Python για Όλους

Εξερευνώντας Δεδομένα με Χρήση της Python 3

Dr. Charles R. Severance

Συντελεστές

Συντακτική υποστήριξη: Elliott Hauser, Sue Blumenberg

Σχεδίαση εξωφύλλου: Aimee Andrion

Μετάφραση

Κιουρτίδου Δ. Κωνσταντία

Ιστορικό Εκτύπωσης

- 2016-Ιουλ-05 Πρώτη ολοκληρωμένη έκδοση σε Python 3.0
- 2015-Δεκ-20 Αρχική, κατά προσέγγιση μετατροπή, σε Python 3.0

Λεπτομέρειες πνευματικών δικαιωμάτων

Πρόλογος

Ανασκευή ενός "Ανοιχτού" Βιβλίου

Είναι πολύ φυσικό για τους ακαδημαϊκούς, στους οποίους λένε συνεχώς «δημοσιεύστε ή χάνετε» να θέλουν να δημιουργούν πάντα κάτι από την αρχή, που να είναι το δικό τους, φρέσκο δημιούργημα. Αυτό το βιβλίο είναι ένα πείραμα στο να μην ξεκινήσω από το μηδέν, αλλά αντίθετα να "αναμείξω" το βιβλίο με τίτλο *Think Python: How to Think Like a Computer Scientist*, που γράφτηκε από τους Allen B. Downey, Jeff Elkner και άλλους.

Τον Δεκέμβριο του 2009, ετοιμαζόμουν να διδάξω *SI502 - Networked Programming* στο Πανεπιστήμιο του Michigan, για πέμπτο συνεχόμενο εξάμηνο και αποφάσισα ότι ήρθε η ώρα να γράψω ένα εγχειρίδιο Python, που να επικεντρωνόταν στην διαχείριση δεδομένων αντί στην κατανόηση αλγορίθμων και στις αφαιρέσεις. Ο στόχος μου, στο SI502, είναι να διδάξω δεξιότητες δια βίου χειρισμού δεδομένων, χρησιμοποιώντας Python. Λίγοι από τους φοιτητές μου σχεδίαζαν να γίνουν επαγγελματίες προγραμματιστές υπολογιστών. Αντίθετα, σχεδίαζαν να γίνουν βιβλιοθηκονόμοι, διευθυντές, δικηγόροι, βιολόγοι, οικονομολόγοι κ.λπ., που έτυχε να θέλουν να χρησιμοποιήσουν επιδέξια την τεχνολογία, στον τομέα που επέλεξαν.

Δεν κατάφερα να βρω το τέλειο βιβλίο Python, με γνώμονα τα δεδομένα του μαθήματός μου, γι' αυτό ξεκίνησα να γράψω ένα τέτοιο βιβλίο. Ευτυχώς σε μια συνεδρίαση της σχολής, τρεις εβδομάδες πριν ξεκινήσω το νέο μου βιβλίο από την αρχή κατά τη διάρκεια των διακοπών, ο Δρ. Atul Prakash μου έδειξε το βιβλίο *Think Python* που είχε χρησιμοποιήσει για να διδάξει το μάθημά του για την Python εκείνο το εξάμηνο. Είναι ένα καλογραμμένο κείμενο Επιστήμης Υπολογιστών με έμφαση σε σύντομες, άμεσες επεξηγήσεις και στη διευκόλυνση της εκμάθησης.

Η συνολική δομή του βιβλίου έχει αλλάξει για να μπορεί κανείς να αντιμετωπίσει προβλήματα ανάλυσης δεδομένων, όσο το δυνατόν γρηγορότερα, και να έχει μια σειρά από παραδείγματα και ασκήσεις σχετικά με την ανάλυση δεδομένων από την αρχή.

Τα κεφάλαια 2–10 είναι παρόμοια με το βιβλίο *Think Python*, αλλά υπήρξαν σημαντικές αλλαγές. Παραδείγματα και ασκήσεις που προσανατολίζονται σε αριθμούς έχουν αντικατασταθεί με ασκήσεις προσανατολισμένες σε δεδομένα. Τα θέματα παρουσιάζονται με τη σειρά που απαιτείται για τη δημιουργία ολοένα και πιο εξελιγμένων λύσεων ανάλυσης δεδομένων. Ορισμένα θέματα όπως το `try` και `except`

μεταφέρθηκαν και παρουσιάζονται ως μέρος του κεφαλαίου της δομής επιλογής. Οι συναρτήσεις αντιμετωπίζονται πολύ επιφανειακά, μέχρι να χρειαστούν για την αντιμετώπιση της πολυπλοκότητας του προγράμματος, αντί να εισαχθούν ως πρώιμο μάθημα αφαίρεσης. Σχεδόν όλες οι συναρτήσεις που καθορίζονται από τον χρήστη έχουν αφαιρεθεί από τον κώδικα των παραδειγμάτων και τις ασκήσεις, εκτός του Κεφαλαίου 4. Η λέξη "recursion (αναδρομή)"[1] δεν εμφανίζεται καθόλου στο βιβλίο.

Στα κεφάλαια 1 και 11–16, όλο το υλικό είναι ολοκαίνουργιο, εστιάζοντας σε πραγματικές χρήσεις και απλά παραδείγματα Python για ανάλυση δεδομένων, συμπεριλαμβανομένων των κανονικών εκφράσεων για αναζήτηση και ανάλυση, αυτοματοποίηση εργασιών στον υπολογιστή σας, ανάκτηση δεδομένων από όλο το δίκτυο. ιστοσυγκομιδή δεδομένων, αντικειμενοστραφή προγραμματισμό, χρήση διαδικτυακών υπηρεσιών, ανάλυση δεδομένων XML και JSON, δημιουργία και χρήση βάσεων δεδομένων με χρήση δομημένης γλώσσας ερωτημάτων και οπτικοποίηση δεδομένων.

Ο απώτερος στόχος όλων αυτών των αλλαγών είναι η στροφή από την Επιστήμη των Υπολογιστών στην Πληροφορική και η συμπερίληψη μόνο θεμάτων μιας πρώτης τάξεως τεχνολογίας, που μπορεί να είναι χρήσιμα ακόμα κι αν κάποιος επιλέξει να μην γίνει επαγγελματίας προγραμματιστής.

Οι μαθητές που βρίσκουν αυτό το βιβλίο ενδιαφέρον και θέλουν να το εξερευνήσουν περαιτέρω θα πρέπει να κοιτάξουν το βιβλίο *Think Python* του Allen B. Downey. Επειδή υπάρχει μεγάλη αλληλοεπικάλυψη μεταξύ των δύο βιβλίων, οι μαθητές θα αποκτήσουν γρήγορα δεξιότητες στους πρόσθετους τομείς του τεχνικού προγραμματισμού και της αλγοριθμικής σκέψης που καλύπτονται στο *Think Python*. Και δεδομένου ότι τα βιβλία έχουν παρόμοιο στυλ γραφής, θα πρέπει να μπορούν να μεταβούν γρήγορα στο *Think Python*, με ελάχιστη προσπάθεια.

Ως κάτοχος πνευματικών δικαιωμάτων του *Think Python*, ο Allen μου έδωσε την άδεια να αλλάξω την άδεια χρήσης του βιβλίου, για το υλικό από το βιβλίο του που παραμένει σε αυτό το βιβλίο, από την άδεια GNU Free Documentation στην πιο πρόσφατη άδεια Creative Commons Attribution — Share Alike. Αυτό ακολουθεί μια γενική αλλαγή στις άδειες ανοιχτής τεκμηρίωσης που μετακινούνται από το GFDL στο CC-BY-SA (π.χ. Wikipedia). Η χρήση της άδειας CC-BY-SA διατηρεί την ισχυρή παράδοση copyleft του βιβλίου, ενώ καθιστά ακόμη πιο απλό για τους νέους συγγραφείς να

[1] Εκτός, φυσικά, από αυτήν τη γραμμή.

επαναχρησιμοποιήσουν αυτό το υλικό, όπως τους βολεύει.

Πιστεύω ότι αυτό το βιβλίο χρησιμεύει ως παράδειγμα του γιατί το ανοιχτό υλικό είναι τόσο σημαντικό για το μέλλον της εκπαίδευσης και θέλω να ευχαριστήσω τους Allen B. Downey και Cambridge University Press για τη καινοτόμα απόφασή τους, να διαθέσουν το βιβλίο με ανοιχτά πνευματικά δικαιώματα . Ελπίζω να είναι ευχαριστημένοι με τα αποτελέσματα των προσπαθειών μου και ελπίζω ότι εσείς, οι αναγνώστες, να είστε ευχαριστημένοι με τις συλλογικές *μας* προσπάθειες.

Θα ήθελα να ευχαριστήσω τους Allen B. Downey και Lauren Cowles για τη βοήθειά τους, την υπομονή και την καθοδήγησή τους στην αντιμετώπιση και επίλυση των ζητημάτων πνευματικών δικαιωμάτων γύρω από αυτό το βιβλίο.

Charles Severance
www.dr-chuck.com
Ann Arbor, MI, USA
9 Σεπτεμβρίου 2013

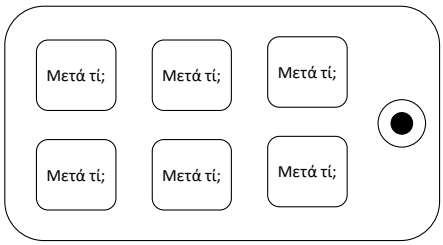
Ο Charles Severance είναι Clinical Associate Professor στο University of Michigan School of Information.

Κεφάλαιο 1

Γιατί πρέπει να μάθετε να γράφετε προγράμματα;

Η συγγραφή προγραμμάτων (ή προγραμματισμός) είναι μια πολύ δημιουργική και ανταποδοτική δραστηριότητα. Μπορείτε να γράψετε προγράμματα για πολλούς λόγους, που κυμαίνονται από το να αποκομίσετε τα προς το ζην έως το να επιλύσετε ένα δύσκολο πρόβλημα ανάλυσης δεδομένων ή να διασκεδάσετε ή να βοηθήσετε κάποιον άλλο να λύσει ένα πρόβλημα. Αυτό το βιβλίο το υποθέτει ότι *όλοι* πρέπει να γνωρίζουν πώς να προγραμματίζουν και ότι μόλις μάθετε πώς να προγραμματίζετε θα καταλάβετε τι θέλετε να κάνετε με τη νέα σας δεξιότητα.

Είμαστε περιτριγυρισμένοι στην καθημερινότητά μας με υπολογιστές που κυμαίνονται από φορητούς υπολογιστές έως κινητά τηλέφωνα. Μπορούμε να σκεφτούμε αυτούς τους υπολογιστές ως τους «προσωπικούς βοηθούς» μας που μπορούν να φροντίσουν πολλά πράγματα για λογαριασμό μας. Το υλικό στους σημερινούς υπολογιστές μας είναι ουσιαστικά κατασκευασμένο για να μας θέτει συνεχώς την ερώτηση "Τι θα θέλατε να κάνω στη συνέχεια;"



Εικόνα 1.1: Προσωπικός Ψηφιακός Βοηθός

Οι προγραμματιστές προσθέτουν ένα λειτουργικό σύστημα και ένα σύνολο εφαρμογών στο υλικό και καταλήγουμε σε έναν Προσωπικό Ψηφιακό Βοηθό, που είναι αρκετά χρήσιμος και ικανός να μας βοηθήσει να κάνουμε πολλά διαφορετικά πράγματα.

Οι υπολογιστές μας είναι γρήγοροι και έχουν τεράστια ποσότητα μνήμης. Θα μπορούσαν να μας βοηθήσουν πολύ αν, μόνο, γνωρίζαμε τη γλώσσα στην οποία θα έπρεπε να μιλήσουμε, για να εξηγήσουμε στον υπολογιστή τι θα θέλαμε να «κάνει στη συνέχεια». Αν γνωρίζαμε αυτή τη γλώσσα, θα μπορούσαμε να πούμε στον υπολογιστή να κάνει διάφορες επαναλαμβανόμενες εργασίες για λογαριασμό μας. Είναι ενδιαφέρον ότι τα πράγματα που μπορούν να κάνουν οι υπολογιστές είναι συχνά τα πράγματα που

εμείς οι άνθρωποι θεωρούμε βαρετά και μπερδεμένα.

Για παράδειγμα, κοιτάξτε τις τρεις πρώτες παραγράφους αυτού του κεφαλαίου και πείτε μου τη λέξη που χρησιμοποιείται περισσότερο και πόσες φορές χρησιμοποιείται η λέξη αυτή. Ενώ μπορούσατε να διαβάσετε και να καταλάβετε τις λέξεις σε λίγα δευτερόλεπτα, το να τις μετρήσετε είναι σχεδόν επώδυνο γιατί αυτό δεν είναι το είδος των προβλημάτων, που έχει σχεδιαστεί ο ανθρώπινος νους για να λύνει . Για έναν υπολογιστή, ισχύει το αντίθετο. Η ανάγνωση και η κατανόηση κειμένου από ένα κομμάτι χαρτί είναι δύσκολο για έναν υπολογιστή, αλλά το να μετράει τις λέξεις και να σας λέει πόσες φορές χρησιμοποιήθηκε η πιο συχνά επαναλαμβανόμενη λέξη, είναι πολύ εύκολο για τον υπολογιστή:

```
python words.py
Εισάγετε αρχείο: words.txt
to 16
```

Ο "προσωπικός βοηθός ανάλυσης πληροφοριών" μας είπε γρήγορα ότι η λέξη "to" χρησιμοποιήθηκε δεκαέξι φορές στις τρεις πρώτες παραγράφους αυτού του κεφαλαίου. (Προφανώς στην αγγλική έκδοση του βιβλίου αυτού)

Αυτό ακριβώς το γεγονός, ότι δηλαδή οι υπολογιστές είναι καλοί σε πράγματα στα οποία δεν είναι οι άνθρωποι, είναι και ο λόγος που πρέπει να ειδικευτείτε στην ομιλία "γλώσσας υπολογιστών". Μόλις μάθετε αυτήν τη νέα γλώσσα, μπορείτε να αναθέσετε καθημερινές, τετριμμένες εργασίες στον σύντροφό σας (τον υπολογιστή), εξοικονομώντας χρόνο για τα πράγματα που σας ταιριάζουν και αγαπάτε. Εσείς συνεισφέρετε σε δημιουργικότητα, διαίσθηση και εφευρετικότητα σε αυτήν τη συνεργασία.

Δημιουργικότητα και κίνητρο

Παρόλο που αυτό το βιβλίο δεν προορίζεται για επαγγελματίες προγραμματιστές, ο επαγγελματικός προγραμματισμός μπορεί να είναι μια πολύ ανταποδοτική δουλειά τόσο οικονομικά όσο και προσωπικά. Η δημιουργία χρήσιμων, κομψών και έξυπνων προγραμμάτων για χρήση από άλλους είναι μια πολύ δημιουργική δραστηριότητα. Ο υπολογιστής σας ή ο προσωπικός ψηφιακός βοηθός (PDA) σας, συνήθως περιέχει πολλά διαφορετικά προγράμματα από πολλές διαφορετικές ομάδες προγραμματιστών, που το καθένα ανταγωνίζεται για την προσοχή και το ενδιαφέρον σας. Προσπαθούν με τον καλύτερο δυνατό τρόπο να καλύψουν τις ανάγκες σας και να σας προσφέρουν μια

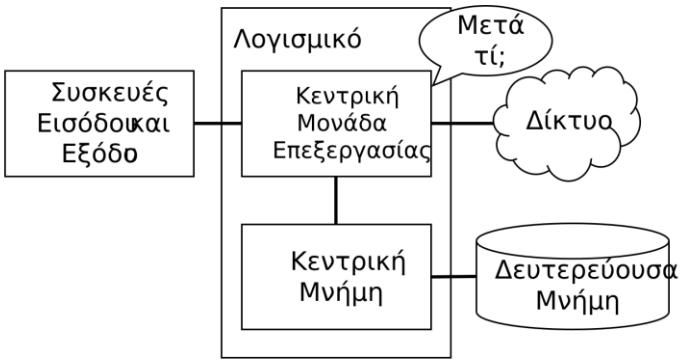
εξαιρετική εμπειρία χρήσης. Σε ορισμένες περιπτώσεις, όταν επιλέγετε ένα κομμάτι λογισμικού, οι προγραμματιστές αποζημιώνονται άμεσα λόγω της επιλογής σας.

Αν σκεφτούμε τα προγράμματα ως τη δημιουργική παραγωγή ομάδων προγραμματιστών, ίσως το παρακάτω σχήμα να είναι μια πιο λογική εκδοχή του PDA μας:

Εικόνα 1.2: Οι προγραμματιστές σας μιλάνε

Προς το παρόν, το κύριο κίνητρό μας δεν είναι να κερδίσουμε χρήματα ή να ευχαριστήσουμε τους τελικούς χρήστες, αλλά αντίθετα να είμαστε πιο παραγωγικοί στο χειρισμό των δεδομένων και των πληροφοριών που θα συναντήσουμε στην καθημερινή μας ζωή. Όταν ξεκινάτε για πρώτη φορά, θα είστε και ο προγραμματιστής και ο τελικός χρήστης των προγραμμάτων σας. Καθώς αποκτάτε δεξιότητες ως προγραμματιστής και ο προγραμματισμός σας φαίνεται πιο δημιουργικός, οι σκέψεις σας μπορεί να στραφούν στην ανάπτυξη προγραμμάτων για άλλους.

Αρχιτεκτονική υλικού υπολογιστών



Εικόνα 1.3: Αρχιτεκτονική Υλικού

Πριν ξεκινήσουμε να μαθαίνουμε τη γλώσσα που πρέπει να μιλάμε για να δίνουμε οδηγίες στους υπολογιστές για την ανάπτυξη λογισμικού, πρέπει να μάθουμε λίγα πράγματα για τον τρόπο κατασκευής των υπολογιστών. Αν αποσυναρμολογούσατε τον

υπολογιστή ή το κινητό σας τηλέφωνο και κοιτούσατε βαθιά μέσα του, θα βρίσκατε τα ακόλουθα μέρη:

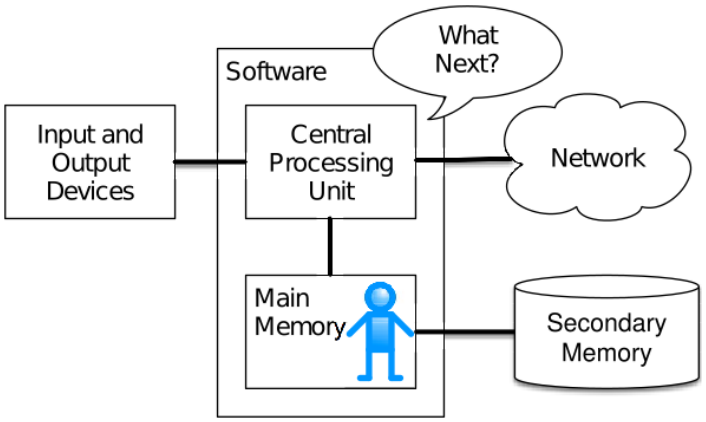
Οι ορισμοί υψηλού επιπέδου αυτών των τμημάτων είναι οι εξής:

- Η *Κεντρική Μονάδα Επεξεργασίας* (ή CPU) είναι το τμήμα του υπολογιστή που έχει φτιαχτεί για να έχει εμμονή με το "και μετά τί;" Εάν ο υπολογιστής σας έχει χρονιστεί στα 3,0 Gigahertz, αυτό σημαίνει ότι η CPU θα ρωτήσει "μετά τί;" τρία δισεκατομμύρια φορές το δευτερόλεπτο. Θα πρέπει να μάθετε πώς να μιλάτε γρήγορα για να συμβαδίσετε με την CPU.
- Η *Κύρια Μνήμη* χρησιμοποιείται για την αποθήκευση πληροφοριών που χρειάζεται η CPU, γρήγορα. Η κύρια μνήμη είναι σχεδόν τόσο γρήγορη όσο η CPU. Αλλά οι πληροφορίες που είναι αποθηκευμένες στην κύρια μνήμη εξαφανίζονται όταν ο υπολογιστής είναι απενεργοποιημένος.
- Η *Δευτερεύουσα Μνήμη* χρησιμοποιείται επίσης για την αποθήκευση πληροφοριών, αλλά είναι πολύ πιο αργή από την κύρια μνήμη. Το πλεονέκτημα της δευτερεύουσας μνήμης είναι ότι μπορεί να κρατήσει αποθηκευμένες τις πληροφορίες ακόμη και όταν σταματά η τροφοδοσία ρεύματος στον υπολογιστή. Παραδείγματα δευτερεύουσας μνήμης είναι οι μονάδες δίσκου ή μνήμη flash (συνήθως συναντώνται σε USB sticks και φορητές συσκευές αναπαραγωγής μουσικής).
- Οι συσκευές *Εισόδου και Εξόδου* είναι απλώς η οθόνη, το πληκτρολόγιο, το ποντίκι, το μικρόφωνο, το ηχείο, η επιφάνεια αφής κλπ. Είναι όλοι οι τρόποι αλληλεπίδρασης με τον υπολογιστή.
- Στις μέρες μας, οι περισσότεροι υπολογιστές διαθέτουν επίσης *Σύνδεση Δικτύου* για ανάκτηση πληροφοριών μέσω δικτύου. Μπορούμε να σκεφτόμαστε το δίκτυο ως ένα πολύ αργό μέρος για την αποθήκευση και την ανάκτηση δεδομένων, που μπορεί να μην είναι πάντα "up", δηλαδή διαθέσιμο. Κατά κάποιον τρόπο, το δίκτυο είναι μια πιο αργή και μερικές φορές αναξιόπιστη μορφή *Δευτερεύουσας Μνήμης*.

Ενώ οι περισσότερες λεπτομέρειες, για το πώς λειτουργούν αυτά τα εξαρτήματα είναι καλύτερα να αφεθούν στους κατασκευαστές υπολογιστών, βοηθά να γνωρίζουμε την ορολογία, ώστε να μπορούμε να μιλάμε για αυτά τα βασικά κομμάτια του υπολογιστή καθώς προχωράμε στη συγγραφή των προγράμματά μας.

Ως προγραμματιστές, η δουλειά σας είναι να χρησιμοποιήσετε και να ενορχηστρώσετε καθέναν από αυτούς τους πόρους, για να λύσετε το πρόβλημα που χρειάζεται να

λύσετε, και για να αναλύσετε τα δεδομένα, που λαμβάνετε από τη λύση. Ως προγραμματιστές θα "μιλάτε" κυρίως με την CPU και θα της λέτε τι πρέπει να κάνει στη συνέχεια. Μερικές φορές θα πείτε στην CPU να χρησιμοποιήσει την κύρια μνήμη, τη δευτερεύουσα μνήμη, το δίκτυο ή τις συσκευές εισόδου/εξόδου.



Εικόνα 1.4: Πού Βρίσκεστε;

Είστε το άτομο που πρέπει να απαντά στη CPU, στην ερώτηση "Μετά τί;". Αλλά θα ήταν κάπως άβολο αν έπρεπε να σας συρρικνώσουμε, σε ύψος 5 χιλιοστών, και να σας εισάγουμε στον υπολογιστή, μόνο και μόνο για να μπορείτε να δίνετε μια εντολή, τρεις δισεκατομμύρια φορές το δευτερόλεπτο. Επομένως, πρέπει να γράψετε τις οδηγίες σας εκ των προτέρων. Αυτές τις αποθηκευμένες οδηγίες τις ονομάζουμε *πρόγραμμα* και την πράξη της καταγραφής αυτών των οδηγιών και την διασφάλιση της ορθότητας αυτών *προγραμματισμό*.

Κατανόηση του προγραμματισμού

Στο υπόλοιπο αυτού του βιβλίου, θα προσπαθήσουμε να σας μετατρέψουμε σε ένα άτομο, εξειδικευμένο στην τέχνη του προγραμματισμού. Στο τέλος θα είστε *προγραμματιστής* - ίσως όχι επαγγελματίας προγραμματιστής, αλλά τουλάχιστον θα έχετε τις δεξιότητες να εξετάσετε ένα πρόβλημα ανάλυσης δεδομένων/πληροφοριών και να αναπτύξετε ένα πρόγραμμα για την επίλυση του προβλήματος.

