα οριστούν στην αρχή του προγράμματος σας.

  ```
  A = "g t g g c a a c g t g c"
  B = "g t a g c a g c g c g c"
  C = "g c g g c a c a g g g t"
  D = "g t g a c a a c g t g c"
  ```

  Υπόδειξη : Για να υπολογίσετε την απόσταση δύο αλυσίδων DNA, μπορείτε να χρησιμοποιήσετε κάποιον από τους ακόλουθους τρόπους :

  1. Να εξετάσετε βάση προς βάση κάθε αλληλουχία και να μετρήσετε των αριθμό των διαφορετικών βάσεων

  2. Να εξετάσετε ξεχωριστά τον αριθμό των :

- Αδενίνη (A)
- Κυτοσίνη C)
- Γουανίνη (G)
- Θυμίνη (T)

Μπορείτε να γράψετε τον απαραίτητο κώδικα χωρίς να χρησιμοποιήσετε βρόγχους επανάληψης ή περίπλοκους μαθηματικούς υπολογισμούς αν κάνετε χρήση συναρτήσεων μέλους (member functions).

- Πόσο εύκολο είναι να φτιάξει κάποιος ένα πρόγραμμα που να κάνει αυτόματο ορθογραφικό έλεγχο σε ένα κείμενο που είναι στα αγγλικά χρησιμοποιώντας Python. Ας δούμε σε μεγαλύτερη κλίμακα το πρόβλημα με την σύγκριση αλφαριθμητικών που είδαμε μόλις πριν.
  Η έξοδος που απαιτείται και θα ελέγχεται με βάση τις προτιμήσεις του χρήστη (πχ θα μπορεί να υπάρχει silent mode) είναι :

  1. Πόσες λέξεις τροποποιήθηκαν
  2. Ποιες λέξεις άλλαξαν

- Δημιουργείστε μια αριθμομηχανή που να υποστηρίζει :

  - Πρόσθεση
  - Αφαίρεση
  - Πολλαπλασιασμό
  - Διαίρεση

  Οι όροι θα εισάγονται σε ξεχωριστά πεδία. Προσοχή θέλει κατά την διαίρεση όπου θα πρέπει να κάνετε χειρισμό της εξαίρεσης που παρουσιάζεται αν γίνει προσπάθεια διαίρεσης με το 0.
  Για την δημιουργία των γραφικών μπορείτε να χρησιμοποιήσετε την βιβλιοθήκη του tkinter.

17. Βιβλιογραφία

[1] http:/ / unixhelp. ed. ac. uk/ CGI/ man-cgi?replace
[2] http:/ / pleac. sourceforge. net/ pleac_python/ index. html
[3] http:/ / www. rosettacode. org/ wiki/ Category:Python
[4] http:/ / www. java2s. com/ Code/ Python/ CatalogPython. htm
[5] http:/ / code. activestate. com/ recipes/ langs/ python/
[6] http:/ / docs. python. org/ dev/ howto/ doanddont. html
[7] http:/ / www. python. org/ doc/ faq/ general/
[8] http:/ / norvig. com/ python-iaq. html
[9] http:/ / dev. fyicenter. com/ Interview-Questions/ Python/ index. html
[10] http:/ / beta. stackoverflow. com/ questions/ tagged/ python
[11] http:/ / www. siafoo. net/ article/ 52
[12] http:/ / ivory. idyll. org/ articles/ advanced-swc/
[13] http:/ / gnosis. cx/ publish/ tech_index_cp. html
[14] http:/ / www. diveintopython. org
[15] http:/ / showmedo. com/ videos/ python
[16] http:/ / youtube. com/ results?search_query=googletechtalks+ python
[17] http:/ / www. awaretek. com/ tutorials. html
[18] http:/ / effbot. org/ zone/
[19] http:/ / groups. google. com/ group/ comp. lang. python. announce/ t/ 37de95ef0326293d
[20] http:/ / pythonpapers. org
[21] http:/ / groups. google. com/ group/ comp. lang. python/ topics
[22] http:/ / planet. python. org
[23] http:/ / peak. telecommunity. com/ DevCenter/ EasyInstall#using-easy-install
[24] http:/ / zetcode. com/ tutorials/ pyqt4/
[25] http:/ / www. qtrac. eu/ pyqtbook. html
[26] http:/ / spe. pycs. net/
[27] http:/ / wxglade. sourceforge. net/
[28] http:/ / zetcode. com/ wxpython/
[29] http:/ / www. pythonware. com/ library/ tkinter/ introduction/
[30] http:/ / www. python. org/ cgi-bin/ moinmoin/ GuiProgramming
[31] http:/ / archive. pythonpapers. org/ ThePythonPapersVolume1Issue1. pdf
[32] http:/ / www. jython. org
[33] http:/ / www. codeplex. com/ Wiki/ View. aspx?ProjectName=IronPython
[34] http:/ / codespeak. net/ pypy/ dist/ pypy/ doc/ home. html
[35] http:/ / www. stackless. com
[36] http:/ / common-lisp. net/ project/ clpython/
[37] http:/ / wiki. mozilla. org/ Tamarin:IronMonkey