Στην ουσία, χρειάζεστε δύο δεξιότητες για να είστε προγραμματιστής:

- Πρώτον, πρέπει να γνωρίζετε τη γλώσσα προγραμματισμού (Python) - πρέπει να γνωρίζετε το λεξιλόγιο και τη γραμματική της. Πρέπει να είστε σε θέση να γράψετε σωστά τις λέξεις, σε αυτήν τη νέα γλώσσα, και να ξέρετε πώς να δημιουργήσετε καλά σχηματισμένες "προτάσεις" σε αυτήν.
- Δεύτερον, πρέπει να είστε σε θέση να "πείτε μια ιστορία". Γράφοντας μια ιστορία,

συνδυάζετε λέξεις και προτάσεις, για να μεταφέρετε μια ιδέα στον αναγνώστη. Απαιτείται μια ικανότητα και τέχνη στην κατασκευή της ιστορίας και η ικανότητα στη συγγραφή ιστοριών βελτιώνεται με το να γράφουμε και να λαμβάνουμε κάποια ανατροφοδότηση. Στον προγραμματισμό, το πρόγραμμά μας είναι η «ιστορία» και το πρόβλημα που προσπαθείτε να λύσετε είναι η «ιδέα».

Μόλις μάθετε μία γλώσσα προγραμματισμού, όπως η Python, θα είναι πολύ πιο εύκολο να μάθετε μια δεύτερη γλώσσα, όπως η JavaScript ή η C ++. Η νέα γλώσσα προγραμματισμού θα έχει πολύ διαφορετικό λεξιλόγιο και γραμματική, αλλά οι δεξιότητες επίλυσης προβλημάτων που απαιτούνται είναι οι ίδιες, σε όλες τις γλώσσες προγραμματισμού.

Θα μάθετε το "λεξιλόγιο" και τις "προτάσεις" της Python αρκετά γρήγορα. Θα χρειαστεί περισσότερος χρόνος για να μπορέσετε να γράψετε ένα πρόγραμμα με συνοχή, για την επίλυση ενός ολοκαίνουργιου προβλήματος. Διδάσκουμε προγραμματισμό όπως και τη γραφή. Αρχίζουμε να διαβάζουμε και να εξηγούμε προγράμματα, μετά γράφουμε απλά προγράμματα και μετά γράφουμε όλο και πιο πολύπλοκα προγράμματα με την πάροδο του χρόνου. Κάποια στιγμή "βρίσκετε τη μούσα σας" και βλέπετε τα μοτίβα μόνοι σας και μπορείτε να δείτε πιο φυσικά πώς να αντιμετωπίσετε ένα πρόβλημα και να γράψετε ένα πρόγραμμα που να το λύνει. Και μόλις φτάσετε σε αυτό το σημείο, ο προγραμματισμός γίνεται μια πολύ ευχάριστη και δημιουργική διαδικασία.

Ξεκινάμε με το λεξιλόγιο και τη δομή των προγραμμάτων Python. Κάντε υπομονή καθώς τα απλά παραδείγματα θα σας θυμίζουν την εποχή που ξεκινήσατε να διαβάζετε για πρώτη φορά.

Λέξεις και προτάσεις

Σε αντίθεση με τις ανθρώπινες γλώσσες, το λεξιλόγιο της Python είναι πραγματικά πολύ μικρό. Αυτό το «λεξιλόγιο» το αποκαλούμε «δεσμευμένες λέξεις». Αυτές είναι λέξεις που έχουν πολύ ιδιαίτερη σημασία για την Python. Όταν η Python βλέπει αυτές τις λέξεις σε ένα πρόγραμμα Python, έχουν μία και μοναδική σημασία. Αργότερα καθώς γράφετε προγράμματα θα φτιάξετε τις δικές σας λέξεις, που έχουν νόημα για εσάς και ονομάζονται *μεταβλητές*. Θα έχετε ένα μεγάλο εύρος επιλογών για την ονοματολογία των μεταβλητών σας, αλλά δεν μπορείτε να χρησιμοποιήσετε καμία από τις δεσμευμένες λέξεις της Python, ως όνομα μεταβλητής.

Όταν εκπαιδεύουμε έναν σκύλο, χρησιμοποιούμε ειδικές λέξεις όπως "κάθισε", "μείνε"

και "φέρε". Όταν μιλάτε σε ένα σκυλί και δεν χρησιμοποιείτε καμία από αυτές τις δεσμευμένες λέξεις, απλώς σας κοιτά με ένα ερωτηματικό βλέμμα στο πρόσωπό του μέχρι να πείτε μια δεσμευμένη λέξη. Για παράδειγμα, αν πείτε: "Μακάρι να περπατούσαν περισσότερο οι άνθρωποι, για να βελτιώσουν την υγεία τους", αυτό που πιθανότατα ακούνε τα περισσότερα σκυλιά είναι "μπλα μπλα μπλα *περπάτα* μπλα μπλα μπλα μπλα." Αυτό συμβαίνει επειδή το "περπάτημα" είναι μια δεσμευμένη λέξη στη γλώσσα του σκύλου. Πολλοί μπορεί να αντιτείνουν ότι η γλώσσα μεταξύ ανθρώπων και γατών δεν έχει δεσμευμένες λέξεις[1] .

Κάποιες από τις δεσμευμένες λέξεις στη γλώσσα, που οι άνθρωποι μιλούν με την Python, είναι και οι ακόλουθες:

```
and       del       global      not       with
as        elif      if          or        yield
assert    else      import      pass
break     except    in          raise
class     finally   is          return
continue  for       lambda      try
def       from      nonlocal    while
```

Πράγματι, και σε αντίθεση με έναν σκύλο, η Python είναι ήδη πλήρως εκπαιδευμένη. Όταν λέτε "try", η Python θα δοκιμάζει, κάθε φορά που το λέτε, χωρίς αποτυχία.

Θα μάθουμε αυτές τις δεσμευμένες λέξεις και πώς χρησιμοποιούνται έγκυρα, αλλά προς το παρόν θα επικεντρωθούμε στο ισοδύναμο, της Python, του "μιλάω" (στη γλώσσα ανθρώπου-σε-σκύλο). Το ωραίο, όταν λέμε στην Python να μιλήσει, είναι ότι μπορούμε ακόμη να της πούμε τι να πει, δίνοντάς της ένα μήνυμα σε εισαγωγικά:

```
print('Γειά σου κόσμε!')
```

Και, μόλις, γράψαμε την πρώτη μας συντακτικά σωστή πρόταση, στην Python. Η πρόταση μας ξεκινά με τη συνάρτηση *print* ακολουθούμενη από μια σειρά κειμένου της επιλογής μας που περικλείεται σε απλά εισαγωγικά. Οι συμβολοσειρές στις εντολές εκτύπωσης περικλείονται σε εισαγωγικά. Τα απλά εισαγωγικά και τα διπλά εισαγωγικά είναι ισοδύναμα. Οι περισσότεροι χρησιμοποιούν απλά εισαγωγικά, εκτός από περιπτώσεις όπου ένα μόνο εισαγωγικό (το οποίο είναι μπορεί να δηλώνει και "απόστροφο") πρέπει να εμφανιστεί στη συμβολοσειρά.

[1] http://xkcd.com/231/

Συνομιλία με την Python

ώρα που μάθαμε μια λέξη και μια απλή πρόταση, στην Python, πρέπει να γνωρίζουμε και πώς να ξεκινήσουμε μια συνομιλία με την Python, προκειμένου να δοκιμάσουμε τις νέες γλωσσικές μας δεξιότητες.

Για να καταφέρετε να συνομιλήσετε με την Python, πρέπει πρώτα να εγκαταστήσετε το λογισμικό Python στον υπολογιστή σας και να μάθετε πώς να εκκινείτε την Python στον υπολογιστή σας. Αυτό απαιτεί πάρα πολλές λεπτομερές για να συμπεριληφθεί σε αυτό το κεφάλαιο, οπότε προτείνω να συμβουλευτείτε το www.gr.py4e.com όπου σας παρέχω λεπτομερείς οδηγίες και βίντεο, για τη ρύθμιση και την εκκίνηση της Python σε συστήματα Macintosh και Windows. Έτσι, κάποια στιγμή, σε ένα τερματικό ή στο παράθυρο εντολών θα πληκτρολογήσετε *python* και ο διερμηνέας Python θα αρχίσει να εκτελείται σε διαδραστική λειτουργία, οπότε θα δείτε κάτι όπως το εξής:

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:54:25)
[MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Η προτροπή >>> είναι ο τρόπος του διερμηνέα Python να σας ρωτήσει: "Τι θέλετε να κάνω στη συνέχεια;" Η Python είναι έτοιμη να συζητήσει μαζί σας. Το μόνο που πρέπει να γνωρίζετε είναι πώς να της μιλήσετε, στη γλώσσα Python.

Ας πούμε για παράδειγμα, ότι δεν γνωρίζατε ούτε τις πιο απλές λέξεις ή προτάσεις της γλώσσας Python. Μπορεί να θέλετε να χρησιμοποιήσετε την κλασσική πρόταση, που χρησιμοποιούν οι αστροναύτες όταν προσγειώνονται σε έναν μακρινό πλανήτη και προσπαθούν να μιλήσουν με τους κατοίκους του πλανήτη:

```
>>> Ερχόμαστε ειρηνικά, παρακαλώ πηγαίνετέ μας στον αρχηγό σας
File "<stdin>", line 1
  Ερχόμαστε ειρηνικά, παρακαλώ πηγαίνετέ μας στον αρχηγό σας
       ^
SyntaxError: invalid syntax
>>>
```

Αυτό δεν πήγε και τόσο καλά. Αν δεν σκεφτείτε κάτι γρήγορα, οι κάτοικοι του πλανήτη είναι πιθανό να σας επιτεθούν με τα δόρατά τους, να σας βάλουν στη σούβλα, να σας ψήσουν στη φωτιά και να σας φάνε για δείπνο.

Ευτυχώς είχατε ένα αντίγραφο αυτού του βιβλίου, μαζί σας, στο ταξίδι σας και

ανοίγοντάς το βρεθήκατε σε αυτήν τη σελίδα, οπότε προσπαθείτε ξανά:

```
>>> print('Γειά σου κόσμε!')
Γειά σου κόσμε!
```

Αυτό λειτούργησε πολύ καλύτερα, οπότε προσπαθείτε να επικοινωνήσετε περισσότερο:

```
>>> print('Πρέπει να είστε οι θρυλικοί θεοί που έρχεστε από τον ουρανό')
Πρέπει να είστε οι θρυλικοί θεοί που έρχεστε από τον ουρανό
>>> print('Σας περιμέναμε πολύ καιρό')
Σας περιμέναμε πολύ καιρό
>>> print('Ο μύθος μας λέει ότι θα είστε πολύ νόστιμοι με μουστάρδα')
Ο μύθος μας λέει ότι θα είστε πολύ νόστιμοι με μουστάρδα
>>> print 'Θα έχουμε συμπόσιο απόψε, εκτός κι αν το πείτε
File "<stdin>", line 1
  print 'Θα έχουμε συμπόσιο απόψε, εκτός κι αν το πείτε
                                                      ^
SyntaxError: Missing parentheses in call to 'print'
>>>
```

Η συζήτηση πήγαινε τόσο καλά μέχρι που κάνατε ένα μικρό λαθάκι, χρησιμοποιώντας τη γλώσσα Python, και η Python ξανά έβγαλε τα δόρατα.

Σε αυτό το σημείο, θα πρέπει επίσης να συνειδητοποιήσετε ότι, ενώ η Python είναι εκπληκτικά πολύπλοκη και ισχυρή και πολύ επιλεκτική σχετικά με τη σύνταξη που χρησιμοποιείτε για να επικοινωνήσετε μαζί της, η Python *δεν* είναι έξυπνη. Στην πραγματικότητα συνομιλείτε με τον εαυτό σας, αλλά χρησιμοποιείτε σωστή σύνταξη.

Κατά μία έννοια, όταν χρησιμοποιείτε ένα πρόγραμμα γραμμένο από κάποιον άλλο, η συζήτηση γίνεται μεταξύ εσάς και εκείνου του άλλου προγραμματιστή, με την Python να ενεργεί ως ενδιάμεσος. Η Python είναι ένας τρόπος, για τους δημιουργούς προγραμμάτων, να εκφράσουν πώς υποτίθεται ότι θα προχωρήσει η συνομιλία. Και σε λίγα ακόμη κεφάλαια, θα είστε ένας από αυτούς τους προγραμματιστές, που χρησιμοποιούν την Python για να μιλήσουν στους χρήστες του προγράμματός τους.

Πριν αφήσουμε την πρώτη μας συνομιλία με τον διερμηνέα της Python, μάλλον θα πρέπει να μάθετε τον καθώς πρέπει τρόπο για να πείτε "αντίο" όταν αλληλεπιδράτε με τους κατοίκους του Πλανήτη Python:

```
>>> αντίο
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
```

```
NameError: name 'αντίο' is not defined
>>> if you don't mind, I need to leave
File "<stdin>", line 1
  if you don't mind, I need to leave
           ^
SyntaxError: invalid syntax
>>> quit()
```

Θα παρατηρήσετε ότι το σφάλμα είναι διαφορετικό για τις δύο πρώτες εσφαλμένες προσπάθειες. Το δεύτερο σφάλμα είναι διαφορετικό γιατί το *if* είναι μια δεσμευμένη λέξη και η Python είδε την δεσμευμένη λέξη και σκέφτηκε ότι προσπαθούσαμε να πούμε κάτι, αλλά αντιλήφθηκε λάθος στη σύνταξη της πρότασης.

Ο σωστός τρόπος για να πείτε "αντίο" στην Python είναι να πληκτρολογήσετε *quit()* στη διαδραστική ερώτηση >>>. Πιθανότατα θα σας έπαιρνε αρκετή ώρα για να το μαντέψετε, οπότε το να έχετε ένα βιβλίο κοντά σας πιθανότατα θα σας φανεί χρήσιμο.

Ορολογία: Διερμηνέας και μεταγλωττιστής

Η Python είναι μια γλώσσα *υψηλού επιπέδου*, κατασκευασμένη ώστε να είναι σχετικά απλό για τους ανθρώπους να διαβάζουν και να γράφουν και για τους υπολογιστές να διαβάζουν και να επεξεργάζονται. Άλλες γλώσσες υψηλού επιπέδου είναι οι Java, C ++, PHP, Ruby, Basic, Perl, JavaScript και πολλές άλλες. Το πραγματικό υλικό μέσα στην κεντρική μονάδα επεξεργασίας (CPU) δεν καταλαβαίνει καμία από αυτές τις γλώσσες υψηλού επιπέδου.

Η CPU καταλαβαίνει μια γλώσσα που ονομάζουμε *γλώσσα μηχανής*. Η γλώσσα της μηχανής είναι πολύ απλή και, ειλικρινά, πολύ κουραστική για να γραφτεί, επειδή τα αναπαριστά όλα με μηδενικά και μονάδες:

```
00101000111010010010101010000001111
11100110000011101010010101101101
...
```

Η γλώσσα της μηχανής φαίνεται, επιφανειακά, αρκετά απλή, δεδομένου ότι υπάρχουν μόνο μηδενικά και μονάδες, αλλά η σύνταξή της είναι ακόμη πιο σύνθετη και πιο πολύπλοκη από της Python. Έτσι, πολύ λίγοι προγραμματιστές γράφουν, ενίοτε, σε γλώσσα μηχανής. Αντ 'αυτού, κατασκευάζουμε διάφορους μεταφραστές που επιτρέπουν στους προγραμματιστές να γράφουν σε γλώσσες υψηλού επιπέδου, όπως η

Python ή η JavaScript και αυτοί οι μεταφραστές μετατρέπουν τα προγράμματα σε γλώσσα μηχανής για να εκτελεστεί πραγματικά από την CPU.

Δεδομένου ότι η γλώσσα μηχανής είναι συνδεδεμένη με το υλικό του υπολογιστή, η γλώσσα μηχανής δεν είναι *φορητή*, σε διαφορετικούς τύπους υλικού. Προγράμματα γραμμένα σε γλώσσες υψηλού επιπέδου μπορούν να μεταφερθούν μεταξύ διαφορετικών υπολογιστών χρησιμοποιώντας διαφορετικό διερμηνέα στο νέο μηχάνημα ή επαναμεταφράζοντας τον κώδικα για να δημιουργηθεί μια έκδοση του προγράμματος σε γλώσσας μηχανής για το νέο μηχάνημα.

Αυτοί οι μεταφραστές γλωσσών προγραμματισμού εμπίπτουν σε δύο γενικές κατηγορίες: (1) διερμηνείς και (2) μεταγλωττιστές.

Ένας *διερμηνέας* διαβάζει τον πηγαίο κώδικα του προγράμματος, όπως έχει γραφτεί από τον προγραμματιστή, αναλύει τον πηγαίο κώδικα και ερμηνεύει τις οδηγίες εν κινήσει. Η Python είναι ένας διερμηνέας και όταν τρέχουμε την Python διαδραστικά, μπορούμε να πληκτρολογήσουμε μια γραμμή Python (μια πρόταση), η Python την επεξεργάζεται αμέσως και είναι έτοιμη για να πληκτρολογήσουμε την επόμενη γραμμή Python.

Μερικές από τις γραμμές του κώδικα λένε στην Python ότι θέλετε να θυμάται κάποια τιμή για αργότερα. Πρέπει να επιλέξουμε ένα όνομα για να απομνημονευθεί αυτή η τιμή και μπορούμε να χρησιμοποιήσουμε αυτό το συμβολικό όνομα για να ανακτήσουμε την τιμή αργότερα. Χρησιμοποιούμε τον όρο *μεταβλητή* για να αναφερθούμε στα ονόματα που χρησιμοποιούμε, για να χειριστούμε αυτά τα αποθηκευμένα δεδομένα.

```
>>> x = 6
>>> print(x)
6
>>> y = x * 7
>>> print(y)
42
>>>
```

Σε αυτό το παράδειγμα, ζητάμε από την Python να θυμάται την τιμή έξι και να χρησιμοποιήσει το όνομα *x*, ώστε να μπορούμε να ανακτήσουμε την τιμή αργότερα. Επαληθεύουμε ότι η Python έχει κρατήσει την τιμή, χρησιμοποιώντας το *print*. Στη συνέχεια, ζητάμε από την Python να ανακτήσει το *x* και να το πολλαπλασιάσει επί επτά και να θέσει την νέα τιμή που υπολογίστηκε στο *y*. Στη συνέχεια, ζητάμε από την Python να εκτυπώσει την τρέχουσα τιμή του *y*.

Παρόλο που πληκτρολογούμε αυτές τις εντολές σε Python μία γραμμή τη φορά, η Python τις αντιμετωπίζει ως μια διατεταγμένη σειρά δηλώσεων με μεταγενέστερες δηλώσεις που μπορούν να ανακτήσουν δεδομένα που δημιουργήθηκαν σε προγενέστερες δηλώσεις. Γράφουμε την πρώτη μας απλή παράγραφο με τέσσερις προτάσεις με λογική και ουσιαστική σειρά.

Είναι στη φύση ενός *διερμηνέα* να μπορεί να έχει μια διαδραστική συνομιλία όπως φαίνεται παραπάνω. Σε έναν *μεταγλωττιστής* πρέπει να παραδοθεί ολόκληρο το πρόγραμμα, σε ένα αρχείο και στη συνέχεια εκτελεί μια διαδικασία για τη μετάφραση του πηγαίου κώδικα υψηλού επιπέδου σε γλώσσα μηχανής. Στη συνέχεια, ο μεταγλωττιστής αποθηκεύει την γλώσσα μηχανής, που προκύπτει σε ένα αρχείο για μετέπειτα εκτέλεση.

Εάν διαθέτετε σύστημα Windows, συχνά αυτά τα εκτελέσιμα προγράμματα γλώσσας μηχανής έχουν ένα επίθημα ".exe" ή ".dll", που σημαίνει "εκτελέσιμο" και "dynamic link library / βιβλιοθήκη δυναμικών συνδέσμων" αντίστοιχα. Σε Linux και Macintosh, δεν υπάρχει επίθημα που να χαρακτηρίζει μοναδικά ένα αρχείο ως εκτελέσιμο.

Εάν ανοίγατε ένα εκτελέσιμο αρχείο σε έναν επεξεργαστή κειμένου, θα φαινόταν εντελώς τρελό και δεν θα μπορούσε να διαβαστεί:

```
^?ELF^A^A^A^@^@^@^@^@^@^@^@^@^B^@^C^@^A^@^@^@\xa0\x82
^D^H4^@^@^@\x90^]^@^@^@^@^@^@4^@ ^@^G^@(^@$^@!^@^F^@
^@^@4^@^@^@4\x80^D^H4\x80^D^H\xe0^@^@^@\xe0^@^@^@^E
^@^@^@^D^@^@^@^C^@^@^@^T^A^@^@^T\x81^D^H^T\x81^D^H^S
^@^@^@^S^@^@^@^D^@^@^@^A^@^@^@^A\^D^HQVhT\x83^D^H\xe8
....
```

Δεν είναι εύκολο να διαβάσετε ή να γράψετε σε γλώσσα μηχανής, οπότε είναι ωραίο να έχουμε *διερμηνείς* και *μεταγλωττιστές*, που μας επιτρέπουν να γράφουμε σε γλώσσες υψηλού επιπέδου, όπως η Python ή η C.

Τώρα, σε αυτό το σημείο στη συζήτησή μας για μεταγλωττιστές και διερμηνείς, θα πρέπει να αναρωτηθείτε λίγο για τον ίδιο τον διερμηνέα της Python. Σε ποια γλώσσα γράφεται; Είναι γραμμένος σε μεταγλωττισμένη γλώσσα; Όταν πληκτρολογούμε "python", τι ακριβώς συμβαίνει;

Ο διερμηνέας της Python είναι γραμμένος σε μια γλώσσα υψηλού επιπέδου που ονομάζεται "C". Μπορείτε να δείτε τον πραγματικό πηγαίο κώδικα για τον διερμηνέα της Python πηγαίνοντας στο www.python.org και βρίσκοντας τη διαδρομή προς τον πηγαίο κώδικα. Έτσι η Python είναι ένα πρόγραμμα που είναι μεταγλωττισμένο σε

γλώσσα μηχανής. Όταν εγκαταστήσατε την Python στον υπολογιστή σας (ή ο προμηθευτής την εγκατέστησε), αντιγράψατε ένα αντίγραφο κώδικα μηχανής του μεταφρασμένου προγράμματος Python στο σύστημά σας. Στα Windows, ο εκτελέσιμος κώδικας μηχανής για την ίδια την Python είναι πιθανό σε ένα αρχείο με όνομα όπως:

```
C:\Python35\python.exe
```

Αυτά είναι περισσότερα από ό,τι πραγματικά πρέπει να γνωρίζετε για να είστε προγραμματιστής Python, αλλά μερικές φορές αξίζει να απαντήσετε σε αυτές τις μικρές ενοχλητικές ερωτήσεις στην αρχή.

Γράφοντας ένα πρόγραμμα

Η πληκτρολόγηση εντολών στον διερμηνέα της Python είναι ένας πολύ καλός τρόπος για να πειραματιστείτε με τις δυνατότητες της Python, αλλά δεν συνιστάται για την επίλυση πιο πολύπλοκων προβλημάτων.

Όταν θέλουμε να γράψουμε ένα πρόγραμμα, χρησιμοποιούμε έναν συντάκτη κειμένου για να γράψουμε τις εντολές Python σε ένα αρχείο, το οποίο ονομάζεται *script / σενάριο*. Κατά συνθήκη, τα σενάρια Python έχουν ονόματα που τελειώνουν με `.py`.

Για να εκτελέσετε το σενάριο, πρέπει να πείτε στον διερμηνέα της Python το όνομα του αρχείου. Στο τερματικό / γραμμή εντολών, πληκτρολογείτε `python hello.py` ως εξής:

```
$ cat hello.py
print('Γειά σου κόσμε!')
$ python hello.py
Γειά σου κόσμε!
```

Το "$" είναι η προτροπή του λειτουργικού συστήματος και το "cat hello.py" μας δείχνει ότι το αρχείο "hello.py" περιέχει ένα πρόγραμμα Python μιας γραμμής, για την εκτύπωση μιας συμβολοσειράς.

Καλούμε τον διερμηνέα Python και του λέμε να διαβάσει τον πηγαίο κώδικα από το αρχείο "hello.py" αντί να μας ζητήσει τις γραμμές του κώδικα Python διαδραστικά.

Θα παρατηρήσετε ότι δεν ήταν ανάγκη να γράψετε *quit()* στο τέλος του προγράμματος Python στο αρχείο. Όταν η Python διαβάζει τον πηγαίο κώδικα από ένα αρχείο, ξέρει να σταματά όταν φτάσει στο τέλος του αρχείου.

Τι είναι ένα πρόγραμμα;

Ο στοιχειώδης ορισμός ενός *προγράμματος* είναι μια ακολουθία δηλώσεων Python που έχουν σχεδιαστεί για να κάνουν κάτι. Ακόμα και το απλό σενάριο *hello.py* είναι ένα πρόγραμμα. Είναι ένα πρόγραμμα μιας γραμμής και δεν είναι ιδιαίτερα χρήσιμο, αλλά με τον αυστηρό ορισμό, είναι ένα πρόγραμμα Python.

Ίσως είναι πιο εύκολο να καταλάβετε τι είναι ένα πρόγραμμα σκεπτόμενοι ένα πρόβλημα, το οποίο θέλετε να επιλύσετε κατασκευάζοντας ένα πρόγραμμα και μετά να προσπαθήσετε να κατασκευάσετε το πρόγραμμα αυτό.

Ας υποθέσουμε ότι κάνετε μια έρευνα Κοινωνικής Δικτύωσης σε αναρτήσεις στο Facebook και σας ενδιαφέρει η πιο συχνά χρησιμοποιούμενη λέξη σε μια σειρά αναρτήσεων. Θα μπορούσατε να εκτυπώσετε τη ροή των αναρτήσεων στο Facebook και να μελετήσετε το αποτέλεσμα, αναζητώντας την πιο συνηθισμένη λέξη, αλλά αυτό θα πάρει πολύ χρόνο και θα είναι πολύ επιρρεπές σε λάθη. Θα ήταν έξυπνο να γράψετε ένα πρόγραμμα Python για να χειριστεί την εργασία γρήγορα και με ακρίβεια, ώστε να μπορείτε να περάσετε το Σαββατοκύριακο σας κάνοντας κάτι πιο διασκεδαστικό.

Για παράδειγμα, κοιτάξτε το παρακάτω κείμενο για έναν κλόουν και ένα αυτοκίνητο. Κοιτάξτε το κείμενο και εντοπίστε την πιο κοινή λέξη και πόσες φορές εμφανίζεται.

```
the clown ran after the car and the car ran into the tent
and the tent fell down on the clown and the car
```

Στη συνέχεια, φανταστείτε ότι κάνετε αυτήν την δουλειά κοιτάζοντας εκατομμύρια γραμμές κειμένου. Ειλικρινά θα ήταν πιο γρήγορο για εσάς να μάθετε Python και να γράψετε ένα πρόγραμμα Python, για να μετρήσετε τις λέξεις από ό,τι θα ήταν να σαρώσετε τις λέξεις με το χέρι.

Τα ακόμη καλύτερα νέα είναι ότι ετοίμασα ήδη ένα απλό πρόγραμμα, για να βρω την πιο κοινή λέξη σε ένα αρχείο κειμένου. Το έγραψα, το δοκίμασα και τώρα σας το δίνω για να το χρησιμοποιήσετε και να εξοικονομήσετε χρόνο.

```
όνομα = input('Εισάγετε αρχείο:')
handle = open(όνομα, 'r')
πλήθη = dict()
for γραμμή in handle:
    λέξεις = γραμμή.split()
    for λέξη in λέξεις:
        πλήθη[λέξη] = πλήθη.get(λέξη, 0) + 1
```

```
maxπλήθος = None
maxλέξη = None
for λέξη, πλήθος in list(πλήθη.items()):
    if maxπλήθος is None or πλήθος > maxπλήθος:
        maxλέξη = λέξη
        maxπλήθος = πλήθος

print(maxλέξη, maxπλήθος)
```

 # Κώδικας στο: http://www.gr-py4e.com/code3/word.py

Δεν χρειάζεται καν να γνωρίζετε την Python για να χρησιμοποιήσετε αυτό το πρόγραμμα. Θα χρειαστεί να ανατρέξετε στο Κεφάλαιο 10 αυτού του βιβλίου, για να κατανοήσετε πλήρως τις εκπληκτικές τεχνικές Python που χρησιμοποιήθηκαν για τη δημιουργία του προγράμματος. Είστε ο τελικός χρήστης, απλά χρησιμοποιείτε το πρόγραμμα και θαυμάζετε την εξυπνάδα του και πώς σας γλίτωσε από τόσο πολύ χειρωνακτική προσπάθεια. Απλώς πληκτρολογείτε τον κώδικα σε ένα αρχείο που ονομάζεται *words.py* και το εκτελείτε ή κατεβάζετε τον πηγαίο κώδικα από το http://www.gr.py4e.com/code3/ και το εκτελείτε.

Αυτό είναι ένα καλό παράδειγμα για το πώς η Python και η γλώσσα Python λειτουργούν ως ενδιάμεσος μεταξύ εσάς (του τελικού χρήστη) και εμένα (του προγραμματιστή). Η Python είναι ένας τρόπος για να ανταλλάξουμε χρήσιμες ακολουθίες οδηγιών (δηλ. προγράμματα) σε μια κοινή γλώσσα που μπορεί να χρησιμοποιηθεί από οποιονδήποτε εγκαταστήσει την Python στον υπολογιστή του. Κανείς μας λοιπόν δεν μιλάει *με την Python*, αλλά επικοινωνούμε μεταξύ μας *μέσω* της Python.

Τα δομικά στοιχεία των προγραμμάτων

Στα επόμενα κεφάλαια, θα μάθουμε περισσότερα για το λεξιλόγιο, τη δομή των προτάσεων, τη δομή των παραγράφων και τη δομή της "ιστορίας" της Python. Θα μάθουμε για τις ισχυρές δυνατότητες της Python και πώς να συνδυάσουμε αυτές τις δυνατότητες για να δημιουργήσουμε χρήσιμα προγράμματα.

Υπάρχουν ορισμένα εννοιολογικά πρότυπα χαμηλού επιπέδου, που χρησιμοποιούμε για την κατασκευή προγραμμάτων. Αυτές οι κατασκευές δεν είναι μόνο για προγράμματα Python, είναι μέρος κάθε γλώσσας προγραμματισμού από τη γλώσσα μηχανής έως τις γλώσσες υψηλού επιπέδου.

είσοδος : Λήψη δεδομένα από τον "έξω κόσμο". Αυτό μπορεί να είναι η ανάγνωση

δεδομένων από ένα αρχείο ή ακόμη και κάποιου είδους αισθητήρα όπως μικρόφωνο ή GPS. Στα αρχικά μας προγράμματα, η εισαγωγή μας θα προέρχεται από τον χρήστη που πληκτρολογεί δεδομένα στο πληκτρολόγιο.

έξοδος : Εμφάνιση των αποτελεσμάτων του προγράμματος σε μια οθόνη ή αποθήκευση τους σε ένα αρχείο ή ίσως εγγραφή τους σε μια συσκευή, όπως μια συσκευή αναπαραγωγής μουσικής ή εκφώνησης κειμένου.

σειριακή εκτέλεση : Εκτέλεση δηλώσεων της μίας μετά την άλλη, με τη σειρά που συναντώνται στο σενάριο.

υπό όρους εκτέλεση : Έλεγχος για ορισμένες συνθήκες και, στη συνέχεια, εκτέλεση ή παράληψη μιας ακολουθίας εντολών.

επαναλαμβανόμενη εκτέλεση : Εκτέλεση κάποιου συνόλου δηλώσεων κατ' επανάληψη, συνήθως με κάποια παραλλαγή.

επαναχρησιμοποίηση : Γραφή μιας σειράς εντολών μία φορά, απόδοση όνομα σε αυτές και, στη συνέχεια, επαναχρησιμοποίησή τους, όπως χρειάζεται κάθε φορά, σε όλο το πρόγραμμά σας.

Ακούγεται, σχεδόν, πάρα πολύ απλό για να είναι αληθινό, και φυσικά δεν είναι ποτέ τόσο απλό. Είναι σαν να λέμε ότι το περπάτημα είναι απλά «να βάζεις το ένα πόδι μπροστά από το άλλο». Η «τέχνη» της συγγραφής ενός προγράμματος συνθέτει και υφαίνει αυτά τα βασικά στοιχεία μαζί πολλές φορές, για να παράγει κάτι που είναι χρήσιμο για τους χρήστες του.

Το παραπάνω πρόγραμμα καταμέτρησης λέξεων χρησιμοποιεί άμεσα όλα τα παραπάνω μοτίβα, εκτός από ένα.

Τι θα μπορούσε να πάει στραβά;

Όπως είδαμε στις πρώτες μας συνομιλίες με την Python, πρέπει να επικοινωνούμε με μεγάλη ακρίβεια, όταν γράφουμε κώδικα Python. Η παραμικρή απόκλιση ή λάθος θα κάνει την Python να σταματήσει την εκτέλεση του προγράμματός σας.

Οι αρχάριοι προγραμματιστές συχνά θεωρούν το γεγονός ότι η Python δεν αφήνει περιθώρια για λάθη ως απόδειξη ότι η Python είναι κακιά, μισητή και σκληρή. Ενώ η Python φαίνεται να συμπαθεί όλους τους άλλους, αυτούς τους γνωρίζει προσωπικά και τους κρατά κακία. Λόγω αυτής της μνησικακίας, η Python παίρνει τα τέλεια γραμμένα

προγράμματα μας και τα απορρίπτει ως "ακατάλληλα" μόνο και μόνο για να μας βασανίσει.

```
>>> primt 'Γειά σου κόσμε!'
File "<stdin>", line 1
  primt 'Γειά σου κόσμε!'
                    ^
SyntaxError: invalid syntax
>>> primt ('Γειά σου κόσμε')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'primt' is not defined
>>> Python σε μισώ!
File "<stdin>", line 1
  Python σε μισώ!
       ^
SyntaxError: invalid syntax
>>> αν βγεις από κει μέσα, θα σου δώσω ένα μαθηματάκι
File "<stdin>", line 1
   αν βγεις από κει μέσα, θα σου δώσω ένα μαθηματάκι
            ^
SyntaxError: invalid syntax
>>>
```

Δεν έχετε να κερδίσετε κάτι από τη διαμάχη με την Python. Είναι απλώς ένα εργαλείο. Δεν έχει συναισθήματα και είναι χαρούμενο και έτοιμο να σας εξυπηρετήσει όποτε το χρειαστείτε. Τα μηνύματα λάθους της ακούγονται σκληρά, αλλά είναι απλώς το κάλεσμα της Python για βοήθεια. Έχει εξετάσει τι πληκτρολογήσατε και απλά δεν μπορεί να καταλάβει τι έχετε εισαγάγει.

Η Python μοιάζει πάρα πολύ με ένα σκύλο, σε αγαπάει άνευ όρων, έχει μερικές λέξεις κλειδιά που καταλαβαίνει, σε κοιτάζει με ένα γλυκό βλέμμα (>>>) και περιμένει να πεις κάτι που καταλαβαίνει. Όταν η Python λέει "SyntaxError: invalid syntax", απλά κουνάει την ουρά της και λέει: "Φαινόσαστε να λέτε κάτι, αλλά απλώς δεν καταλαβαίνω τι εννοούσατε, αλλά συνεχίστε να μου μιλάτε (>>>)."

Καθώς τα προγράμματά σας γίνονται όλο και πιο εξελιγμένα, θα συναντήσετε τρεις γενικούς τύπους σφαλμάτων:

Syntax errors - Συντακτικά λάθη : Αυτά είναι τα πρώτα λάθη που θα κάνετε και είναι τα πιο εύκολα στο να τα διορθώσετε. Ένα συντακτικό λάθος σημαίνει ότι έχετε

παραβιάσει τους κανόνες "γραμματικής" της Python. Η Python κάνει ό,τι μπορεί για να δείξει ακριβώς σε ποια γραμμή και ποιον χαρακτήρα μπερδεύτηκε. Το μόνο δύσκολο κομμάτι συντακτικών σφαλμάτων είναι ότι μερικές φορές το λάθος που χρειάζεται διόρθωση είναι στην πραγματικότητα σε προηγούμενο σημείο στο πρόγραμμα και όχι εκεί όπου *παρατήρησε* η Python ότι μπερδεύτηκε. Έτσι, η γραμμή και ο χαρακτήρας που υποδεικνύει η Python σε ένα συντακτικό λάθος, μπορεί να είναι απλώς το σημείο εκκίνησης για την έρευνά σας.

Logic errors -Λογικά λάθη : Ένα λογικό λάθος προκύπτει όταν το πρόγραμμά σας έχει καλή σύνταξη αλλά υπάρχει κάποιο λάθος στη σειρά των δηλώσεων ή ίσως λάθος στον τρόπο που οι προτάσεις σχετίζονται μεταξύ τους. Ένα καλό παράδειγμα λογικού σφάλματος μπορεί να είναι: "Πάρτε ένα ποτό από το μπουκάλι νερό σας, βάλτε το στο σακίδιο σας, περπατήστε στη βιβλιοθήκη και, στη συνέχεια, τοποθετήστε το καπάκι στο μπουκάλι".

Semantic errors - Σημασιολογικά λάθη : Ένα σημασιολογικό σφάλμα είναι όταν η περιγραφή των βημάτων, που πρέπει να ακολουθηθούν είναι συντακτικά τέλεια και με τη σωστή σειρά, αλλά απλώς υπάρχει ένα λάθος στο πρόγραμμα. Το πρόγραμμα είναι απόλυτα σωστό αλλά δεν κάνει αυτό που το *προορίζατε* για να κάνει. Ένα απλό παράδειγμα θα ήταν αν δίνατε σε κάποιον οδηγίες για το πώς θα φτάσει σε ένα εστιατόριο και λέγατε, "... όταν φτάσετε στη διασταύρωση με το βενζινάδικο, στρίψτε αριστερά, προχωρήστε ένα μίλι και το εστιατόριο είναι ένα κόκκινο κτίριο στα αριστερά σας". Ο φίλος σας έχει αργήσει πολύ και σας καλεί για να σας πει ότι βρίσκεται σε ένα αγρόκτημα και τριγυρνά πίσω από έναν αχυρώνα, χωρίς σημάδι εστιατορίου. Τότε του λέτε "έστριψες αριστερά ή δεξιά στο βενζινάδικο;" και σας λέει, "ακολούθησα τέλεια τις οδηγίες σου, τις έχω καταγράψει, λέει στρίψε αριστερά και προχώρησε ένα μίλι μετά το βενζινάδικο". Και τότε του λέτε: «Λυπάμαι πολύ, γιατί ενώ οι οδηγίες μου ήταν συντακτικά σωστές, δυστυχώς περιείχαν ένα μικρό, αλλά μη εντοπισμένο σημασιολογικό σφάλμα».

Και πάλι, και στους τρεις τύπους λαθών, η Python προσπαθεί σκληρά, απλώς να κάνει ό,τι ακριβώς ζητήσατε.

Εκσφαλμάτωση

Όταν, στην Python, προκύπτει ένα λάθος ή ακόμα και όταν σας δίνει ένα αποτέλεσμα που είναι διαφορετικό από αυτό που θα έπρεπε, τότε ξεκινά η αναζήτηση της αιτίας του

σφάλματος. Η εντοπισμός σφαλμάτων είναι η διαδικασία εύρεσης της αιτίας του σφάλματος στον κώδικά σας. Όταν κάνετε εντοπισμό σφαλμάτων σε ένα πρόγραμμα, και ειδικά εάν εργάζεστε σε ένα δύσκολο σφάλμα, υπάρχουν τέσσερα πράγματα που πρέπει να δοκιμάσετε:

ανάγνωση : Ελέγξτε τον κώδικά σας, διαβάστε τον ξανά στον εαυτό σας, και ελέγξτε ότι λέει αυτό που θέλατε να πείτε.

εκτέλεση : Πειραματιστείτε κάνοντας αλλαγές και εκτελώντας διαφορετικές εκδόσεις. Συχνά, εάν εμφανίζεται το σωστό πράγμα στο σωστό σημείο του προγράμματος, το πρόβλημα γίνεται εμφανές, αλλά μερικές φορές πρέπει να αφιερώσετε λίγο χρόνο για να φτιάξετε κρηπιδώματα.

μηρυκασμός : Αφιερώστε λίγο χρόνο στο να σκεφτείτε! Τι είδους σφάλμα είναι: σύνταξης, εκτέλεσης, σημασιολογικό; Τι πληροφορίες μπορείτε να πάρετε από τα μηνύματα σφάλματος ή από την έξοδο του προγράμματος; Τι είδους σφάλμα μπορεί να προκαλέσει το πρόβλημα που βλέπετε; Τι αλλάξατε τελευταία, πριν εμφανιστεί το πρόβλημα;

υποχώρηση : Κάποια στιγμή, το καλύτερο που μπορείτε να κάνετε είναι να κάνετε πίσω, αναιρώντας τις πρόσφατες αλλαγές, μέχρι να επιστρέψετε σε ένα πρόγραμμα που λειτουργεί και το καταλαβαίνετε. Στη συνέχεια, μπορείτε να ξανά ξεκινήσετε την κατασκευή του.

Οι αρχάριοι προγραμματιστές κολλάνε μερικές φορές σε ένα από τα παραπάνω και ξεχνούν τα υπόλοιπα. Ο εντοπισμός ενός δύσκολου σφάλματος απαιτεί ανάγνωση, εκτέλεση, μηρυκασμό και, μερικές φορές, υποχώρηση. Εάν κολλήσετε σε μία από αυτές τις δραστηριότητες, δοκιμάστε και τις υπόλοιπες. Κάθε δραστηριότητα συνοδεύεται από το δικό της τρόπο αποτυχίας.

Για παράδειγμα, η ανάγνωση του κώδικα μπορεί να βοηθήσει εάν το πρόβλημα είναι κάποιο τυπογραφικό λάθος, αλλά όχι εάν το πρόβλημα είναι κάποια εννοιολογική παρανόηση. Εάν δεν καταλαβαίνετε τι κάνει το πρόγραμμά σας, μπορεί να το διαβάσετε 100 φορές και να μην εντοπίσετε ποτέ το λάθος, επειδή το σφάλμα είναι στο μυαλό σας.

Η εκτέλεση πειραμάτων μπορεί να βοηθήσει, ειδικά αν εκτελείτε μικρές, απλές δοκιμές. Αλλά εάν εκτελείτε πειράματα χωρίς να σκεφτείτε ή να διαβάσετε τον κώδικά σας, μπορεί να πέσετε σε ένα μοτίβο που ονομάζω "προγραμματισμός τυχαίων περιπάτων", η οποία είναι η διαδικασία πραγματοποίησης τυχαίων αλλαγών έως ότου το πρόγραμμά σας λειτουργήσει σωστά. Περιττό να πούμε ότι ο προγραμματισμός

τυχαίων περιπάτων μπορεί να διαρκέσει πολύ.

Πρέπει να αφιερώσετε χρόνο στο να σκεφτείτε. Το Debugging είναι σαν μια πειραματική επιστήμη. Θα πρέπει να έχετε τουλάχιστον μία υπόθεση για το ποιο είναι το πρόβλημα. Εάν υπάρχουν δύο ή περισσότερες υποθέσεις, προσπαθήστε να σκεφτείτε μια δοκιμή που θα εξαλείψει μία από αυτές.

Το διάλειμμα βοηθά στη σκέψη. Το ίδιο και η συζήτηση. Εάν εξηγήσετε το πρόβλημα σε κάποιον άλλο (ή ακόμα και στον εαυτό σας), μερικές φορές θα βρείτε την απάντηση πριν τελειώσετε την ερώτηση.

Αλλά ακόμη και οι καλύτερες τεχνικές εντοπισμού σφαλμάτων θα αποτύχουν εάν υπάρχουν πάρα πολλά λάθη ή εάν ο κώδικας που προσπαθείτε να διορθώσετε είναι πολύ μεγάλος και περίπλοκος. Μερικές φορές η καλύτερη επιλογή είναι να υποχωρήσετε, απλοποιώντας το πρόγραμμα μέχρι να φτάσετε σε κάτι που λειτουργεί και που καταλαβαίνετε.

Οι αρχάριοι προγραμματιστές είναι συχνά απρόθυμοι να υποχωρήσουν επειδή δεν αντέχουν να διαγράψουν ούτε μια γραμμή κώδικα (ακόμα και αν είναι λάθος). Εάν σας κάνει να νιώσετε καλύτερα, αντιγράψτε το πρόγραμμά σας σε άλλο αρχείο πριν αρχίσετε να το απογυμνώνετε. Στη συνέχεια, μπορείτε να αρχίσετε να επικολλάτε ξανά, μικρά κομμάτια κάθε φορά.

Το ταξίδι της μάθησης

Καθώς προχωράτε στο υπόλοιπο βιβλίο, μην φοβηθείτε εάν οι έννοιες δεν φαίνεται να ταιριάζουν και τόσο καλά, την πρώτη φορά. Όταν μαθαίνατε να μιλάτε, δεν ήταν πρόβλημα που τα πρώτα σας χρόνια κάνατε απλώς χαριτωμένους θορύβους και γρυλισμούς. Και δεν πείραζε, αν και χρειάστηκαν έξι μήνες για να μεταβείτε από το απλό λεξιλόγιο σε απλές προτάσεις και χρειάστηκαν 5-6 χρόνια ακόμη για να μεταβείτε από προτάσεις σε παραγράφους και μερικά χρόνια ακόμη για να μπορέσετε να γράψετε ένα ενδιαφέρον, μικρό και πλήρες διήγημα μόνοι σας.

Θέλουμε να μάθετε την Python πολύ πιο γρήγορα, οπότε τα διδάσκουμε όλα ταυτόχρονα στα επόμενα κεφάλαια. Αλλά είναι σαν να μαθαίνετε μια νέα γλώσσα, που χρειάζεται χρόνο για να την απορροφήσετε και να την κατανοήσετε, πριν κάνετε κτήμα σας. Αυτό οδηγεί σε κάποια σύγχυση, καθώς εξετάζουμε και επανεξετάζουμε θέματα για να προσπαθήσουμε να σας κάνουμε να δείτε τη μεγάλη εικόνα ενώ ορίζουμε τα μικροσκοπικά κομμάτια που συνθέτουν αυτή τη μεγάλη εικόνα. Ενώ το βιβλίο είναι

γραμμένο γραμμικά, και αν παρακολουθείτε ένα μάθημα θα προχωρήσει με γραμμικό τρόπο, μην διστάσετε να είστε μη γραμμικοί στον τρόπο με τον οποίο προσεγγίζετε το υλικό. Κοιτάξτε μπροστά και πίσω και διαβάστε χαλαρά. Μια γρήγορη ανάγνωση του πιο προηγμένου υλικού, χωρίς απαραίτητα να κατανοείτε πλήρως τις λεπτομέρειες, μπορεί να οδηγήσει σε καλύτερη κατανόηση του "γιατί;" του προγραμματισμού. Επανεξετάζοντας το προηγούμενο υλικό και ακόμη επαναλαμβάνοντας προηγούμενες ασκήσεις, θα συνειδητοποιήσετε ότι στην πραγματικότητα μάθατε πολύ υλικό ακόμα κι αν το υλικό που κοιτάτε επί του παρόντος φαίνεται λίγο ακατανόητο.

Συνήθως όταν μαθαίνετε την πρώτη σας γλώσσα προγραμματισμού, υπάρχουν μερικές υπέροχες "Αχά!" στιγμές, που μπορείτε να κοιτάξετε από ψηλά το πελέκημα της πέτρας με σφυρί και σμίλη και, κάνοντας ένα βήμα πίσω, να δείτε ότι πράγματι δημιουργείτε ένα όμορφο γλυπτό.

Αν κάτι φαίνεται ιδιαίτερα δύσκολο, συνήθως δεν αξίζει να ξενυχτάτε και να το κοιτάτε επίμονα. Κάντε ένα διάλειμμα, πάρτε έναν υπνάκο, φάτε ένα σνακ, εξηγήστε σε κάποιον τι αντιμετωπίζετε (ή ίσως το σκυλί σας) και, στη συνέχεια, επιστρέψτε σε αυτό με φρέσκια ματιά. Σας διαβεβαιώνω ότι μόλις μάθετε τις έννοιες προγραμματισμού του βιβλίου, θα κοιτάξετε πίσω και θα δείτε ότι όλα ήταν πραγματικά εύκολα και κομψά και απλώς σας πήρε λίγο χρόνο για να το απορροφήσετε.

Γλωσσάριο

bug : Ένα λάθος του προγράμματος.

ανάλυση - parse : Η εξέταση ενός προγράμματος και η ανάλυση της συντακτικής δομής του.

γλώσσα μηχανής : Η γλώσσα χαμηλότερου επιπέδου για λογισμικό, η οποία είναι η γλώσσα που εκτελείται απευθείας από την κεντρική μονάδα επεξεργασίας (CPU).

γλώσσα υψηλού επιπέδου : Μια γλώσσα προγραμματισμού, όπως η Python, που έχει σχεδιαστεί για να είναι εύκολη η ανάγνωσή της και η γραφή της από ανθρώπους.

γλώσσα χαμηλού επιπέδου : Μια γλώσσα προγραμματισμού που έχει σχεδιαστεί για να είναι εύκολη στην κατανόησή της από τον υπολογιστή. Ονομάζεται επίσης "γλώσσα μηχανής" ή "γλώσσα assembly".

δευτερεύουσα μνήμη : Αποθηκεύει προγράμματα και δεδομένα και διατηρεί τις πληροφορίες ακόμη και όταν σταματήσει η τροφοδοσία ρεύματος. Γενικά πιο

αργή από την κύρια μνήμη. Παραδείγματα δευτερεύουσας μνήμης αποτελούν οι μονάδες δίσκου και η μνήμη flash σε USB sticks.

διαδραστική λειτουργία - interactive mode : Ένας τρόπος χρήσης του διερμηνέα της Python, πληκτρολογώντας εντολές και εκφράσεις στη γραμμή προτροπής.

διερμηνεία : Για να εκτελέσετε ένα πρόγραμμα σε γλώσσα υψηλού επιπέδου, μεταφράζοντάς το μία προς μία γραμμή.

επίλυση προβλήματος : Η διαδικασία διατύπωσης ενός προβλήματος, εύρεσης μιας λύσης και έκφρασης της λύσης αυτής.

κεντρική μονάδα επεξεργασίας : Η καρδιά κάθε υπολογιστή. Είναι αυτό που τρέχει το λογισμικό που γράφουμε. Ονομάζεται επίσης "CPU" ή "ο επεξεργαστής".

κύρια μνήμη : Αποθηκεύει προγράμματα και δεδομένα. Η κύρια μνήμη χάνει τις πληροφορίες της όταν σταματήσει η τροφοδοσία ρεύματος. \index{κύρια μνήμη}

μεταγλώττιση - compile : Η μετάφραση ενός ολόκληρου προγράμματος, γραμμένου σε γλώσσα υψηλού επιπέδου, ταυτόχρονα, σε γλώσσα χαμηλού επιπέδου, προετοιμάζοντάς το για μετέπειτα εκτέλεση.

μεταφερσιμότητα : Η ιδιότητα ενός προγράμματος να μπορεί να εκτελεστεί σε περισσότερα από ένα είδη υπολογιστών.

πηγαίος κώδικας : Ένα πρόγραμμα σε γλώσσα υψηλού επιπέδου.

πρόγραμμα : Ένα σύνολο οδηγιών, που καθορίζει έναν υπολογισμό.

προτροπή - prompt : Όταν ένα πρόγραμμα εμφανίζει ένα μήνυμα και περιμένει από ο χρήστης να πληκτρολογήσει κάποια είσοδο στο πρόγραμμα.

σημασιολογία : Το νόημα ενός προγράμματος.

σημασιολογικό λάθος : Ένα σφάλμα σε ένα πρόγραμμα που ως αποτέλεσμα έχει το να κάνει κάτι διαφορετικό από αυτό που ήθελε ο προγραμματιστής.

συνάρτηση print : Μια οδηγία προς τον διερμηνέα της Python, που προκαλεί την εμφάνιση τιμών στην οθόνη.

Ασκήσεις

Άσκηση 1: Ποια είναι η λειτουργία της δευτερεύουσας μνήμης σε έναν υπολογιστή;

a) Εκτελέστε όλο τον υπολογισμό και τη λογική του προγράμματος
b) Ανακτά ιστοσελίδες μέσω Διαδικτύου
c) Αποθηκεύστε πληροφορίες μακροπρόθεσμα, ακόμη και πέρα από έναν κύκλο ισχύος
d) Δέχεται είσοδο από τον χρήστη

Άσκηση 2: Τι είναι ένα πρόγραμμα;

Άσκηση 3: Ποια είναι η διαφορά μεταξύ μεταγλωττιστή και διερμηνέα;

Άσκηση 4: Ποιο από τα παρακάτω περιέχει "κώδικα μηχανής";

a) Ο διερμηνέας της Python
b) Το πληκτρολόγιο
c) Το πηγαίο αρχείου Python
d) Ένα έγγραφο επεξεργαστή κειμένου

Άσκηση 5: Τί λάθος υπάρχει στον ακόλουθο κώδικα;

```
>>> primt 'Hello world!'
File "<stdin>", line 1
  primt 'Hello world!'
                     ^
SyntaxError: invalid syntax
>>>
```

Άσκηση 6: Πού αποθηκεύεται στον υπολογιστή μια μεταβλητή όπως το "x", μετά την ολοκλήρωση της ακόλουθης γραμμής Python;

```
x = 123
```

a) Κεντρική μονάδα επεξεργασίας
b) Κύρια μνήμη
c) Δευτερεύουσα μνήμη
d) Συσκευές εισόδου
e) Συσκευές εξόδου

Άσκηση 7: Τί θα εμφανίσει το ακόλουθο πρόγραμμα;

```
x = 43
x = x + 1
print(x)
```

a) 43

b) 44

c) x + 1

d) Σφάλμα επειδή το x = x + 1 δεν είναι μαθηματικά σωστό

Άσκηση 8: Εξηγήστε καθένα από τα παρακάτω χρησιμοποιώντας ως παράδειγμα μία ανθρώπινη ικανότητα: (1) Κεντρική μονάδα επεξεργασίας, (2) Κύρια μνήμη, (3) Δευτερεύουσα μνήμη, (4) Συσκευή εισόδου και (5) Συσκευή εξόδου. Για παράδειγμα, "Τι είναι το ανθρώπινο ισοδύναμο με μια κεντρική μονάδα επεξεργασίας";

Άσκηση 9: Πώς διορθώνετε ένα "Συντακτικό Λάθος";

Κεφάλαιο 2

Μεταβλητές, εκφράσεις και εντολές

Τιμές και τύποι

Μια *τιμή*, όπως ένα γράμμα ή ένας αριθμός, είναι ένα από τα βασικά στοιχεία με τα οποία λειτουργεί ένα πρόγραμμα. Οι τιμές που έχουμε δει μέχρι τώρα είναι 1, 2 και "Γειά σου κόσμε!"

Αυτές οι τιμές ανήκουν σε διαφορετικούς *τύπους*: το 2 είναι ένας ακέραιος αριθμός και το "Γειά σου κόσμε!" είναι μια *συμβολοσειρά (string)*, που ονομάζεται επειδή περιέχει μια "σειρά" συμβόλων και γραμμάτων. Μπορείτε (εσείς αλλά και ο διερμηνέας) να εντοπίσετε συμβολοσειρές επειδή περικλείονται σε εισαγωγικά.

Η εντολή `print` λειτουργεί και για ακέραιους αριθμούς. Χρησιμοποιούμε την εντολή `python` για να ξεκινήσουμε τον διερμηνέα.

```
python
>>> print(4)
4
```

Εάν δεν είστε σίγουροι τι τύπου είναι μια τιμή, ο διερμηνέας μπορεί να σας πει.

```
>>> type('Hello, World!')
<class 'str'>
>>> type(17)
<class 'int'>
```

Δεν αποτελεί έκπληξη το γεγονός ότι οι συμβολοσειρές ανήκουν στον τύπο `str` και οι ακέραιοι στον τύπο `int`. Λιγότερο προφανώς, οι αριθμοί με υποδιαστολή ανήκουν σε έναν τύπο που ονομάζεται `float`, επειδή αυτοί οι αριθμοί αντιπροσωπεύονται από μια μορφή που ονομάζεται *floating point*.

```
>>> type(3.2)
<class 'float'>
```

Τι γίνεται με τις τιμές όπως το "17" και το "3.2"; Μοιάζουν με αριθμούς, αλλά περικλείονται με εισαγωγικά σαν συμβολοσειρές.

```
>>> type('17')
<class 'str'>
```

```
>>> type('3.2')
<class 'str'>
```

Είναι συμβολοσειρές.

Όταν πληκτρολογείτε έναν μεγάλο ακέραιο, μπορεί να μπείτε στον πειρασμό να χρησιμοποιήσετε διαχωριστικά χιλιάδων, όπως στο 1.000.000. Αυτός δεν είναι ένας έγκυρος ακέραιος αριθμός στην Python, αποδεκτό είναι το:

```
>>> print(1,000,000)
1 0 0
```

Ε, αυτό δεν το περιμέναμε καθόλου! Η Python ερμηνεύει το 1,000,000 ως μια ακολουθία ακέραιων διαχωρισμένων με κόμμα, την οποία εκτυπώνει με κενά μεταξύ τους.

Αυτό είναι το πρώτο παράδειγμα που έχουμε δει για ένα σημασιολογικό σφάλμα: ο κώδικας τρέχει χωρίς να παράγει μήνυμα λάθους, αλλά δεν κάνει το "σωστό".

Μεταβλητές

Ένα από τα πιο ισχυρά χαρακτηριστικά μιας γλώσσας προγραμματισμού είναι η δυνατότητα χειρισμού *μεταβλητών*. Μια μεταβλητή είναι ένα όνομα που αναφέρεται σε μια τιμή.

Μια *εντολή εκχώρησης* δημιουργεί νέες μεταβλητές και τους δίνει τιμές:

```
>>> message = 'Και τώρα κάτι εντελώς διαφορετικό'
>>> n = 17
>>> pi = 3.1415926535897931
```

Αυτό το παράδειγμα υλοποιεί τρεις αναθέσεις. Η πρώτη αναθέτει μια συμβολοσειρά σε μια νέα μεταβλητή με το όνομα `message`, η δεύτερη αναθέτει τον ακέραιο 17 στο `n` και η τρίτη αναθέτει την τιμή (κατά προσέγγιση) του π στο `pi`.

Για να εμφανίσετε την τιμή μιας μεταβλητής, μπορείτε να χρησιμοποιήσετε μια εντολή print:

```
>>> print(n)
17
>>> print(pi)
3.141592653589793
```

Ο τύπος μιας μεταβλητής είναι ο τύπος της τιμής στην οποία αναφέρεται.

```
>>> type(message)
<class 'str'>
>>> type(n)
<class 'int'>
>>> type(pi)
<class 'float'>
```

Ονόματα μεταβλητών και δεσμευμένες λέξεις

Οι προγραμματιστές επιλέγουν, γενικά, ονόματα για τις μεταβλητές τους που έχουν νόημα και δηλώνουν τον λόγο για τον οποίο χρησιμοποιείται η μεταβλητή.

Τα ονόματα των μεταβλητών μπορεί να είναι αυθαίρετα μεγάλα. Μπορούν να περιέχουν γράμματα και αριθμούς, αλλά δεν μπορούν να ξεκινήσουν με αριθμό. Είναι αποδεκτό να χρησιμοποιείτε κεφαλαία γράμματα, αλλά είναι καλή ιδέα να αρχίζετε τα ονόματα μεταβλητών με πεζό γράμμα (θα δείτε το γιατί αργότερα).

Σε ένα όνομα μπορεί να χρησιμοποιηθεί και ο χαρακτήρας υπογράμμισης (_) ή κάτω παύλα. Συχνά χρησιμοποιείται σε ονόματα με πολλές λέξεις, όπως `my_name` ή `airspeed_of_unladen_swallow`. Τα ονόματα μεταβλητών μπορούν να ξεκινούν με χαρακτήρα υπογράμμισης, αλλά γενικά αποφεύγουμε να το κάνουμε αυτό, εκτός εάν γράφουμε κώδικα βιβλιοθήκης για χρήση από άλλους.

Εάν δώσετε σε μια μεταβλητή ένα μη αποδεκτό όνομα, προκύπτει σφάλμα σύνταξης:

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

Το `76trombones` είναι μη αποδεκτό επειδή αρχίζει με αριθμό. Το `more@` είναι μη αποδεκτό επειδή περιέχει έναν μη αποδεκτό χαρακτήρα, @. Αλλά ποιο το πρόβλημα με το `class`;

Αποδεικνύεται ότι το `class` είναι μία από τις *δεσμευμένες λέξεις* της Python. Ο διερμηνέας χρησιμοποιεί δεσμευμένες λέξεις για να αναγνωρίσει τη δομή του προγράμματος και δεν μπορούν να χρησιμοποιηθούν ως ονόματα μεταβλητών.

Η Python διαθέτει 35 δεσμευμένες λέξεις:

```
and       del       from      None      True
as        elif      global    nonlocal  try
assert    else      if        not       while
break     except    import    or        with
class     False     in        pass      yield
continue  finally   is        raise     async
def       for       lambda    return    await
```

Ίσως θα ήταν χρήσιμο να κρατήσετε αυτήν τη λίστα εύκαιρη. Εάν ο διερμηνέας παραπονιέται για ένα από τα ονόματα μεταβλητών σας και δεν ξέρετε γιατί, ελέγξτε αν βρίσκεται σε αυτήν τη λίστα.

Εντολές

Μια *εντολή* είναι μια μονάδα κώδικα που μπορεί να εκτελέσει ο διερμηνέας της Python. Έχουμε συναντήσει δύο είδη εντολών: την εντολή print και την ανάθεση τιμής.

Όταν πληκτρολογείτε μια εντολή σε διαδραστική λειτουργία, ο διερμηνέας την εκτελεί και εμφανίζει το αποτέλεσμα, εάν προκύπτει κάποιο.

Ένα σενάριο/script περιέχει συνήθως μια ακολουθία εντολών. Εάν υπάρχουν περισσότερες από μία εντολές, τα αποτελέσματα εμφανίζονται ένα κάθε φορά, καθώς εκτελούνται οι εντολές.

Για παράδειγμα, το script

```
print(1)
x = 2
print(x)
```

παράγει την έξοδο

```
1
2
```

Η εντολή εκχώρησης δεν παράγει έξοδο.

Τελεστές και τελεστέοι

Οι *τελεστές* είναι ειδικά σύμβολα που αναπαριστούν υπολογισμούς, όπως της πρόσθεσης και του πολλαπλασιασμού. Οι τιμές στις οποίες εφαρμόζεται ο τελεστής

καλούνται *τελεστέοι*.

Οι τελεστές +, -, *, / και ** εκτελούν πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση και ύψωση σε δύναμη, όπως στα παρακάτω παραδείγματα:

```
20 + 32
ώρα - 1
ώρα * 60 + λεπτά
λεπτά / 60
5**2
(5 + 9) * (15 - 7)
```

Υπήρξε μια αλλαγή στον τελεστή της διαίρεσης, μεταξύ Python 2.x και Python 3.x. Στην Python 3.x, το αποτέλεσμα αυτής της διαίρεσης είναι float:

```
>>> λεπτά = 59
>>> λεπτά / 60
0.9833333333333333
```

Ο τελεστής διαίρεσης στην Python 2.0, όταν διαιρεί δύο ακέραιους αριθμούς περικόπτει το αποτέλεσμα σε ακέραιο:

```
>>> λεπτά = 59
>>> λεπτά / 60
0
```

Για να λάβετε την ίδια απάντηση στην Python 3.0, χρησιμοποιήστε τη ευκλείδεια διαίρεση (// integer).

```
>>> λεπτά = 59
>>> λεπτά // 60
0
```

Στην Python 3.0, η ακέραιη διαίρεση λειτουργεί πολύ καλύτερα από ό,τι θα περιμένατε εάν εισαγάγατε την έκφραση σε μια αριθμομηχανή.

Εκφράσεις

Μια *έκφραση* είναι ένας συνδυασμός τιμών, μεταβλητών και τελεστών. Μια τιμή, από μόνη της, θεωρείται ως μία έκφραση και το ίδιο και μια μεταβλητή. Έτσι, τα παρακάτω είναι αποδεκτές μορφές εκφράσεων (υποθέτοντας ότι στη μεταβλητή x έχει εκχωρηθεί μία τιμή):

```
17
x
x + 17
```

Εάν πληκτρολογήσετε μια έκφραση σε διαδραστική λειτουργία, ο διερμηνέας την *υπολογίζει* και εμφανίζει το αποτέλεσμα:

```
>>> 1 + 1
2
```

Αλλά σε ένα script, μια έκφραση από μόνη της δεν κάνει κάτι! Αυτή είναι μια συνήθης πηγή σύγχυσης για αρχάριους.

Άσκηση 1: Πληκτρολογήστε τις ακόλουθες εντολές στον διερμηνέα της Python για να δείτε τι κάνουν:

```
5
x = 5
x + 1
```

Προτεραιότητα τελεστών

Όταν περισσότεροι από ένας τελεστές εμφανίζονται σε μία έκφραση, η σειρά εκτέλεσης εξαρτάται από τους *κανόνες προτεραιότητας* των πράξεων. Για τους μαθηματικούς τελεστές, αριθμητικούς, η Python ακολουθεί τη μαθηματική σύμβαση. Το ακρωνύμιο *PEMDAS* είναι ένας τρόπος για να θυμάστε τον κανόνα:

- *P*arentheses/Παρενθέσεις, έχουν την υψηλότερη προτεραιότητα και μπορούν να χρησιμοποιηθούν για να αναγκάσετε μια έκφραση να υπολογιστεί με βάση τη σειρά που εσείς επιθυμείτε. Δεδομένου ότι οι εκφράσεις στις παρενθέσεις αξιολογούνται πρώτες, το `2 * (3 - 1)` είναι 4 και το `(1 + 1) ** (5 - 2)` είναι 8. Μπορείτε επίσης να χρησιμοποιήσετε παρενθέσεις για να κάνετε μια έκφραση πιο εύκολα κατανοητή, όπως στο `(λεπτό * 100) / 60`, ακόμα κι αν δεν αλλάζει το αποτέλεσμα.
- *E*xponentiation/Ύψωση σε δύναμη, έχει την επόμενη υψηλότερη προτεραιότητα, οπότε `2 ** 1 + 1` κάνει 3, όχι 4, και `3 * 1 ** 3` κάνει 3 και όχι 27.
- *M*ultiplication/Πολλαπλασιασμός και *D*ivision/Διαίρεση, έχουν την ίδια προτεραιότητα, που είναι υψηλότερη από την *A*ddition/Πρόσθεση και την

Subtraction/Αφαίρεση, που έχουν επίσης την ίδια προτεραιότητα. Άρα το `2 * 3 - 1` είναι 5, όχι 4, και το `6 + 4 / 2` είναι 8 και όχι 5.

- Οι τελεστές με την ίδια προτεραιότητα εκτελούνται από αριστερά προς τα δεξιά. Άρα η έκφραση `5 - 3 - 1` κάνει 1 και όχι 3, γιατί το `5 - 3` εκτελείται πρώτο και μετά το `1` αφαιρείται από το 2.

Όταν έχετε αμφιβολίες, βάζετε πάντα παρενθέσεις στις εκφράσεις σας για να βεβαιωθείτε ότι οι υπολογισμοί εκτελούνται με τη σειρά που θέλετε.

Τελεστής Modulus/Ακέραιο Υπόλοιπο

Ο *τελεστής του ακεραίου υπολοίπου* εφαρμόζεται σε ακεραίους και επιστρέφει το υπόλοιπο του προκύπτει όταν πρώτος τελεστέος διαιρεθεί με τον δεύτερο. Στην Python, ο τελεστής του ακεραίου υπολοίπου είναι το σύμβολο επί τοις εκατό (%). Η σύνταξη είναι η ίδια όπως και στους υπόλοιπους τελεστές:

```
>>> πηλίκο = 7 // 3
>>> print(πηλίκο)
2
>>> υπόλοιπο = 7 % 3
>>> print(υπόλοιπο)
1
```

Άρα το 7 διαιρούμενο με το 3 είναι 2 με 1 να περισσεύει.

Ο τελεστής υπολοίπου αποδεικνύεται εκπληκτικά χρήσιμος. Για παράδειγμα, μπορείτε να ελέγξετε εάν ένας αριθμός διαιρείται με έναν άλλο: αν `x % y` είναι μηδέν, τότε το `x` διαιρείται με το `y`.

Μπορείτε επίσης να εξαγάγετε το τελευταίο ψηφίο ή τα τελευταίο ψηφία ενός αριθμού. Για παράδειγμα, το `x % 10` δίνει το τελευταίο ψηφίο του `x` (στη βάση 10). Ομοίως, το `x % 100` δίνει τα δύο τελευταία του ψηφία.

Τελεστές συμβολοσειρών

ΤΟ τελεστής + λειτουργεί στις συμβολοσειρές, αλλά δεν είναι η πρόσθεση με τη μαθηματική της έννοια. Αντ' αυτού, εκτελεί *συνένωση (concatenation)*, που σημαίνει ότι ενώνει τις συμβολοσειρές συνδέοντάς τες την μία μετά την άλλη. Για παράδειγμα:

```
>>> πρώτο = 10
```

```
>>> δεύτερο = 15
>>> print(πρώτο + δεύτερο)
25
>>> πρώτο = '100'
>>> δεύτερο = '150'
>>> print(πρώτο + δεύτερο)
100150
```

Ο τελεστής * λειτουργεί επίσης με συμβολοσειρές πολλαπλασιάζοντας το περιεχόμενο μιας συμβολοσειράς με έναν ακέραιο. Για παράδειγμα:

```
>>> πρώτο = 'Τεστ '
>>> δεύτερο = 3
>>> print(πρώτο * δεύτερο)
Τεστ Τεστ Τεστ
```

Ζητώντας είσοδο από το χρήστη

Μερικές φορές θέλουμε να πάρουμε την τιμή μιας μεταβλητής από τον χρήστη, μέσω του πληκτρολογίου του. Η Python περιέχει μια ενσωματωμένη συνάρτηση που ονομάζεται `input` και λαμβάνει είσοδο από το πληκτρολόγιο[1]. Όταν καλείτε αυτή η συνάρτηση, το πρόγραμμα σταματά και περιμένει τον χρήστη να πληκτρολογήσει κάτι. Όταν ο χρήστης πατήσει `Return` ή `Enter`, το πρόγραμμα συνεχίζει την εκτέλεσή του και η `input` επιστρέφει αυτό που ο χρήστης πληκτρολόγησε ως συμβολοσειρά.

```
>>> inp = input()
Some silly stuff
>>> print(inp)
Some silly stuff
```

Πριν ζητήσουμε είσοδο από τον χρήστη, καλό θα ήταν να εκτυπώσουμε μια προτροπή προς το χρήστη, που να του λέει τι να εισάγει. Μπορείτε να δώσετε μια συμβολοσειρά στο `input` για να εμφανιστεί στον χρήστη πριν γίνει η παύση για την εισαγωγή:

```
>>> όνομα = input('Πώς σε λένε;\n')
Πώς σε λένε;
Chuck
```

[1] Στην Python 2.0, αυτή η συνάρτηση ονομαζόταν `raw_input`.

```
>>> print(name)
Chuck
```

Η ακολουθία \n στο τέλος της προτροπής αντιπροσωπεύει μια *νέα γραμμή (newline)*, η οποία είναι ένας ειδικός χαρακτήρας, που προκαλεί αλλαγή γραμμής. Αυτός είναι ο λόγος για τον οποίο η εισαγωγή του χρήστη εμφανίζεται κάτω από την προτροπή.

Εάν ο χρήστης θα πρέπει να πληκτρολογήσει έναν ακέραιο, μπορείτε να δοκιμάσετε να μετατρέψετε την τιμή επιστροφής σε int χρησιμοποιώντας τη συνάρτηση int():

```
>>> προτροπή = 'Με τί...ταχύτητα πετάει ένα χελιδόνι;\n'
>>> ταχύτητα = input(προτροπή)
Με τί...ταχύτητα πετάει ένα χελιδόνι;
17
>>> int(ταχύτητα)
17
>>> int(ταχύτητα) + 5
22
```

Αλλά αν ο χρήστης πληκτρολογήσει κάτι άλλο, εκτός από μια σειρά ψηφίων, λαμβάνετε μήνυμα λάθους:

```
>>> ταχύτητα = input(προτροπή)
Με τί...ταχύτητα πετάει ένα χελιδόνι;
Τί εννοείς, ένα Αφρικανικό ή Ευρωπαϊκό χελιδόνι;
>>> int(ταχύτητα)
ValueError: invalid literal for int() with base 10:
```

Θα μάθουμε πως να χειριζόμαστε αυτά τα λάθη αργότερα.

Σχόλια

Καθώς τα προγράμματα γίνονται μεγαλύτερα και πιο περίπλοκα, γίνονται πιο δύσκολο να διαβαστούν. Οι επίσημες γλώσσες είναι πυκνές και συχνά είναι δύσκολο να κοιτάξουμε ένα κομμάτι κώδικα και να καταλάβουμε τι κάνει ή γιατί.

Για το λόγο αυτό, είναι καλή ιδέα να προσθέτετε σημειώσεις στα προγράμματά σας για να εξηγείτε σε φυσική γλώσσα τι κάνει το πρόγραμμα. Αυτές οι σημειώσεις ονομάζονται *σχόλια* και στην Python ξεκινούν με το σύμβολο #:

```
# υπολογίζει το ποσοστό της ώρας που έχει παρέλθει
ποσοστό = (λεπτά * 100) / 60
```

Σε αυτήν την περίπτωση, το σχόλιο εμφανίζεται μόνο του σε μια γραμμή. Μπορείτε επίσης να βάλετε σχόλια στο τέλος μιας γραμμής:

```
ποσοστό = (λεπτά * 100) / 60     # ποσοστό της ώρας
```

Ότι γράψετε από το # έως το τέλος της γραμμής αγνοείται, δεν έχει καμία επίδραση στο πρόγραμμα.

Τα σχόλια είναι πιο χρήσιμα όταν τεκμηριώνουν μη προφανή χαρακτηριστικά του κώδικα. Είναι λογικό να υποθέσουμε ότι ο αναγνώστης μπορεί να καταλάβει *τι* κάνει ο κώδικας. Είναι πολύ πιο χρήσιμο να εξηγήσουμε το *γιατί*.

Αυτό το σχόλιο είναι περιττό και άχρηστο για τον κώδικα:

```
v = 5     # αναθέτει το 5 στο v
```

Αυτό το σχόλιο περιέχει χρήσιμες πληροφορίες που δεν περιέχονται στον κώδικα:

```
v = 5     # ταχύτητα σε μέτρα/δευτερόλεπτο.
```

Η σωστή επιλογή ονομάτων μεταβλητών μπορεί να μειώσει την ανάγκη για σχόλια, αλλά τα μεγάλα ονόματα μπορούν να κάνουν τις σύνθετες εκφράσεις δυσανάγνωστες, οπότε συμβιβαζόμαστε κατά περίπτωση.

Επιλογή μνημονικών ονομάτων μεταβλητών

Εφόσον ακολουθείτε τους απλούς κανόνες ονοματοδοσίας μεταβλητών και αποφεύγετε τις δεσμευμένες λέξεις, έχετε πολλές επιλογές όταν ονομάζετε τις μεταβλητές σας. Στην αρχή, αυτή η επιλογή μπορεί να προκαλέσει σύγχυση, τόσο όταν διαβάζετε ένα πρόγραμμα, όσο και όταν γράφετε τα δικά σας προγράμματα. Για παράδειγμα, τα ακόλουθα τρία προγράμματα είναι πανομοιότυπα ως προς το τι επιτυγχάνουν, αλλά πολύ διαφορετικά όταν τα διαβάζετε και προσπαθείτε να τα καταλάβετε.

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

```
ώρες = 35.0
ωρομίσθιο = 12.50
μισθός = ώρες * ωρομίσθιο
print(μισθός)
```

```
x1q3z9ahd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ahd * x1q3z9afd
print(x1q3p9afd)
```

Ο διερμηνέας της Python βλέπει και τα τρία αυτά προγράμματα *ακριβώς τα ίδια* αλλά οι άνθρωποι βλέπουν και κατανοούν αυτά τα προγράμματα εντελώς διαφορετικά. Οι άνθρωποι θα καταλάβουν πιο γρήγορα την *πρόθεση* του δεύτερου προγράμματος, επειδή ο προγραμματιστής έχει επιλέξει ονόματα μεταβλητών που αντικατοπτρίζουν την πρόθεσή του, σχετικά με τα δεδομένα που θα αποθηκευτούν σε κάθε μεταβλητή.

Ονομάζουμε, αυτά τα σοφά επιλεγμένα ονόματα μεταβλητών, "μνημονικά ονόματα μεταβλητών". Η λέξη *μνημονική*[1] Σημαίνει "βοήθημα μνήμης". Επιλέγουμε μνημονικά ονόματα μεταβλητών για να μας βοηθήσουν να θυμηθούμε γιατί δημιουργήσαμε τη μεταβλητή εξαρχής.

Παρόλο που όλα αυτά ακούγονται υπέροχα και είναι πολύ καλή ιδέα να χρησιμοποιείτε μνημονικά ονόματα μεταβλητών, τα μνημονικά ονόματα μεταβλητών μπορούν να εμποδίσουν την ικανότητα ενός αρχάριου προγραμματιστή να αναλύσει και να κατανοήσει τον κώδικα. Αυτό συμβαίνει επειδή οι αρχάριοι προγραμματιστές δεν έχουν απομνημονεύσει ακόμη τις δεσμευμένες λέξεις (υπάρχουν μόνο 33) και μερικές φορές μεταβλητές με πολύ περιγραφικά ονόματα αρχίζουν να μοιάζουν με μέρος της γλώσσας και όχι μόνο με καλά επιλεγμένα ονόματα μεταβλητών.

Ρίξτε μια γρήγορη ματιά στον ακόλουθο δείγμα κώδικα Python, το οποίο λειτουργεί επαναληπτικά. Θα καλύψουμε τους βρόχους σύντομα, αλλά προς το παρόν προσπαθήστε να κατανοήσετε τι σημαίνει αυτό:

```
for word in words:
    print(word)
```

Τι συμβαίνει εδώ? Ποιες από τις λέξεις (for, word, in, κ.λπ.) είναι δεσμευμένες λέξεις και ποιες είναι απλά ονόματα μεταβλητών; Κατανοεί η Python σε θεμελιώδες επίπεδο την έννοια των λέξεων; Οι αρχάριοι προγραμματιστές έχουν πρόβλημα να διαχωρίσουν ποια μέρη του κώδικα *πρέπει* να παραμείνουν ίδια με αυτό το παράδειγμα και ποια μέρη του κώδικα είναι απλώς επιλογές που γίνονται από τον προγραμματιστή.

---

[1] Βλ. Https://en.wikipedia.org/wiki/Mnemonic για εκτεταμένη περιγραφή της λέξης "μνημονική".

Ο παρακάτω κωδικός είναι ισοδύναμος με τον παραπάνω κωδικό:

```
for slice in pizza:
    print(slice)
```

Είναι ευκολότερο για τον αρχάριο προγραμματιστή να κοιτάξει αυτόν τον κώδικα και να καταλάβει ποια μέρη είναι δεσμευμένες λέξεις, που ορίζονται από την Python και ποια μέρη είναι απλά ονόματα μεταβλητών που επιλέγονται από τον προγραμματιστή. Είναι αρκετά σαφές ότι η Python δεν έχει θεμελιώδη κατανόηση της πίτσας (pizza) και των φετών (slice) και το γεγονός ότι μια πίτσα αποτελείται από ένα σύνολο από μία ή περισσότερες φέτες.

Αλλά αν το πρόγραμμά μας έχει να κάνει πραγματικά για την ανάγνωση δεδομένων και την αναζήτηση λέξεων στα δεδομένα, η «πίτσα» και η «φέτα» είναι πολύ μη μνημονικά ονόματα μεταβλητών. Η επιλογή τους ως ονόματα μεταβλητών αποσπά την προσοχή από το νόημα του προγράμματος.

Μετά από ένα αρκετά σύντομο χρονικό διάστημα, θα γνωρίζετε τις πιο συνηθισμένες δεσμευμένες λέξεις και θα αρχίσετε να βλέπετε τις δεσμευμένες λέξεις να ξεπηδάνε αυθόρμητα από μέσα σας:

```
for word in words:
    print(word)
```

Τα μέρη του κώδικα που ορίζονται από την Python (`for`, `in`, `print` και `:`) είναι με έντονα γράμματα και οι μεταβλητές που επιλέγονται από τον προγραμματιστή (`word` και `words`) δεν είναι έντονες. Πολλοί συντάκτες κειμένου γνωρίζουν τη σύνταξη της Python και θα χρωματίσουν τις δεσμευμένες λέξεις διαφορετικά, για να σας δώσουν κάποιες ενδείξεις, ώστε να διαχωρίσετε τις μεταβλητές από τις δεσμευμένες λέξεις. Μετά από λίγο θα αρχίσετε να διαβάζετε Python και θα προσδιορίζετε γρήγορα τι είναι μεταβλητή και τι είναι δεσμευμένη λέξη.

Εκσφαλμάτωση

Σε αυτό το σημείο, το σφάλμα σύνταξης (syntax error) που πιθανότατα θα κάνετε είναι ένα μη αποδεκτό όνομα μεταβλητής, όπως `class` και `yield`, οι οποίες είναι δεσμευμένες λέξεις, ή `odd~job` και `US$`, που περιέχουν μη αποδεκτούς χαρακτήρες.

Εάν βάλετε ένα κενό σε ένα όνομα μεταβλητής, η Python θα πιστέψει ότι είναι δύο τελεστέοι χωρίς τελεστή:

```
>>> bad name = 5
```

```
SyntaxError: invalid syntax
```

```
>>> month = 09
  File "<stdin>", line 1
    month = 09
              ^
SyntaxError: invalid token
```

Για σφάλματα σύνταξης, τα μηνύματα σφάλματος δεν βοηθούν πολύ. Τα πιο συνηθισμένα μηνύματα είναι `SyntaxError: invalid syntax` και `SyntaxError: invalid token`, κανένα από τα οποία δεν είναι πολύ κατατοπιστικό.

Το σφάλμα χρόνου εκτέλεσης που είναι πιο πιθανό να κάνετε είναι "χρήση πριν από τον ορισμό" δηλαδή, το να προσπαθήσετε να χρησιμοποιήσετε μια μεταβλητή προτού της εκχωρήσετε μια τιμή. Αυτό μπορεί να συμβεί αν γράψετε λάθος ένα όνομα μεταβλητής:

```
>>> principal = 327.68
>>> interest = principle * rate
NameError: name 'principle' is not defined
```

Τα ονόματα των μεταβλητών κάνουν διάκριση πεζών-κεφαλαίων (case sensitive), οπότε το `LaTeX` δεν είναι το ίδιο με το `laTeX`.

Σε αυτό το σημείο, η πιο πιθανή αιτία σημασιολογικού σφάλματος είναι η σειρά των λειτουργιών. Για παράδειγμα, για να αξιολογήσετε το `½ pi`, μπορεί να μπείτε στον πειρασμό να γράψετε

```
>>> 1.0 / 2.0 * pi
```

Αλλά η διαίρεση εκτελείτε πρώτη, οπότε θα πάρετε το `pi/2`, το οποίο δεν είναι το ίδιο! Δεν υπάρχει τρόπος για την Python να γνωρίζει τι θέλατε να γράψετε, οπότε σε αυτήν την περίπτωση δεν λαμβάνετε μήνυμα σφάλματος. Απλά παίρνετε λάθος απάντηση.

Γλωσσάριο

ακέραιος (integer) : Ένας τύπος που αναπαριστά αριθμούς χωρίς δεκαδικό μέρος.

ανάθεση - εκχώρηση τιμής : Μια εντολή που αναθέτει μια τιμή σε μια μεταβλητή.

αξιολόγηση (evaluate) : Το να υπολογίσουμε μια παράσταση, εκτελώντας τις πράξεις, για να αποδώσουμε μια ενιαία τιμή.

έκφραση : Ένας συνδυασμός μεταβλητών, τελεστών και σταθερών που ως

αποτέλεσμα έχει μία και μοναδική τιμή.

εντολή - δήλωση : Ένα τμήμα κώδικα που αντιπροσωπεύει μια εντολή ή ενέργεια. Μέχρι στιγμής, οι εντολές που είδαμε είναι εκχωρήσεις τιμών και εντολές εκτύπωσης εκφράσεων.

δεσμευμένη λέξη : Μία λέξη, η οποία έχει δεσμευθεί από τον μεταγλωττιστής για την εκτέλεση του προγράμματος. Δεν επιτρέπεται η χρήση δεσμευμένων λέξεων όπως `if`, `def` και `while`, ως ονόματα μεταβλητών.

κανόνες προτεραιότητας : Το σύνολο των κανόνων που διέπουν τη σειρά αξιολόγησης εκφράσεων, που περιλαμβάνουν πολλαπλούς τελεστές και τελεστέους.

κινητής υποδιαστολής (floating point) : Ένας τύπος που αναπαριστά αριθμούς με δεκαδικό μέρος.

μεταβλητή : Ένα όνομα που αναφέρεται σε μία τιμή.

μνημονικό : Ένα βοήθημα μνήμης. Συχνά δίνουμε στις μεταβλητές μνημονικά ονόματα για να μας βοηθήσουν να θυμηθούμε τι έχουμε αποθηκεύσει στη μεταβλητή.

συμβολοσειρά : Ένας τύπος που αντιπροσωπεύει ακολουθίες χαρακτήρων.

συνένωση (concatenate) : Η ένωση δύο τελεστέων από άκρο σε άκρο.

σχόλιο : Πληροφορίες σε ένα πρόγραμμα που προορίζονται για άλλους προγραμματιστές (ή οποιονδήποτε διαβάζει τον πηγαίο κώδικα) και δεν έχουν καμία επίδραση στην εκτέλεση του προγράμματος.

τελεστέος : Μία από τις τιμές στις οποίες εφαρμόζεται ένας τελεστής.

τελεστής : Ένα ειδικό σύμβολο που αντιπροσωπεύει έναν απλό υπολογισμό όπως η πρόσθεση, ο πολλαπλασιασμός ή η συνένωση συμβολοσειρών.

τελεστής ακεραίου υπολοίπου (modulus) : Ένας τελεστής, που συμβολίζεται με το σύμβολο ποσοστού (%), λειτουργεί σε ακέραιους αριθμούς και δίνει το υπόλοιπο της ευκλείδειας διαίρεσης του πρώτου αριθμού με τον δεύτερο.

τιμή : Μία από τις βασικές μονάδες δεδομένων, που χειρίζεται ένα πρόγραμμα, όπως ένας αριθμός ή μια συμβολοσειρά.

τύπος : Μια κατηγορία τιμών. Οι τύποι που έχουμε δει μέχρι τώρα είναι ακέραιοι (τύπος `int`), αριθμοί κινητής υποδιαστολής (τύπος`float`) και συμβολοσειρές (τύπος `str`).

Ασκήσεις

Άσκηση 2: Γράψτε ένα πρόγραμμα που χρησιμοποιεί `input` για να ζητά από το χρήστη το όνομά του και στη συνέχεια τον καλωσορίζει.

```
Εισάγετε το όνομά σας: Chuck
Γειά σου Chuck
```

Άσκηση 3: Γράψτε ένα πρόγραμμα που προτρέπει τον χρήστη να εισάγει ώρες και ωρομίσθιο και να υπολογίζει τον ακαθάριστο μισθό του.

```
Εισάγετε τις Ώρες: 35
Εισάγετε το Ωρομίσθιο: 2.75
Μισθός: 96.25
```

Δεν ασχολούμαστε με τι να βεβαιωθούμε ότι η αμοιβή μας έχει ακριβώς δύο ψηφία μετά την υποδιαστολή προς το παρόν. Αν θέλετε, μπορείτε να παίξετε με την ενσωματωμένη συνάρτηση της Python, την `round` για να στρογγυλοποιήσετε σωστά την αμοιβή που προκύπτει, σε δύο δεκαδικά ψηφία.

Άσκηση 4: Ας υποθέσουμε ότι εκτελούμε τις ακόλουθες εντολές εκχώρησης:

```
πλάτος = 17
ύψος = 12.0
```

Για καθεμία από τις παρακάτω εκφράσεις, γράψτε την τιμή της έκφρασης και τον τύπο (της τιμής της έκφρασης).

1. `πλάτος // 2`
2. `πλάτος / 2.0`
3. `ύψος / 3`
4. `1 + 2 * 5`

Χρησιμοποιήστε τον διερμηνέα της Python για να ελέγξετε τις απαντήσεις σας.

Άσκηση 5: Γράψτε ένα πρόγραμμα που ζητά από τον χρήστη μια θερμοκρασία σε βαθμούς Κελσίου, μετατρέψτε τη θερμοκρασία σε βαθμούς Φαρενάιτ και εκτυπώστε την θερμοκρασία που προκύπτει.

Κεφάλαιο 3

Δομή Επιλογής

Λογικές εκφράσεις

Μια *λογική έκφραση* είναι μια παράσταση, η τιμή της οποίας είναι είτε αληθής/true είτε ψευδής/false. Τα επόμενα παραδείγματα χρησιμοποιούν τον τελεστή ==, ο οποίος συγκρίνει δύο τελεστέους και επιστρέφει `True` αν είναι ίσοι και `False` σε διαφορετική περίπτωση:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

`True` και `False` είναι ειδικές τιμές που ανήκουν στην κλάση `bool`, δεν είναι συμβολοσειρές:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Ο τελεστής == είναι ένας από τους *συγκριτικούς τελεστές*, οι υπόλοιποι είναι:

```
x != y               # το x δεν είναι ίσο με το y
x > y                # το x είναι μεγαλύτερο του y
x < y                # το x είναι μικρότερο του y
x >= y               # το x είναι μεγαλύτερο ή ίσο του y
x <= y               # το x είναι μικρότερο ή ίσο του y
x is y               # το x είναι ίδιο με το y
x is not y           # το x δεν είναι ίδιο με το y
```

Αν και αυτές οι πράξεις σας είναι πιθανώς γνωστές, τα σύμβολα της Python διαφέρουν από τα αντίστοιχα μαθηματικά σύμβολα. Ένα συνηθισμένο λάθος είναι το να χρησιμοποιήσετε ένα μόνο σύμβολο ίσου (=) αντί για ένα διπλό ίσο (==). Θυμηθείτε ότι το = είναι τελεστής εκχώρησης τιμής και το == είναι τελεστής σύγκρισης. Δεν έχουν νόημα τα =< ή =>.

Λογικοί τελεστές

Υπάρχουν τρεις *λογικοί τελεστές*: `and`, `or` και `not`. Η σημασιολογία (έννοια) αυτών των τελεστών είναι παρόμοια με τη σημασία τους στα αγγλικά (και, ή και όχι αντίστοιχα). Για παράδειγμα το

```
x > 0 and x < 10
```

είναι αληθές μόνο αν το `x` είναι μεγαλύτερο του 0 *και* μικρότερο του 10.

Το `n%2 == 0 or n%3 == 0` είναι αληθές αν *τουλάχιστον μία* από τις συνθήκες είναι αληθής, δηλαδή αν ο αριθμός διαιρείται με το 2 *ή* το 3.

Τέλος, ο τελεστής `not` είναι η άρνηση μιας λογικής έκφρασης, έτσι το `not (x > y)` είναι αληθές αν το `x > y` είναι ψευδές, δηλαδή αν το `x` είναι μικρότερο ή ίσο του `y`.

Αυστηρά μιλώντας, οι τελεστέοι των λογικών τελεστών πρέπει να είναι λογικές εκφράσεις, αλλά η Python δεν είναι πολύ αυστηρή. Κάθε μη μηδενικός αριθμός ερμηνεύεται ως "αληθές".

```
>>> 17 and True
True
```

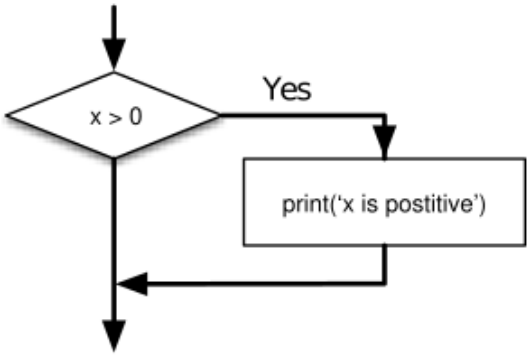
Αυτή η ευελιξία μπορεί να είναι χρήσιμη, αλλά υπάρχουν κάποιες λεπτομέρειες σε αυτό, που μπορεί να προκαλέσουν σύγχυση. Ίσως θα ήταν καλύτερα να το αποφύγετε μέχρι να είστε σίγουροι ότι ξέρετε τι κάνετε.

Απλή επιλογή

Για να γράψουμε χρήσιμα προγράμματα, χρειαζόμαστε σχεδόν πάντα τη δυνατότητα να ελέγξουμε τις συνθήκες και να αλλάξουμε ανάλογα τη συμπεριφορά του προγράμματος. Οι *εντολές επιλογής* μας δίνουν αυτή τη δυνατότητα. Η πιο απλή μορφή τους είναι η εντολή `if`:

```
if x > 0 :
    print('το x είναι θετικό')
```

Η λογική έκφραση που ακολουθεί την εντολή `if` ονομάζεται *συνθήκη*. Τελειώνουμε την γραμμή της εντολής `if` με τον χαρακτήρα άνω κάτω τελεία (:) και στη γραμμή(ές) μετά το if δημιουργούμε εσοχή.



Εικόνα 3.1: if Λογικό Διάγραμμα

Εάν η λογική συνθήκη είναι αληθής, τότε οι εντολές με εσοχή εκτελούνται. Εάν η λογική συνθήκη είναι ψευδής, οι εντολές με εσοχή παραλείπονται.

Η εντολή `if` έχει την ίδια δομή με τους ορισμούς συνάρτησης ή τους βρόχους `for`[1]. Η εντολή αποτελείται από μια γραμμή κεφαλίδας που τελειώνει με την άνω και κάτω τελεία (:) ακολουθούμενη από ένα μπλοκ εντολών με εσοχή. Τέτοιες εντολές ονομάζονται *σύνθετες εντολές* επειδή εκτείνονται σε περισσότερες από μία γραμμές.

Δεν υπάρχει όριο στον αριθμό των εντολών που μπορούν να εμφανιστούν στο μπλοκ των εντολών, αλλά πρέπει να υπάρχει τουλάχιστον μία. Περιστασιακά, είναι χρήσιμο να έχετε ένα μπλοκ χωρίς εντολές (συνήθως ως δέσμευση θέσης για κάποιον κώδικα, που δεν έχετε γράψει ακόμα). Σε αυτή την περίπτωση, μπορείτε να χρησιμοποιήσετε τη εντολή `pass`, η οποία δεν κάνει τίποτα.

```
if x < 0 :
    pass          # πρέπει να χειριστώ τις αρνητικές τιμές!
```

Εάν εισαγάγετε μια εντολή `if` στον διερμηνέα της Python, η προτροπή θα αλλάξει από >>> σε τρεις τελείες για να υποδείξει ότι βρίσκεστε στη μέση ενός μπλοκ δηλώσεων, όπως φαίνεται παρακάτω:

```
>>> x = 3
>>> if x < 10:
...     print('Μικρό')
...
Μικρό
>>>
```

[1] Θα μάθουμε για τις συναρτήσεις στο Κεφάλαιο 4 και τους βρόχους στο Κεφάλαιο 5.

Όταν χρησιμοποιείτε τον διερμηνέα της Python, πρέπει να αφήσετε μια κενή γραμμή στο τέλος του μπλοκ, διαφορετικά η Python θα επιστρέψει σφάλμα:

```
>>> x = 3
>>> if x < 10:
...     print('Μικρό')
... print('Τέλος')
  File "<stdin>", line 3
    print('Τέλος')
        ^
SyntaxError: invalid syntax
```

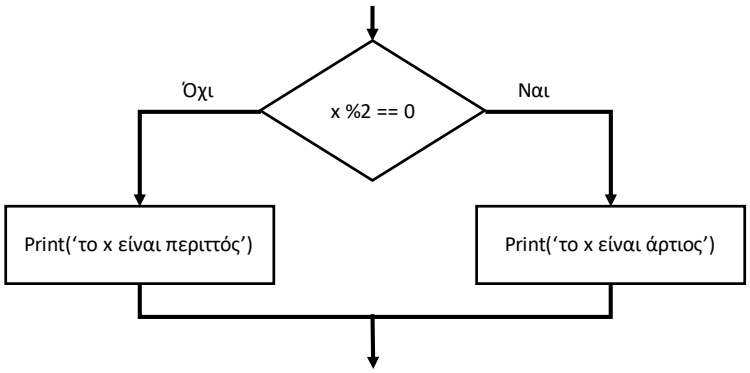
Η κενή γραμμή στο τέλος ενός μπλοκ δηλώσεων δεν είναι απαραίτητη όταν γράφετε και εκτελείτε σενάριο, μπορεί όμως να βελτιώσει την αναγνωσιμότητα του κώδικα σας.

Σύνθετη επιλογή

Η δεύτερη μορφή της εντολής `if` είναι η *σύνθετη επιλογή*, στην οποία υπάρχουν δύο περιπτώσεις και η συνθήκη καθορίζει ποια θα εκτελεστεί. Η σύνταξη μοιάζει με αυτήν:

```
if x%2 == 0 :
    print('το x είναι άρτιος')
else :
    print('το x είναι περιττός')
```

Εάν το υπόλοιπο, όταν το x διαιρεθεί με το 2 είναι 0, τότε ξέρουμε ότι το x είναι άρτιος και το πρόγραμμα εμφανίζει σχετικό μήνυμα. Εάν η συνθήκη είναι ψευδής εκτελείτε το δεύτερο μπλοκ εντολών.



Εικόνα 3.2: If-Then-Else Λογικό Διάγραμμα

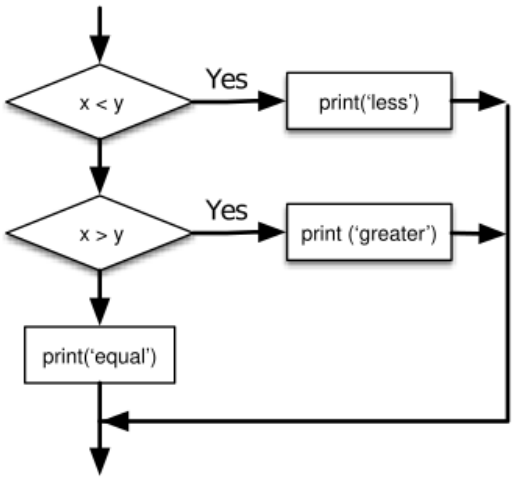
Δεδομένου ότι η συνθήκη πρέπει να είναι αληθής ή ψευδής, θα εκτελεστεί ακριβώς μία από τις εναλλακτικές περιπτώσεις. Οι εναλλακτικές περιπτώσεις ονομάζονται *κλάδοι*, επειδή αποτελούν διακλαδώσεις στη ροή εκτέλεσης.

Πολλαπλή επιλογή

Μερικές φορές υπάρχουν περισσότερες από δύο δυνατότητες και χρειαζόμαστε περισσότερους από δύο κλάδους. Ένας τρόπος για να εκφράσετε έναν τέτοιο υπολογισμό είναι μια *πολλαπλή επιλογή*:

```
if x < y:
    print('το x είναι μικρότερο από το y')
elif x > y:
    print('το x είναι μεγαλύτερο από το y')
else:
    print('τα x και y είναι ίσα')
```

Το `elif` είναι μια συντομογραφία του "else if". Και πάλι, θα εκτελεστεί ακριβώς ένας κλάδος.



Εικόνα 3.3: If-Then-ElseIf Λογικό Διάγραμμα

Δεν υπάρχει όριο στον αριθμό των δηλώσεων `elif`. Εάν υπάρχει ο όρος `else`, πρέπει να είναι στο τέλος, αλλά δεν είναι απαραίτητο να υπάρχει.

```
if choice == 'a':
    print('Μάντεψες Λάθος')
elif choice == 'b':
    print('Μάντεψες Σωστά')
elif choice == 'c':
    print('Πλησίασες, αλλά όχι σωστό')
```

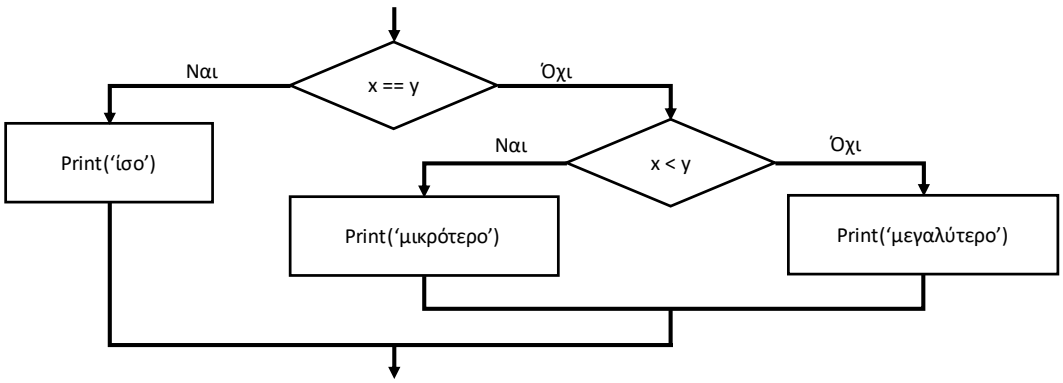
Κάθε συνθήκη ελέγχεται με τη σειρά. Εάν η πρώτη είναι ψευδής, ελέγχεται η επόμενη και ούτω καθεξής. Εάν μία από αυτές είναι αληθής, ο αντίστοιχος κλάδος εκτελείται και η εντολή τελειώνει. Ακόμα κι αν περισσότερες από μία συνθήκες είναι αληθείς, εκτελείται μόνο ο πρώτος αληθής κλάδος.

Εμφωλευμένη επιλογή

Μια εντολή επιλογής μπορεί να εμφωλευτεί σε άλλη. Θα μπορούσαμε να γράψουμε το παράδειγμα των τριών-κλάδων και έτσι:

```
if x == y:
    print('τα x και y είναι ίσα')
else:
    if x < y:
        print('το x είναι μικρότερο από το y')
    else:
        print('το x είναι μεγαλύτερο από το y')
```

Η εξωτερική επιλογή περιέχει δύο κλάδους. Ο πρώτος κλάδος περιέχει μια απλή εντολή. Ο δεύτερος κλάδος περιέχει άλλη μια εντολή `if`, η οποία έχει δύο δικούς της κλάδους. Αυτοί οι δύο κλάδοι περιέχουν και οι δύο απλές εντολές, αν και θα μπορούσαν να ήταν και νέες εντολές επιλογής.



Εικόνα 3.4: Εμφωλευμένες Εντολές If

Αν και η χρήση εσοχών καθιστά εμφανή τη δομή, οι *εμφωλευμένες επιλογές* είναι δύσκολο να διαβαστούν πολύ γρήγορα. Γενικά, είναι καλή ιδέα να τις αποφεύγετε όταν μπορείτε.

Οι λογικοί τελεστές παρέχουν συχνά έναν τρόπο απλοποίησης των εμφωλευμένων επιλογών. Για παράδειγμα, μπορούμε να ξαναγράψουμε τον ακόλουθο κώδικα χρησιμοποιώντας μία μόνο εντολή επιλογής:

```
if 0 < x:
    if x < 10:
        print('το x είναι ένας θετικός μονοψήφιος αριθμός.')
```

Η εντολή print εκτελείται μόνο αν ικανοποιούνται και οι δύο συνθήκες, οπότε μπορούμε να έχουμε το ίδιο αποτέλεσμα χρησιμοποιώντας τον τελεστή and:

```
if 0 < x and x < 10:
    print('το x είναι ένας θετικός μονοψήφιος αριθμός.')
```

Εντοπισμός εξαιρέσεων χρησιμοποιώντας το try και except

Νωρίτερα είδαμε ένα τμήμα κώδικα όπου χρησιμοποιήσαμε τις συναρτήσεις input και int για να διαβάσουμε και να αναλύσουμε έναν ακέραιο αριθμό που εισήγαγε ο χρήστης. Είδαμε επίσης πόσο εύκολα μπορεί να οδηγήσει σε σφάλματα:

```
>>> μήνυμα = "Με τί ταχύτητα πετάει ένα χελιδόνι;\n"
>>> ταχύτητα = input(μήνυμα)
Με τί ταχύτητα πετάει ένα χελιδόνι;
Τί εννοείς, ένα Αφρικανικό ή Ευρωπαϊκό χελιδόνι;
>>> int(ταχύτητα)
ValueError: invalid literal for int() with base 10:
>>>
```

Όταν εκτελούμε αυτές τις εντολές στον διερμηνέα της Python, λαμβάνουμε ένα νέο μήνυμα από τον διερμηνέα, σκεφτόμαστε "ωχ" και προχωράμε στην επόμενη εντολή μας.

Ωστόσο, εάν γράψετε αυτόν τον κώδικα σε ένα σενάριο Python και παρουσιαστεί αυτό το σφάλμα, το σενάριό σας σταματά αμέσως γιατί εντοπίζει το πρόβλημα με ιχνηλάτηση/traceback. Δεν εκτελεί την ακόλουθη πρόταση.

Ακολουθεί ένα δείγμα προγράμματος για τη μετατροπή θερμοκρασίας Φαρενάιτ σε θερμοκρασία Κελσίου:

```
inp = input('Εισαγάγετε τη θερμοκρασία Φαρενάιτ: ')
fahr = float(inp)
```

```
cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)
```

#Κώδικας στο /code3/fahren.py

Εάν εκτελέσουμε αυτόν τον κώδικα και του δώσουμε μη έγκυρη είσοδο, απλώς αποτυγχάνει με ένα όχι και τόσο φιλικό μήνυμα σφάλματος:

```
python fahren.py
Εισαγάγετε τη θερμοκρασία Φαρενάιτ:72
22.22222222222222
```

```
python fahren.py
Εισαγάγετε τη θερμοκρασία Φαρενάιτ:fred
Traceback (most recent call last):
  File "fahren.py", line 2, in <module>
    fahr = float(inp)
ValueError: could not convert string to float: 'fred'
```

Υπάρχει μια δομή εκτέλεσης υπό όρους ενσωματωμένη στην Python για να χειρίζεται αυτούς τους τύπους αναμενόμενων και/ή απροσδόκητων σφαλμάτων που ονομάζονται "try / except". Η βάση των `try` και `except` είναι ότι γνωρίζετε ότι κάποια ακολουθία οδηγιών μπορεί να προκαλέσει πρόβλημα και θέλετε να προσθέσετε ορισμένες προτάσεις που θα εκτελεστούν σε περίπτωση σφάλματος. Αυτές οι πρόσθετες προτάσεις (το μπλοκ except) αγνοούνται εάν δεν προκληθεί σφάλμα.

Μπορείτε να θεωρήσετε τη λειτουργία `try` και `except` στην Python ως μια «δικλείδα ασφαλείας» για μια ακολουθία εντολών.

Μπορούμε να ξαναγράψουμε τον μετατροπέα θερμοκρασίας μας ως εξής:

```
inp = input('Εισαγάγετε τη θερμοκρασία Φαρενάιτ:')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print(cel)
except:
    print('Παρακαλώ εισάγετε έναν αριθμό')
```

#Code: http://www.gr.py4e.com/code3/fahren2.py

Η Python ξεκινά εκτελώντας την ακολουθία εντολών που περιέχονται στο μπλοκ `try`.

Αν όλα πάνε καλά, παραλείπει το μπλοκ except και προχωρά. Εάν προκύψει πρόβλημα στο μπλοκ try, η Python βγαίνει από το μπλοκ try και εκτελεί την ακολουθία εντολών του μπλοκ except.

```
python fahren2.py
Εισαγάγετε τη θερμοκρασία Φαρενάιτ:72
22.22222222222222
```

```
python fahren2.py
Εισαγάγετε τη θερμοκρασία Φαρενάιτ:fred
Παρακαλώ εισάγετε έναν αριθμό
```

Η διαχείριση μιας εξαίρεσης με την εντολή try ονομάζεται *σύλληψη* της εξαίρεσης. Σε αυτό το παράδειγμα, ο όρος except εκτυπώνει ένα φιλικό μήνυμα σφάλματος. Σε γενικές γραμμές, η σύλληψη μιας εξαίρεσης σας δίνει την ευκαιρία να διορθώσετε το πρόβλημα ή να προσπαθήσετε ξανά ή τουλάχιστον να το πρόγραμμα τερματίσει χωρίς μήνυμα σφάλματος.

Ελαχιστοποίηση αξιολόγησης λογικών εκφράσεων

Όταν η Python επεξεργάζεται μια λογική έκφραση όπως η `x >= 2 and (x/y) > 2`, αξιολογεί την έκφραση από αριστερά προς τα δεξιά. Λόγω του ορισμού του and, αν το x είναι μικρότερο του 2, η έκφραση `x >= 2` είναι `False` και έτσι συνολικά η έκφραση είναι `False` ανεξάρτητα από το αν το `(x/y) > 2` έχει ως αποτέλεσμα `True` ή `False`.

Όταν η Python διαπιστώσει ότι δεν θα να κερδίσει τίποτα με την αξιολόγηση της υπόλοιπης λογικής έκφρασης, σταματά την αξιολόγησή της και δεν κάνει τους υπολογισμούς στην υπόλοιπη λογική έκφραση. Όταν σταματήσει η αξιολόγηση μιας λογικής έκφρασης επειδή η συνολική τιμή είναι ήδη γνωστή, ονομάζεται *short-circuiting/βραχυκύκλωμα* της αξιολόγηση.

Ενώ αυτό μπορεί να φαίνεται σαν μια καλή τεχνική, η συμπεριφορά βραχυκυκλώματος οδηγεί σε μια εξυπνότερη τεχνική που ονομάζεται *τιμή φρουρός/guardian pattern*. Εξετάστε το ακόλουθο τμήμα κώδικα στον διερμηνέα της Python:

```
>>> x = 6
>>> y = 2
>>> x >= 2 and (x/y) > 2
True
>>> x = 1
```

```
>>> y = 0
>>> x >= 2 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and (x/y) > 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

Ο τρίτος υπολογισμός απέτυχε επειδή η Python αξιολόγησε το `(x/y)` και το `y` ήταν μηδέν, γεγονός που προκάλεσε σφάλμα χρόνου εκτέλεσης. Αλλά το πρώτο και το δεύτερο παράδειγμα *δεν* απέτυχαν, επειδή στον πρώτο υπολογισμό το `y` ήταν μηδέν και στο δεύτερο, το πρώτο μέρος της έκφρασης, το `x >= 2` αξιολογήθηκε ως `False` οπότε το `(x /y)` δεν εκτελέστηκε ποτέ λόγω του κανόνα *short-circuiting* και δεν προέκυψε σφάλμα.

Μπορούμε να κατασκευάσουμε τη λογική έκφραση με τέτοιο τρόπο ώστε να τοποθετήσουμε στρατηγικά έναν *φύλακα* αξιολόγηση, ακριβώς πριν από την αξιολόγηση που μπορεί να προκαλέσει σφάλμα ως εξής:

```
>>> x = 1
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x >= 2 and (x/y) > 2 and y != 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

Στην πρώτη λογική έκφραση, το `x >= 2` είναι `False`, έτσι η αξιολόγηση σταματά στο `and`. Στη δεύτερη λογική έκφραση, το `x >= 2` είναι `True` αλλά το `y != 0` είναι `False`

οπότε δεν φτάνουμε ποτέ στο `(x/y)`.

Στην τρίτη λογική έκφραση, το `y != 0` βρίσκεται *μετά* τον υπολογισμό `(x/y)` έτσι η έκφραση αποτυγχάνει με συνέπεια ένα σφάλμα.

Στη δεύτερη έκφραση, λέμε ότι το `y != 0` λειτουργεί ως *φρουρός* για να διασφαλίσει ότι εκτελούμε το `(x/y)` μόνο αν το y είναι μη μηδενικό.

Εκσφαλμάτωση

Το traceback της Python εμφανίζεται όταν προκύψει ένα σφάλμα και περιέχει πολλές πληροφορίες, αλλά αυτές μπορεί να είναι υπερβολικές. Τα πιο χρήσιμα μέρη είναι συνήθως τα:

- Τι είδους λάθος ήταν και
- Πού συνέβη.

Τα σφάλματα σύνταξης είναι συνήθως εύκολο να βρεθούν, αλλά υπάρχουν μερικές παγίδες. Τα σφάλματα στους λευκούς χαρακτήρες μπορεί να είναι δύσκολα επειδή τα κενά και τα tab είναι αόρατα και έχουμε συνηθίσει να τα αγνοούμε.

```
>>> x = 5
>>>  y = 6
  File "<stdin>", line 1
    y = 6
    ^
IndentationError: unexpected indent
```

Σε αυτό το παράδειγμα, το πρόβλημα είναι ότι στη δεύτερη γραμμή έχει δημιουργηθεί εσοχή ενός διαστήματος. Αλλά το μήνυμα σφάλματος δείχνει το y, το οποίο είναι παραπλανητικό. Σε γενικές γραμμές, τα μηνύματα σφάλματος υποδεικνύουν πού εντοπίστηκε το πρόβλημα, αλλά το πραγματικό σφάλμα μπορεί να είναι νωρίτερα στον κώδικα, μερικές φορές σε προηγούμενη γραμμή.

Σε γενικές γραμμές, τα μηνύματα σφάλματος σας λένε πού ανακαλύφθηκε το πρόβλημα, αλλά συχνά δεν προκλήθηκε εκεί.

Γλωσσάριο

traceback : Μια λίστα με τις λειτουργίες που εκτελούνται, που εκτυπώνονται όταν προκύψει ένα σφάλμα.

short circuit - βραχυκύκλωμα : Όταν η Python αξιολογεί εν μέρει μια λογική έκφραση και σταματά την αξιολόγηση επειδή γνωρίζει την τελική τιμή για την έκφραση χωρίς να χρειάζεται να αξιολογήσει την υπόλοιπη έκφραση.

εμφωλευμένη επιλογή : Μια εντολή επιλογής που εμφανίζεται σε έναν από τους κλάδους μιας άλλης εντολής επιλογής.

εντολή επιλογής : Μια εντολή που διαφοροποιεί την ροή της εκτέλεσης ανάλογα με κάποια συνθήκη.

κλάδος : Μία από τις εναλλακτικές ακολουθίες εντολών σε μια εντολή επιλογής.

λογική έκφραση : Μια έκφραση της οποίας η τιμή είναι είτε `True` είτε `False`.

λογικός τελεστής : Ένας από τους τελεστές που συνδυάζουν λογικές εκφράσεις: `and`, `or` και `not`.

πολλαπλή επιλογή : Μια εντολή επιλογής με μια σειρά εναλλακτικών κλάδων.

σύνθετη εντολή/compound statement : Μια εντολή που αποτελείτε από κεφαλίδα και σώμα. Η κεφαλίδα τελειώνει με άνω κάτω τελεία (:). Το σώμα τοποθετείται σε εσοχή, σε σχέση με την κεφαλίδα

συνθήκη : Η λογική έκφραση σε μια εντολή επιλογής, που καθορίζει ποιος κλάδος θα εκτελεστεί.

συγκριτικός τελεστής : Ένας από τους τελεστές που συγκρίνουν τους τελεστέους τους: ==, !=, >, <, >=, and <=.

σώμα : Το μπλοκ των εντολών μέσα σε μια σύνθετη εντολή.

τιμή φρουρός : Όπου κατασκευάζουμε μια λογική έκφραση με πρόσθετες συγκρίσεις για να επωφεληθούμε από τη συμπεριφορά short-circuit/βραχυκυκλώματος.

Ασκήσεις

Άσκηση 1: Ξαναγράψτε τον υπολογισμό της αμοιβής για να δώσετε στον υπάλληλο 1,5 φορές το ωρομίσθιο για τις ώρες εργασίας πέραν των 40 ωρών.

```
Δώστε Ώρες: 45
Δώστε Ποσό/Ώρα: 10
Μισθός: 475.0
```

Άσκηση 2: Ξαναγράψτε το πρόγραμμα πληρωμών χρησιμοποιώντας το `try` και

except έτσι ώστε το πρόγραμμά σας να χειρίζεται μη αριθμητικές τιμές εισόδου σωστά, εκτυπώνοντας ένα μήνυμα και τερματίζοντας την εκτέλεση. Παρακάτω φαίνεται το αποτέλεσμα δύο εκτελέσεων του προγράμματος:

```
Δώστε Ώρες: 20
Δώστε Ποσό/Ώρα: εννιά
Σφάλμα, παρακαλώ δώστε αριθμητική είσοδο
```

```
Δώστε Ώρες: σαράντα
Σφάλμα, παρακαλώ δώστε αριθμητική είσοδο
```

Άσκηση 3: Γράψτε ένα πρόγραμμα για να ζητήσετε βαθμολογία μεταξύ 0,0 και 1,0. Εάν η βαθμολογία είναι εκτός εμβέλειας, να εκτυπώνετε ένα μήνυμα σφάλματος. Εάν η βαθμολογία είναι μεταξύ 0.0 και 1.0, να εκτυπώνετε μια αξιολόγηση με βάση τον ακόλουθο πίνακα:

```
Βαθμός   Αξιολόγηση
>= 0.9     A
>= 0.8     B
>= 0.7     C
>= 0.6     D
 < 0.6     F
```

```
Εισάγετε βαθμολογία: 0.95
A
```

```
Εισάγετε βαθμολογία: τέλεια
Άκυρη βαθμολογία
```

```
Εισάγετε βαθμολογία: 10.0
Άκυρη βαθμολογία
```

```
Εισάγετε βαθμολογία: 0.75
C
```

```
Εισάγετε βαθμολογία: 0.5
F
```

Εκτελέστε το πρόγραμμα επανειλημμένα όπως φαίνεται παραπάνω για να το δοκιμάσετε για τις διαφορετικές τιμές εισόδου.

Κεφάλαιο 4

Συναρτήσεις

Κλήση συναρτήσεων

Στο πλαίσιο του προγραμματισμού, μια *συνάρτηση* είναι μια ομάδα εντολών, που τους έχει αποδοθεί ένα όνομα και εκτελούν έναν υπολογισμό. Όταν ορίζετε μια συνάρτηση, καθορίζετε το όνομα και τη ακολουθία των εντολών. Αργότερα, μπορείτε να "καλέσετε" τη συνάρτηση με το όνομά της. Έχουμε ήδη δει ένα παράδειγμα κλήσης *συνάρτησης*:

```
>>> type(32)
<class 'int'>
```

Το όνομα της συνάρτησης είναι `type`. Η έκφραση στην παρένθεση καλείται *όρισμα* της συνάρτησης. Το όρισμα είναι μια τιμή ή μεταβλητή που μεταβιβάζουμε στη συνάρτηση ως είσοδό της. Το αποτέλεσμα, της συνάρτησης `type`, είναι ο τύπος του ορίσματος.

Συνηθίζουμε να λέμε ότι μια συνάρτηση "δέχεται" ένα όρισμα και "επιστρέφει" ένα αποτέλεσμα. Το αποτέλεσμα ονομάζεται *τιμή επιστροφής*.

Ενσωματωμένες συναρτήσεις

Η Python παρέχει μια σειρά σημαντικών ενσωματωμένων συναρτήσεων που μπορούμε να χρησιμοποιήσουμε χωρίς να χρειαστεί να τις ορίσουμε. Οι δημιουργοί της Python έγραψαν ένα σύνολο λειτουργιών για την επίλυση κοινών προβλημάτων και τις συμπεριέλαβαν σε αυτήν για να τις χρησιμοποιήσουμε.

Οι συναρτήσεις `max` και `min` μας δίνουν την μεγαλύτερη και την μικρότερη τιμή μιας λίστας, αντίστοιχα:

```
>>> max('Γειά σου κόσμε')
'ό'
>>> min('Γειά σου κόσμε')
' '
>>>
```

Η συνάρτηση `max` μας λέει τον "μεγαλύτερο χαρακτήρα" της συμβολοσειράς (που αποδεικνύεται ότι είναι το γράμμα "ό") και η συνάρτηση `min` μας δείχνει τον μικρότερο χαρακτήρα (που αποδεικνύεται ότι είναι το κενό).

Μια άλλη πολύ συνηθισμένη ενσωματωμένη συνάρτηση είναι η συνάρτηση `len`, που μας λέει πόσα στοιχεία υπάρχουν στο όρισμα της. Εάν το όρισμα στη `len` είναι μια συμβολοσειρά, επιστρέφει τον αριθμό των χαρακτήρων στη συμβολοσειρά.

```
>>> len('Γειά σου κόσμε')
14
>>>
```

Αυτές οι συναρτήσεις δεν περιορίζονται στην εξέταση συμβολοσειρών. Μπορούν να λειτουργήσουν σε οποιοδήποτε σύνολο τιμών, όπως θα δούμε σε επόμενα κεφάλαια.

Θα πρέπει να αντιμετωπίζετε τα ονόματα των ενσωματωμένων συναρτήσεων ως δεσμευμένες λέξεις (δηλαδή, αποφύγετε τη χρήση του "max" ως όνομα μεταβλητής).

Συναρτήσεις μετατροπής τύπου

Η Python παρέχει επίσης ενσωματωμένες συναρτήσεις που μετατρέπουν τιμές από τον ένα τύπο στον άλλο. Η συνάρτηση `int` παίρνει οποιαδήποτε τιμή και τη μετατρέπει σε ακέραιο, αν μπορεί, ή διαφορετικά παραπονιέται:

```
>>> int('32')
32
>>> int('Γειά')
ValueError: invalid literal for int() with base 10: 'Γειά'
```

Η `int` μπορεί να μετατρέψει τιμές κινητής υποδιαστολής (floating-point) σε ακεραίους, αλλά δεν τις στρογγυλοποιεί, απλά αποκόπτει το δεκαδικό μέρος:

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

Η `float` μετατρέπει ακεραίους και συμβολοσειρές σε αριθμούς κινητής υποδιαστολής:

```
>>> float(32)
32.0
```

```
>>> float('3.14159')
3.14159
```

Τέλος, η `str` μετατρέπει το όρισμά της σε μια συμβολοσειρά:

```
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

Μαθηματικές συναρτήσεις - Math

Η Python διαθέτει το άρθρωμα (module) `math` που περιέχει τις περισσότερες από τις γνωστές μαθηματικές συναρτήσεις. Πριν μπορέσουμε να χρησιμοποιήσουμε το άρθρωμα, πρέπει να το εισαγάγουμε:

```
>>> import math
```

Αυτή η εντολή δημιουργεί ένα *αντικείμενο αρθρώματος - module object*, που ονομάζεται math. Εάν εκτυπώσετε το αντικείμενο αρθρώματος, θα λάβετε μερικές πληροφορίες σχετικά με αυτό:

```
>>> print(math)
<module 'math' (built-in)>
```

Το αντικείμενο του αρθρώματος περιέχει τις συναρτήσεις και τις μεταβλητές που ορίζονται στο module. Για να αποκτήσετε πρόσβαση σε μία από τις συναρτήσεις, πρέπει να καθορίσετε το όνομα του module και το όνομα της συνάρτησης, χωρισμένα με μια τελεία (dot notation).

```
>>> λόγος = ισχύς_σήματος / ισχύς_θορύβου
>>> decibels = 10 * math.log10(λόγος)


>>> ακτίνια = 0.7
>>> ύψος = math.sin(ακτίνια)
```

Το πρώτο παράδειγμα υπολογίζει τον λογάριθμο με βάση 10 του λόγου σήματος-θορύβου. Το module math παρέχει επίσης μια συνάρτηση που ονομάζεται `log` και υπολογίζει το νεπέριο λογάριθμο, λογάριθμο δηλαδή με βάση το e.

Το δεύτερο παράδειγμα βρίσκει το ημίτονο των `ακτινίων`. Το όνομα της μεταβλητής είναι μια υπόδειξη ότι το `sin` και οι άλλες τριγωνομετρικές συναρτήσεις (`cos`, `tan` κ.λπ.)

δέχονται ορίσματα σε ακτίνια. Για να μετατρέψετε από μοίρες σε ακτίνια, διαιρέστε με 360 και πολλαπλασιάστε με 2π:

```
>>> μοίρες = 45
>>> ακτίνια = μοίρες / 360.0 * 2 * math.pi
>>> math.sin(ακτίνια)
0.7071067811865476
```

Η έκφραση `math.pi` λαμβάνει τη μεταβλητή π από την ενότητα μαθηματικών. Η τιμή αυτής της μεταβλητής είναι μια προσέγγιση του π, με ακρίβεια περίπου 15 ψηφίων.

Εάν γνωρίζετε τριγωνομετρία, μπορείτε να ελέγξετε το προηγούμενο αποτέλεσμα συγκρίνοντάς το με την τετραγωνική ρίζα του δύο, δια του δύο:

```
>>> math.sqrt(2) / 2.0
0.7071067811865476
```

Τυχαίος αριθμός

Με τις ίδιες εισόδους, τα περισσότερα προγράμματα υπολογιστών παράγουν τις ίδιες εξόδους κάθε φορά, επομένως λέμε ότι είναι *αιτιοκρατικά*. Η αιτιοκρατία (ντετερμινισμός) είναι συνήθως καλό πράγμα, αφού περιμένουμε ότι ο ίδιος υπολογισμός θα δώσει το ίδιο αποτέλεσμα. Για ορισμένες εφαρμογές, ωστόσο, θέλουμε ο υπολογιστής να είναι απρόβλεπτος. Τα παιχνίδια είναι ένα προφανές παράδειγμα, αλλά υπάρχουν περισσότερα.

Το να κάνετε ένα πρόγραμμα πραγματικά μη ντετερμινιστικό αποδεικνύεται ότι δεν είναι τόσο εύκολο, αλλά υπάρχουν τρόποι να το κάνετε τουλάχιστον να φαίνεται μη ντετερμινιστικό. Ένας από αυτούς είναι η χρήση *αλγορίθμων* που δημιουργούν *ψευδοτυχαίους* αριθμούς. Οι ψευδοτυχαίοι αριθμοί δεν είναι πραγματικά τυχαίοι επειδή παράγονται από έναν ντετερμινιστικό υπολογισμό, αλλά κοιτάζοντας απλά τους αριθμούς αυτούς είναι αδύνατο να διακριθούν από τους τυχαίους.

Το άρθρωμα (module) `random` παρέχει συναρτήσεις που δημιουργούν ψευδοτυχαίους αριθμούς (τους οποίους θα αποκαλώ απλώς "τυχαίους" από εδώ και πέρα).

Η συνάρτηση `random` επιστρέφει έναν τυχαίο αριθμό κινητής υποδιαστολής μεταξύ των 0,0 και 1,0 (συμπεριλαμβανομένου του 0.0 αλλά όχι του 1.0). Κάθε φορά που καλείται την `random`, παίρνετε τον επόμενο σε σειρά αριθμό μιας μεγάλης ακολουθίας. Για να δείτε ένα παράδειγμα εκτελέστε τον παρακάτω βρόχο:

```
import random

for i in range(10):
    x = random.random()
    print(x)
```

Αυτό το πρόγραμμα παράγει την ακόλουθη λίστα 10 τυχαίων αριθμών μεταξύ 0,0 και 1,0, αλλά χωρίς να περιλαμβάνεται το 1,0.

```
0.11132867921152356
0.5950949227890241
0.04820265884996877
0.841003109276478
0.997914947094958
0.04842330803368111
0.7416295948208405
0.510535245390327
0.27447040171978143
0.028511805472785867
```

Άσκηση 1: Εκτελέστε το πρόγραμμα στο σύστημά σας και δείτε ποιοι αριθμοί παράγονται. Εκτελέστε το πρόγραμμα περισσότερες από μία φορές και δείτε ποιοι αριθμοί παράγονται.

Η συνάρτηση `random` είναι μόνο μία από τις πολλές συναρτήσεις που χειρίζονται τυχαίους αριθμούς. Η συνάρτηση `randint` δέχεται τις παραμέτρους `low` και `high` και επιστέφει έναν ακέραιο μεταξύ των `low` και `high` (συμπεριλαμβανομένων και αυτών).

```
>>> random.randint(5, 10)
5
>>> random.randint(5, 10)
9
```

Για να επιλέξετε τυχαία ένα στοιχείο μιας σειράς μπορείτε να χρησιμοποιήσετε την `choice`:

```
>>> t = [1, 2, 3]
>>> random.choice(t)
2
```

```
>>> random.choice(t)
3
```

Το άρθρωμα random μας παρέχει και συναρτήσεις για τη δημιουργία τυχαίων τιμών από συνεχείς κατανομές, όπως ή κανονική κατανομή, η εκθετική, η γάμμα και μερικές ακόμη.

Προσθήκη νέων συναρτήσεων

Μέχρι στιγμής, χρησιμοποιούσαμε μόνο συναρτήσεις ενσωματωμένες στην Python, αλλά είναι επίσης δυνατή η προσθήκη και νέων συναρτήσεων. Ο *ορισμός συνάρτησης* καθορίζει το όνομα της νέας συνάρτησης και την ακολουθία των εντολών που εκτελούνται όταν κληθεί ή συνάρτηση. Μόλις ορίσουμε μια συνάρτηση, μπορούμε να την επαναχρησιμοποιήσουμε, ξανά και ξανά, σε όλο το πρόγραμμά μας.

Ιδού και ένα παράδειγμα:

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')
```

def είναι η δεσμευμένη λέξη που υποδηλώνει τον ορισμό μιας συνάρτησης. Το όνομα της συνάρτησης είναι print_lyrics. Οι κανόνες ονοματολογίας των συναρτήσεων είναι οι ίδιοι με αυτούς για τα ονόματα μεταβλητών: γράμματα, ψηφία και κάποια σημεία στίξης επιτρέπονται, αλλά ο πρώτος χαρακτήρας δεν μπορεί να είναι ψηφίο. Δεν μπορείτε να χρησιμοποιήσετε δεσμευμένη λέξη ως όνομα συνάρτησης και πρέπει να αποφεύγετε τη χρήση μεταβλητής και συνάρτησης με το ίδιο όνομα.

Οι κενές παρενθέσεις μετά το όνομα σημαίνουν ότι η συγκεκριμένη συνάρτηση δεν δέχεται κάποιο όρισμα. Αργότερα, θα κατασκευάσουμε συναρτήσεις που δέχονται ορίσματα ως είσοδο.

Η πρώτη γραμμή της συνάρτησης, κατά τον ορισμό της, καλείται *επικεφαλίδα*, οι υπόλοιπες αποτελούν το *σώμα*. Η επικεφαλίδα πρέπει να καταλήγει σε άνω κάτω τελεία και το σώμα πρέπει να τοποθετείται σε εσοχή. Κατά σύμβαση, η εσοχή είναι πάντοτε τέσσερα κενά. Το σώμα μπορεί να περιέχει οποιοδήποτε πλήθος δηλώσεων.

Αν ορίσετε μια συνάρτηση σε interactive mode, ο διερμηνευτής εκτυπώνει ellipses (...) στην επόμενη γραμμή για να σας ενημερώσει ότι ο ορισμός δεν έχει ολοκληρωθεί:

```
>>> def print_lyrics():
...     print("I'm a lumberjack, and I'm okay.")
```

```
...     print('I sleep all night and I work all day.')
...
```

Για να ολοκληρώσετε τη συνάρτηση πρέπει να εισάγετε μια κενή γραμμή (αυτό δεν είναι απαραίτητο σε ένα σενάριο).

Ο ορισμός μιας συνάρτησης δημιουργεί μια μεταβλητή με το ίδιο όνομα.

```
>>> print(print_lyrics)
<function print_lyrics at 0xb7e99e9c>
>>> print(type(print_lyrics))
<class 'function'>
```

Η τιμή του `print_lyrics` είναι ένα *αντικείμενο function (συνάρτηση)*, το οποίο είναι του τύπου "function".

Η σύνταξη κλήσης της νέας συνάρτησης είναι ίδια όπως και των ενσωματωμένων συναρτήσεων:

```
>>> print_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Αφού ορίσετε μια συνάρτηση, μπορείτε να τη χρησιμοποιήσετε μέσα σε μια άλλη συνάρτηση. Για παράδειγμα, για να επαναλάβουμε το προηγούμενο ρεφρέν, θα μπορούσαμε να γράψουμε μια συνάρτηση που ονομάζεται `repeat_lyrics`:

```
def repeat_lyrics():
    print_lyrics()
    print_lyrics()
```

Και στη συνέχεια να καλέσουμε την `repeat_lyrics`:

```
>>> repeat_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Αλλά στην πραγματικότητα δεν πάει έτσι το τραγούδι.

Ορισμός και χρήση

Συνδυάζοντας τα τμήματα κώδικα από την προηγούμενη ενότητα, ολοκληρωμένο το πρόγραμμα μοιάζει κάπως έτσι:

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')

def repeat_lyrics():
    print_lyrics()
    print_lyrics()

repeat_lyrics()
```

#Code: http://www.gr.py4e.com/code3/lyrics.py

Αυτό το πρόγραμμα περιέχει δύο ορισμούς συναρτήσεων: `print_lyrics` και `repeat_lyrics`. Οι ορισμοί των συναρτήσεων εκτελούνται όπως και κάθε άλλη εντολή, αλλά το αποτέλεσμά τους είναι η δημιουργία αντικειμένων συνάρτησης. Οι εντολές που περιέχονται σε μία συνάρτηση δεν εκτελούνται έως ότου κληθεί η συνάρτηση και ο ορισμός της συνάρτησης δεν παράγει έξοδο.

Όπως θα περίμενε κανείς, πρέπει να δημιουργήσετε μια συνάρτηση πριν την εκτελέσετε. Με άλλα λόγια, ο ορισμός της συνάρτησης πρέπει να εκτελεστεί πριν από την πρώτη φορά που θα κληθεί.

Άσκηση 2: Μετακινήστε την τελευταία γραμμή αυτού του προγράμματος στην αρχή του, έτσι ώστε η κλήση της συνάρτησης να εμφανίζεται πριν από τον ορισμό της. Εκτελέστε το πρόγραμμα και δείτε ποιο μήνυμα λάθους λαμβάνετε.

Άσκηση 3: Μετακινήστε την κλήση συνάρτησης και πάλι στο κάτω μέρος καθώς και τον ορισμό του `print_lyrics` μετά τον ορισμό του `repeat_lyrics`. Τι συμβαίνει όταν εκτελείτε αυτό το πρόγραμμα;

Ροή εκτέλεσης

Για να διασφαλίσετε ότι μια συνάρτηση ορίζεται πριν από την πρώτη της χρήση, πρέπει να γνωρίζετε τη σειρά με την οποία εκτελούνται οι εντολές, η σειρά αυτή ονομάζεται *ροή εκτέλεσης*.

Η εκτέλεση αρχίζει πάντα με την πρώτη εντολή του προγράμματος. Οι εντολές

εκτελούνται μία κάθε φορά, με σειρά από πάνω προς τα κάτω.

Οι *ορισμοί* συναρτήσεων δεν αλλάζουν τη ροή εκτέλεσης του προγράμματος, αλλά να θυμάστε ότι οι εντολές που περιλαμβάνονται στη συνάρτηση δεν εκτελούνται μέχρι να κληθεί η συνάρτηση.

Μια κλήση συνάρτησης είναι σαν μια παράκαμψη στη ροή της εκτέλεσης. Αντί να μεταβεί στην επόμενη πρόταση, η ροή μεταβαίνει στο σώμα της συνάρτησης, εκτελεί όλες τις εντολές εκεί και μετά επιστρέφει για να συνεχίσει από εκεί που σταμάτησε.

Αυτό ακούγεται αρκετά απλό, έως ότου θυμηθείτε ότι μια συνάρτηση μπορεί να καλέσει μια άλλη. Ενώ βρίσκεται στη μέση μιας συνάρτησης, το πρόγραμμα μπορεί να χρειαστεί να εκτελέσει τις εντολές μιας άλλης συνάρτησης. Αλλά και κατά την εκτέλεση αυτής της νέας συνάρτησης, το πρόγραμμα μπορεί να χρειαστεί να εκτελέσει ακόμη μια συνάρτηση!

Ευτυχώς, η Python είναι καλή στο να παρακολουθεί πού βρίσκεται η ροή εκτέλεσης του προγράμματος, επομένως κάθε φορά που ολοκληρώνεται μια συνάρτηση, το πρόγραμμα συνεχίζει από εκεί που σταμάτησε στη συνάρτηση όπου πραγματοποιήθηκε η κλήση. Όταν η ροή εκτέλεσης φτάσει στο τέλος του προγράμματος, αυτό τερματίζει.

Ποιο είναι το ηθικό δίδαγμα αυτής της ατυχής ιστορίας; Όταν διαβάζετε ένα πρόγραμμα, δεν είναι πάντα καλό το να διαβάζετε από πάνω προς τα κάτω. Μερικές φορές είναι προτιμότερο να ακολουθείτε τη ροή της εκτέλεσης.

Παράμετροι και ορίσματα

Ορισμένες από τις ενσωματωμένες συναρτήσεις που έχουμε δει απαιτούν ορίσματα. Για παράδειγμα, όταν καλείτε την `math.sin` περνάτε έναν αριθμό ως όρισμα. Ορισμένες συναρτήσεις λαμβάνουν περισσότερα από ένα όρισμα: η `math.pow` δέχεται δύο, τη βάση και τον εκθέτη.

Μέσα στη συνάρτηση, τα ορίσματα εκχωρούνται σε μεταβλητές που ονομάζονται *παράμετροι*. Ακολουθεί ένα παράδειγμα συνάρτησης που ορίζεται από το χρήστη που δέχεται ένα όρισμα:

```
def print_twice(bruce):
    print(bruce)
    print(bruce)
```

Αυτή η συνάρτηση εκχωρεί το όρισμα σε μια παράμετρο που ονομάζεται `bruce`. Όταν

καλείται η συνάρτηση, εκτυπώνει την τιμή της παραμέτρου (όποια και αν είναι) δύο φορές.

Η συνάρτηση αυτή λειτουργεί με οποιαδήποτε τιμή μπορεί να εκτυπωθεί.

```
>>> print_twice('Spam')
Spam
Spam
>>> print_twice(17)
17
17
>>> import math
>>> print_twice(math.pi)
3.141592653589793
3.141592653589793
```

Οι ίδιοι κανόνες σύνταξης που ισχύουν για τις ενσωματωμένες συναρτήσεις ισχύουν και για τις συναρτήσεις που ορίζονται από το χρήστη, επομένως μπορούμε να χρησιμοποιήσουμε οποιοδήποτε είδος έκφρασης ως όρισμα για το `print_twice`:

```
>>> print_twice('Spam '*4)
Spam Spam Spam Spam
Spam Spam Spam Spam
>>> print_twice(math.cos(math.pi))
-1.0
-1.0
```

Το όρισμα αξιολογείται πριν από την κλήση της συνάρτησης, επομένως στα παραδείγματα οι εκφράσεις `'Spam' *4` και `math.cos(math.pi)` αξιολογούνται μόνο μία φορά.

Μπορείτε επίσης να χρησιμοποιήσετε ως όρισμα μια μεταβλητή:

```
>>> michael = 'Eric, the half a bee.'
>>> print_twice(michael)
Eric, the half a bee.
Eric, the half a bee.
```

Το όνομα της μεταβλητής, που δίνουμε ως όρισμα (`michael`) δεν έχει καμία σχέση με το όνομα της παραμέτρου (`bruce`). Δεν έχει σημασία τι όνομα είχε τη τιμή στο προηγούμενο πρόγραμμα (σε αυτό από το οποίο έγινε η κλήση), εδώ στην `print_twice` αποκαλείται `bruce`.

Γόνιμες και κενές συναρτήσεις

Κάποιες από τις συναρτήσεις που χρησιμοποιούμε, όπως οι μαθηματικές συναρτήσεις (math) επιστρέφουν αποτελέσματα. Λόγω έλλειψης κάποιου καλύτερου ονόματος, τις αποκαλώ *γόνιμες συναρτήσεις* (fruitful functions). Άλλες συναρτήσεις, όπως η `print_twice`, εκτελούν μια λειτουργία αλλά δεν επιστρέφουν κάποια τιμή. Αυτές τις αποκαλώ *κενές συναρτήσεις* (void functions).

Όταν καλείτε μία γόνιμη συνάρτηση, σχεδόν πάντα θέλετε να κάνετε κάτι με το αποτέλεσμα της. Για παράδειγμα, ίσως θα το εκχωρήσετε σε μία μεταβλητή ή θα το χρησιμοποιήσετε ως μέρος μιας έκφρασης:

```
x = math.cos(radians)
golden = (math.sqrt(5) + 1) / 2
```

Όταν καλείτε μια συνάρτηση σε interactive mode, η Python εμφανίζει το αποτέλεσμα:

```
>>> math.sqrt(5)
2.23606797749979
```

Αλλά σε ένα σενάριο, αν καλέσετε μια γόνιμη συνάρτηση χωρίς να την εκχωρήσετε σε μια μεταβλητή, η επιστρεφόμενη τιμή εξαφανίζεται στην ομίχλη!

```
math.sqrt(5)
```

Αυτό το σενάριο υπολογίζει την τετραγωνική ρίζα του 5, αλλά μιας και δεν αποθηκεύει το αποτέλεσμα σε κάποια μεταβλητή, ούτε εμφανίζει το αποτέλεσμα, δεν είναι πολύ χρήσιμο.

Οι κενές (void) συναρτήσεις μπορεί να εμφανίζουν κάτι στην οθόνη ή να έχουν κάποιο άλλο αποτέλεσμα, αλλά δεν έχουν επιστρεφόμενη τιμή. Αν προσπαθήσετε να εκχωρήσετε το αποτέλεσμα σε μία μεταβλητή, θα πάρετε μια ειδική τιμή την ονομαζόμενη `None`.

```
>>> result = print_twice('Bing')
Bing
Bing
>>> print(result)
None
```

Η τιμή `None` δεν είναι ίδια με τη συμβολοσειρά "None". Είναι μια ειδική τιμή η οποία έχει τον δικό της τύπο:

```
>>> print(type(None))
<class 'NoneType'>
```

Για να επιστρέψουμε ένα αποτέλεσμα από τη συνάρτηση, χρησιμοποιούμε την εντολή `return` στο σώμα της συνάρτησης. Για παράδειγμα, θα μπορούσαμε να κατασκευάσουμε μια πολύ απλή συνάρτηση με όνομα `addtwo`, η οποία προσθέτει δύο αριθμούς και επιστρέφει το αποτέλεσμα.

```
def addtwo(a, b):
    added = a + b
    return added
x = addtwo(3, 5)
print(x)
```

#Code: http://www.gr.py4e.com/code3/addtwo.py

Όταν το παραπάνω σενάριο εκτελείτε, η εντολή `print` θα εκτυπώσει το "8" γιατί η συνάρτηση `addtwo` κλήθηκε με ορίσματα το 3 και το 5. Μέσα στη συνάρτηση, οι παράμετροι `a` και `b` ήταν 3 και 5 αντίστοιχα. Η συνάρτηση υπολογίζει το άθροισμα των δύο αριθμών και το τοποθετεί στην τοπική μεταβλητή της συνάρτησης με όνομα `added`. Έπειτα χρησιμοποιεί την εντολή `return` για να στείλει την υπολογισμένη τιμή πίσω στον καλούντα κώδικα, ως αποτέλεσμα της συνάρτησης, όπου είχε εκχωρηθεί στην μεταβλητή `x` και την εκτυπώνει.

Γιατί συναρτήσεις;

Ίσως να μην έγινε σαφής ο λόγος για τον οποίο αξίζει να χωρίζουμε ένα πρόγραμμα σε συναρτήσεις. Υπάρχουν αρκετοί λόγοι:

- Η δημιουργία μιας νέας συνάρτησης σας δίνει την δυνατότητα να ονομάσετε ένα σύνολο δηλώσεων, γεγονός το οποίο κάνει το πρόγραμμα ευκολότερο στην ανάγνωση και στην εκσφαλμάτωση.
- Οι συναρτήσεις μπορούν να ελαττώσουν το μέγεθος ενός προγράμματος καταργώντας τον επαναλαμβανόμενο κώδικα. Αργότερα, αν κάνετε κάποια αλλαγή, αρκεί να την κάνετε μόνο σε ένα σημείο.
- Η διαίρεση ενός μεγάλου προγράμματος σε συναρτήσεις σας επιτρέπει να εκσφαλματώσετε τα τμήματά του ένα ένα και στη συνέχεια να τα ενσωματώσετε όλα σε ένα ολοκληρωμένο λειτουργικό πρόγραμμα.

- Οι καλοσχεδιασμένες συναρτήσεις είναι συχνά χρήσιμες για πολλά προγράμματα. Μόλις γράψετε και εκσφαλματώσετε μία, μπορείτε να την χρησιμοποιήσετε.

Σε όλο το υπόλοιπο βιβλίο, συχνά θα ορίζουμε μια συνάρτηση για να εξηγήσουμε μια έννοια. Μέρος της ικανότητας δημιουργίας και χρήσης συναρτήσεων είναι να δημιουργείτε μια συνάρτηση που να αποτυπώνει σωστά μια ιδέα όπως "βρείτε τη μικρότερη τιμή σε μια λίστα τιμών". Αργότερα θα σας δείξουμε κώδικα που βρίσκει τη μικρότερη από μια λίστα τιμών και θα σας τον παρουσιάσουμε ως μια συνάρτηση με το όνομα `min`, η οποία παίρνει μια λίστα τιμών ως όρισμα και επιστρέφει τη μικρότερη τιμή στη λίστα.

Εκσφαλμάτωση

Αν χρησιμοποιείτε έναν επεξεργαστή κειμένου για να γράψετε τα σενάριά σας, ίσως αντιμετωπίσετε προβλήματα με τα κενά διαστήματα και τους στηλοθέτες (tabs). Ο καλύτερος τρόπος για να αποφύγετε αυτού του είδους προβλήματα είναι να χρησιμοποιείτε αποκλειστικά κενά διαστήματα (όχι στηλοθέτες). Οι περισσότεροι επεξεργαστές κειμένου που είναι συμβατοί με την Python το κάνουν αυτό εξ ορισμού, αλλά κάποιοι άλλοι όχι.

Οι στηλοθέτες και τα κενά διαστήματα είναι συνήθως μη ορατά, κάτι το οποίο κάνει δύσκολη την εκσφαλμάτωση, οπότε προσπαθήστε να βρείτε έναν επεξεργαστή που να μπορεί να διαχειριστεί την ενδοπαραγραφοποίηση.

Επίσης, μην ξεχνάτε να αποθηκεύετε το πρόγραμμά σας πριν το εκτελέσετε. Κάποια προγραμματιστικά περιβάλλοντα το κάνουν αυτόματα, αλλά κάποια άλλα όχι. Σε αυτήν την περίπτωση, το πρόγραμμα που βλέπετε στον επεξεργαστή κειμένου δεν είναι το ίδιο με το πρόγραμμα που εκτελείτε.

Η εκσφαλμάτωση μπορεί να γίνει χρονοβόρα αν εκτελείτε το ίδιο εσφαλμένο πρόγραμμα ξανά και ξανά!

Σιγουρευτείτε πως ο κώδικας που κοιτάτε είναι ο κώδικας που εκτελείτε. Αν δεν είστε σίγουροι, γράψτε κάτι όπως το `print("hello")` στην αρχή του προγράμματος και εκτελέστε το ξανά. Αν δεν δείτε `hello`, δεν εκτελείτε το σωστό πρόγραμμα!

Γλωσσάριο

dot notation (χωρισμός με τελεία) : Η σύνταξη για την κλήση μιας συνάρτησης, που περιέχεται σε ένα άρθρωμα, καθορίζοντας το όνομα του αρθρώματος ακολουθούμενο από μια τελεία και το όνομα της συνάρτησης.

αιτιοκρατισμός (deterministic) : Αφορά ένα πρόγραμμα που κάνει το ίδιο πράγμα κάθε φορά που εκτελείται, με τις ίδιες εισόδους.

αλγόριθμος : Μια γενική μέθοδος για την επίλυση μιας κατηγορίας προβλημάτων.

αντικείμενο αρθρώματος (module) : Μια τιμή που δημιουργείται από μια εντολή `import` και παρέχει πρόσβαση στα δεδομένα και τον κώδικα που ορίζονται σε ένα άρθρωμα (module).

αντικείμενο συνάρτηση : Μια τιμή που δημιουργείται από τον ορισμό μιας συνάρτησης. Το όνομα της συνάρτησης είναι μια μεταβλητή που αναφέρεται σε ένα αντικείμενο συνάρτησης.

γόνιμη συνάρτηση (fruitful function) : Μια συνάρτηση που επιστρέφει μια τιμή.

εντολή import : Μια εντολή που διαβάζει ένα αρχείο αρθρώματος και δημιουργεί ένα αντικείμενο αρθρώματος.

επικεφαλίδα : Η πρώτη γραμμή του ορισμού μιας συνάρτησης.

κενή συνάρτηση (void function) : Μια συνάρτηση που δεν επιστρέφει τιμή.

κλήση συνάρτησης : Μια δήλωση που εκτελεί μια συνάρτηση. Αποτελείται από το όνομα της συνάρτησης που ακολουθείται από μια λίστα ορισμάτων.

ορισμός συνάρτησης : Μια δήλωση που δημιουργεί μια νέα συνάρτηση, προσδιορίζοντας το όνομά της, τις παραμέτρους και τις εντολές που εκτελεί.

όρισμα : Μια τιμή που παρέχεται σε μια συνάρτηση όταν η συνάρτηση καλείται. Αυτή η τιμή αποδίδεται στην αντίστοιχη παράμετρο στη συνάρτηση.

παράμετρος : Ένα όνομα (μεταβλητής) που χρησιμοποιείται μέσα σε μια συνάρτηση για να αναφερθούμε στην τιμή που δόθηκε ως όρισμα κατά την κλήση της.

ροή εκτέλεσης : Η σειρά με την οποία εκτελούνται οι εντολές κατά την εκτέλεση ενός προγράμματος.

συνάρτηση : Μια επώνυμη ακολουθία εντολών που εκτελεί κάποια χρήσιμη λειτουργία. Οι συναρτήσεις μπορεί να δέχονται, ή και όχι, ορίσματα και μπορεί να

παράγουν ή να μην παράγουν αποτέλεσμα.

σύνθεση (composition) : Η χρήση μιας έκφρασης ως μέρος μιας μεγαλύτερης έκφρασης, ή μιας δήλωσης ως μέρος μιας μεγαλύτερης δήλωσης.

σώμα : Η ακολουθία των εντολών μέσα σε έναν ορισμό συνάρτησης.

τιμή επιστροφής : Το αποτέλεσμα μιας συνάρτησης. Αν η κλήση της συνάρτησης χρησιμοποιηθεί ως έκφραση, η τιμή επιστροφής είναι η τιμή της έκφρασης.

ψευδοτυχαίο : Αφορά μια ακολουθία αριθμών που φαίνεται να είναι τυχαίοι, αλλά παράγονται από ένα ντετερμινιστικό (αιτιοκρατικό) πρόγραμμα.

Ασκήσεις

Άσκηση 4: Ποιος είναι ο σκοπός της δεσμευμένης λέξης "def" στην Python;

a) Είναι αργκό που σημαίνει "ο παρακάτω κώδικας είναι πολύ ωραίος"
b) Υποδεικνύει την αρχή μιας συνάρτησης
c) Υποδεικνύει ότι η ενότητα κώδικα που ακολουθεί σε εσοχή πρόκειται να αποθηκευτεί για αργότερα
d) Ισχύουν το b και το c
e) Κανένα από τα παραπάνω

Άσκηση 5: Τι θα εκτυπώσει το παρακάτω πρόγραμμα Python;

```
def fred():
   print("Zap")

def jane():
   print("ABC")

jane()
fred()
jane()
```

a) Zap ABC jane fred jane
b) Zap ABC Zap
c) ABC Zap jane
d) ABC Zap ABC
e) Zap Zap Zap

Άσκηση 6: Ξαναγράψτε τον υπολογισμό του ακαθάριστου μισθού (Άσκηση 1 του 3ου Κεφαλαίου) και δημιουργήστε μια συνάρτηση με όνομα `computepay` η οποία δέχεται δύο παραμέτρους (`hours` και `rate`).

```
Δώστε Ώρες: 45
Δώστε Ποσό/Ώρα: 10
Μισθός: 475.0
```

Άσκηση 7: Ξαναγράψτε το πρόγραμμα βαθμολόγησης του προηγούμενου κεφαλαίου, χρησιμοποιώντας τώρα μια συνάρτηση με όνομα `computegrade` η οποία δέχεται έναν βαθμό ως παράμετρο και επιστρέφει την αντίστοιχη αξιολόγηση ως string.

```
Βαθμός  Αξιολόγηση
>= 0.9     A
>= 0.8     B
>= 0.7     C
>= 0.6     D
 < 0.6     F
```

```
Εισάγετε βαθμολογία: 0.95
A
```

```
Εισάγετε βαθμολογία: τέλεια
Άκυρη βαθμολογία
```

```
Εισάγετε βαθμολογία: 10.0
Άκυρη βαθμολογία
```

```
Εισάγετε βαθμολογία: 0.75
C
```

```
Εισάγετε βαθμολογία: 0.5
F
```

Εκτελέστε το πρόγραμμα επανειλημμένα όπως φαίνεται παραπάνω για να το δοκιμάσετε για τις διαφορετικές τιμές εισόδου.

Κεφάλαιο 5

Δομή Επανάληψης

Ενημέρωση μεταβλητών

Ένα συνηθισμένο μοτίβο στις εντολές εκχώρησης είναι μια εντολή εκχώρησης που ενημερώνει μια μεταβλητή και η νέα τιμή της μεταβλητής εξαρτάται από την παλιά.

```
x = x + 1
```

Αυτό σημαίνει "πάρε την τρέχουσα τιμή του x, πρόσθεσε 1 και μετά ενημερώστε το x με τη νέα τιμή."

Εάν προσπαθήσετε να ενημερώσετε μια μεταβλητή που δεν υπάρχει, λαμβάνετε ένα σφάλμα, επειδή η Python αξιολογεί το δεξί μέλος πριν εκχωρήσει μια τιμή στο x:

```
>>> x = x + 1
NameError: name 'x' is not defined
```

Πριν μπορέσετε να ενημερώσετε μια μεταβλητή, θα πρέπει να την *αρχικοποιήσετε*, συνήθως με μια απλή εκχώρηση:

```
>>> x = 0
>>> x = x + 1
```

Η ενημέρωση μιας μεταβλητής προσθέτοντας 1 ονομάζεται *αύξηση (increment)*, ενώ η αφαίρεση του 1 ονομάζεται *μείωση (decrement)*.

Η εντολή `while`

Οι υπολογιστές χρησιμοποιούνται συχνά για την αυτοματοποίηση επαναλαμβανόμενων εργασιών. Η επανάληψη ίδιων ή παρόμοιων εργασιών χωρίς σφάλματα είναι κάτι που οι υπολογιστές κάνουν καλά και οι άνθρωποι όχι και τόσο καλά. Επειδή η επανάληψη είναι τόσο συνηθισμένη, η Python παρέχει πολλές γλωσσικές δυνατότητες για να το διευκολύνει.

Μια εντολή επανάληψης στην Python είναι η εντολή `while`. Εδώ έχουμε ένα απλό πρόγραμμα που μετρά αντίστροφα από το πέντε και μετά λέει "Blastoff!".

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('Blastoff!')
```

Μπορείτε σχεδόν να διαβάσετε τη δήλωση `while` σαν να ήταν Αγγλικά. Σημαίνει, "Όσο το `n` είναι μεγαλύτερο από 0, εμφανίστε την τιμή του `n` και στη έπειτα μειώστε την τιμή του `n` κατά 1. Όταν φτάσει στο 0, βγες από την εντολή `while` και εμφανίστε τη λέξη `Blastoff` !"

Πιο επίσημα, αυτή είναι η ροή της εκτέλεσης για μια εντολή `while`:

1. Αξιολογήστε τη συνθήκη, δίνοντας `True (Αληθής)` ή `False (Ψευδής)`.
2. Εάν η συνθήκη είναι ψευδής, βγες από την εντολή `while` και συνέχισε την εκτέλεση με την επόμενη εντολή.
3. Εάν η συνθήκη είναι αληθής, εκτέλεσε τις εντολές μέσα στη `while` και στη συνέχεια πήγαινε πίσω στο βήμα 1.

Αυτός ο τύπος ροής ονομάζεται *βρόχος (loop)* επειδή το τρίτο βήμα επαναφέρει την εκτέλεση και πάλι στην αρχή. Καλούμε, κάθε εκτέλεση του σώματος του βρόχου, *επανάληψη*. Για τον παραπάνω βρόχο, θα λέγαμε, "Πραγματοποιεί πέντε επαναλήψεις", που σημαίνει ότι το σώμα του βρόχου εκτελέστηκε πέντε φορές.

Στο σώμα του βρόχου θα πρέπει να αλλάξει την τιμή μιας ή περισσότερων μεταβλητών έτσι ώστε τελικά η συνθήκη να γίνει ψευδής και ο βρόχος να τερματιστεί. Καλούμε τη μεταβλητή που αλλάζει κάθε φορά που εκτελείται ο βρόχος και ελέγχει πότε θα τερματίσει ο βρόχος *μεταβλητή επανάληψης*. Εάν δεν υπάρχει μεταβλητή επανάληψης, ο βρόχος θα επαναλαμβάνεται επ' άπειρον, με αποτέλεσμα έναν *ατέρμων βρόχο*.

Ατέρμονες βρόχοι

Μια ατελείωτη πηγή διασκέδασης για τους προγραμματιστές είναι η παρατήρηση ότι οι οδηγίες στα σαμπουάν, "Σαπουνίστε, ξεπλύνετε, επαναλάβετε", είναι ένας ατέρμων βρόχος, επειδή δεν υπάρχει *μεταβλητή επανάληψης* που να σας λέει πόσες φορές να εκτελέσετε τον βρόχο.

Στην περίπτωση της `αντίστροφης μέτρησης`, μπορούμε να αποδείξουμε ότι ο βρόχος τερματίζεται επειδή γνωρίζουμε ότι η τιμή του `n` είναι πεπερασμένη και μπορούμε να

δούμε ότι η τιμή του n ελαττώνεται κάθε φορά που εκτελείται ο βρόχος, οπότε τελικά πρέπει να φτάσει στο 0. Άλλες φορές ένας βρόχος είναι προφανώς άπειρος επειδή δεν έχει καμία μεταβλητή επανάληψης.

Μερικές φορές δεν γνωρίζεται ότι είναι ώρα να τερματίσετε έναν βρόχο παρά μόνο όταν έχετε φτάσει στα μισά του σώματος. Σε αυτήν την περίπτωση, μπορείτε να γράψετε εσκεμμένα έναν ατέρμων βρόχο και στη συνέχεια να χρησιμοποιήσετε την εντολή `break` για να βγείτε από τον βρόχο.

Αυτός ο βρόχος είναι προφανώς ένας *ατέρμων βρόχος* επειδή η λογική έκφραση στην `while` είναι απλώς η λογική σταθερά `True`:

```
n = 10
while True:
    print(n, end=' ')
    n = n - 1
print('Τέλος!')
```

Εάν κάνετε το λάθος και εκτελέσετε αυτόν τον κώδικα, θα μάθετε γρήγορα πώς να διακόψετε μια εκτρεπόμενη διαδικασία Python στο σύστημά σας ή θα βρείτε πού βρίσκεται το κουμπί απενεργοποίησης στον υπολογιστή σας. Αυτό το πρόγραμμα θα λειτουργεί επ' άπειρον ή τουλάχιστον μέχρι να εξαντληθεί η μπαταρία σας επειδή η λογική έκφραση στην αρχή του βρόχου είναι πάντα αληθής, λόγω του ότι η έκφραση είναι η σταθερή τιμή `True`.

Αν και πρόκειται για έναν δυσλειτουργικό ατέρμων βρόχο, μπορούμε να χρησιμοποιήσουμε αυτό το μοτίβο για να δημιουργήσουμε χρήσιμους βρόχους, αρκεί να προσθέσουμε προσεκτικά κώδικα στο σώμα του βρόχου για να βγούμε ρητά από τον βρόχο χρησιμοποιώντας `break` όταν έχουμε φτάσει στην κατάσταση εξόδου.

Για παράδειγμα, ας υποθέσουμε ότι θέλετε να λαμβάνετε είσοδο από τον χρήστη μέχρι να πληκτρολογήσει `τέλος`. Θα μπορούσατε να γράψετε:

```
while True:
    γραμμή = input('> ')
    if γραμμή == 'τέλος':
        break
    print(γραμμή)
print('Τέλος!')
```

#Code: http://www.gr.py4e.com/code3/copytildone1.py

Η συνθήκη βρόχου είναι `True`, η οποία είναι πάντα αληθής, επομένως ο βρόχος εκτελείται επανειλημμένα μέχρι να συναντήσει στην εντολή break.

Κάθε φορά που εκτελείται, προτρέπει τον χρήστη με ένα σύμβολο μεγαλύτερου. Εάν ο χρήστης πληκτρολογήσει `τέλος`, η εντολή `break` διακόπτει την εκτέλεση του βρόχου. Διαφορετικά, το πρόγραμμα επαναλαμβάνει ό,τι πληκτρολογεί ο χρήστης και επιστρέφει στην αρχή του βρόχου. Εδώ είναι ένα δείγμα εκτέλεσης:

```
> hello there
hello there
> finished
finished
> τέλος
Τέλος!
```

Αυτός ο τρόπος γραφής των βρόχων `while` είναι συνηθισμένος επειδή μπορείτε να ελέγξετε τη συνθήκη οπουδήποτε στον βρόχο (όχι μόνο στην αρχή) και μπορείτε να εκφράσετε τη συνθήκη διακοπής καταφατικά ("σταμάτα όταν συμβεί αυτό") και όχι αρνητικά ("συνέχισε μέχρι να συμβεί αυτό.").

Ολοκλήρωση επαναλήψεων με continue

Μερικές φορές βρίσκεστε σε μια επανάληψη ενός βρόχου και θέλετε να ολοκληρώσετε την τρέχουσα εκτέλεσή της και να μεταβείτε άμεσα στην επόμενη εκτέλεση. Σε αυτήν την περίπτωση, μπορείτε να χρησιμοποιήσετε την εντολή `continue` για να μεταβείτε στην επόμενη εκτέλεση χωρίς να ολοκληρώσετε το σώμα του βρόχου για την τρέχουσα εκτέλεση.

Ακολουθεί ένα παράδειγμα βρόχου που αντιγράφει την είσοδό του έως ότου ο χρήστης πληκτρολογήσει `τέλος`, αλλά αντιμετωπίζει τις γραμμές που ξεκινούν με τον χαρακτήρα # (Αριθμός - hash), ως γραμμές που δεν πρέπει να εκτυπωθούν (κάπως σαν σχόλια της Python).

```
while True:
    γραμμή = input('> ')
    if γραμμή[0] == '#':
        continue
    if γραμμή == 'τέλος':
        break
```

```
    print(γραμμή)
print('Τέλος!')
```

#Code: http://www.gr.py4e.com/code3/copytildone2.py

Ακολουθεί ένα δείγμα εκτέλεσης αυτού του νέου προγράμματος με την προσθήκη του `continue`.

```
> hello there
hello there
> # don't print this
> print this!
print this!
> τέλος
Τέλος!
```

Όλες οι γραμμές εκτυπώνονται εκτός από αυτή που ξεκινά με το σύμβολο αριθμού, επειδή όταν εκτελεστεί η `continue`, τερματίζει την τρέχουσα εκτέλεση και μεταπηδά πίσω στην αρχή της εντολής `while` για να ξεκινήσει η επόμενη εκτέλεση, παρακάμπτοντας έτσι την εντολή `print`.

Ορισμός βρόχων με χρήση της for

Μερικές φορές θέλουμε να κάνουμε να διατρέξουμε ένα *σύνολο* πραγμάτων, όπως μια λίστα λέξεων, τις γραμμές ενός αρχείου ή μια λίστα αριθμών. Όταν έχουμε να διατρέξουμε μια λίστα πραγμάτων, μπορούμε να κατασκευάσουμε έναν βρόχο *καθορισμένο* χρησιμοποιώντας μια εντολή `for`. Ονομάζουμε την `while` *αόριστο* βρόχο επειδή απλώς επαναλαμβάνεται μέχρις ότου κάποια συνθήκη γίνει `Ψευδής - False`, ενώ ο βρόχος `for` διατρέχει ένα γνωστό σύνολο στοιχείων, επομένως εκτελεί τόσες επαναλήψεις όσα και τα στοιχεία του συνόλου.

Η σύνταξη ενός βρόχου `for` είναι παρόμοια με του βρόχου `while`, καθώς και σε αυτόν υπάρχει μια δήλωση `for` και ένα σώμα βρόχου:

```
φίλοι = ['Δημήτρης', 'Σοφία', 'Άρης']
for φίλος in φίλοι:
    print('Καλή χρονιά:', φίλος)
print('Τέλος!')
```

Στην ορολογία της Python, η μεταβλητή `φίλοι` είναι μια λίστα[1] τριών συμβολοσειρών και ο βρόχος `for` περνάει από τη λίστα και εκτελεί το σώμα μία φορά για καθεμιά από τις τρεις συμβολοσειρές στη λίστα, καταλήγοντας αυτήν την έξοδο:

```
Καλή χρονιά: Δημήτρης
Καλή χρονιά: Σοφία
Καλή χρονιά: Άρης
Τέλος!
```

Η μετάφραση αυτού του βρόχου `for` στα Αγγλικά δεν είναι τόσο άμεση όσο του `while`, αλλά αν σκεφτείτε τους φίλους ως ένα *σύνολο*, έχει ως εξής: "Εκτελέστε τις εντολές στο σώμα του βρόχου for μία φορά για κάθε φίλο *που ανήκει (in)* στο σετ με όνομα φίλοι."

Κοιτάζοντας τον βρόχο `for`, το *for* και το *in* είναι δεσμευμένες λέξεις της Python και τα `φίλος` και `φίλοι` είναι μεταβλητές.

```
for φίλος in φίλοι:
    print('Καλή χρονιά:', φίλος)
```

Συγκεκριμένα, το `φίλος` είναι η *μεταβλητή επανάληψης* για τον βρόχο for. Η μεταβλητή `φίλος` αλλάζει σε κάθε επανάληψη του βρόχου και ελέγχει την ολοκλήρωση ο βρόχος `for`. Η *μεταβλητή επανάληψης* παίρνει διαδοχικά τις τιμές των τριών συμβολοσειρών που είναι αποθηκευμένες στη μεταβλητή `φίλοι`.

Μοτίβα βρόχων

Συχνά χρησιμοποιούμε έναν βρόχο `for` ή `while` για να διατρέξουμε μια λίστα αντικειμένων ή τα περιεχόμενα ενός αρχείου και αναζητούμε κάτι όπως τη μεγαλύτερη ή τη μικρότερη τιμή των δεδομένων που σαρώνουμε.

Αυτοί οι βρόχοι κατασκευάζονται γενικά ως εξής:

- Αρχικοποίηση μιας ή περισσότερων μεταβλητών πριν από την έναρξη του βρόχου
- Εκτέλεση ορισμένων υπολογισμών σε κάθε στοιχείο στο σώμα του βρόχου, πιθανώς αλλάζοντας τις μεταβλητές στο σώμα του βρόχου
- Εξέταση των μεταβλητών που προκύπτουν όταν ολοκληρωθεί ο βρόχος

[1] Θα εξετάσουμε λεπτομερέστερα τις λίστες σε επόμενο κεφάλαιο.

Θα χρησιμοποιήσουμε μια λίστα αριθμών για να δείξουμε τις έννοιες και τον τρόπο κατασκευή αυτών των μοτίβων βρόχων.

Βρόχοι μέτρησης και άθροισης

Για παράδειγμα, για να μετρήσουμε τον αριθμό των στοιχείων σε μια λίστα, θα γράφαμε τον ακόλουθο βρόχο `for`:

```
count = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    count = count + 1
print('Count: ', count)
```

Αρχικοποιούμε τη μεταβλητή `count` με το μηδέν πριν ξεκινήσει ο βρόχος, και, στη συνέχεια, γράφουμε έναν βρόχο `for` για να διατρέξει τη λίστα των αριθμών. Η μεταβλητή *επανάληψης* ονομάζεται `itervar` και ενώ δεν χρησιμοποιούμε το `itervar` μέσα στον βρόχο, ελέγχει τον βρόχο και προκαλεί την εκτέλεση του σώματος του βρόχου μία φορά για καθεμία από τις τιμές στη λίστα.

Στο σώμα του βρόχου, προσθέτουμε 1 στην τρέχουσα τιμή του `count` για καθεμία από τις τιμές στη λίστα. Ενώ εκτελείται ο βρόχος, η τιμή του `count` είναι το πλήθος των τιμών που έχουμε δει «μέχρι στιγμής».

Μόλις ολοκληρωθεί ο βρόχος, η τιμή του `count` είναι ο συνολικός αριθμός των στοιχείων. Ο συνολικός αριθμός «πέφτει στην αγκαλιά μας» στο τέλος του βρόχου. Κατασκευάζουμε τον βρόχο έτσι ώστε να έχουμε αυτό που θέλουμε όταν τελειώσει ο βρόχος.

Άλλος ένας, παρόμοιος βρόχος, που υπολογίζει το άθροισμα ενός συνόλου αριθμών είναι ο εξής:

```
total = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    total = total + itervar
print('Total: ', total)
```

Σε αυτόν τον βρόχο *χρησιμοποιούμε* τη *μεταβλητή επανάληψης*. Αντί απλώς να προσθέσουμε ένα στο `count` όπως στον προηγούμενο βρόχο, προσθέτουμε τον πραγματικό αριθμό (3, 41, 12, κ.λπ.) στο τρέχον σύνολο κατά τη διάρκεια κάθε επανάληψης του βρόχου. Αν σκεφτείτε τη μεταβλητή `total`, περιέχει το «τρέχον σύνολο

των μέχρι στιγμής τιμών». Έτσι, πριν ξεκινήσει ο βρόχος, το `total` είναι μηδέν, επειδή δεν έχουμε δει ακόμη τιμές, κατά τη διάρκεια του βρόχου το `total` είναι το τρέχον σύνολο και στο τέλος του βρόχου το `total` είναι το γενικό σύνολο όλων των τιμών στο λίστα.

Καθώς εκτελείται ο βρόχος, στο `total` συσσωρεύεται το άθροισμα των στοιχείων. Μια μεταβλητή που χρησιμοποιείται με αυτόν τον τρόπο μερικές φορές ονομάζεται *αθροιστής - accumulator*.

Ούτε ο βρόχος μέτρησης ούτε ο βρόχος άθροισης είναι ιδιαίτερα χρήσιμοι στην πράξη επειδή υπάρχουν οι ενσωματωμένες συναρτήσεις `len()` και `sum()` που υπολογίζουν τον αριθμό των στοιχείων σε μια λίστα και το σύνολο των στοιχείων στη λίστα αντίστοιχα.

Βρόχοι μέγιστου και ελάχιστου

Για να βρούμε τη μεγαλύτερη τιμή σε μια λίστα ή ακολουθία, κατασκευάζουμε τον ακόλουθο βρόχο:

```
μέγιστο = None
print('Πριν:', μέγιστο)
for αριθμός in [3, 41, 12, 9, 74, 15]:
    if μέγιστο is None or αριθμός > μέγιστο :
        μέγιστο = αριθμός
    print('Εκτέλεση:', αριθμός, μέγιστο)
print('Μέγιστο:', μέγιστο)
```

Όταν το πρόγραμμα εκτελείται, η έξοδος είναι η εξής:

```
Πριν: None
Εκτέλεση: 3 3
Εκτέλεση: 41 41
Εκτέλεση: 12 41
Εκτέλεση: 9 41
Εκτέλεση: 74 74
Εκτέλεση: 15 74
Μέγιστο: 74
```

Τη μεταβλητή μέγιστο είναι καλύτερα να τη σκέφτεστε ως τη «μεγαλύτερη τιμή που έχουμε δει μέχρι τώρα». Πριν από τον βρόχο, αρχικοποιήσαμε το μέγιστο με τη σταθερά

None. Το None είναι μια ειδική σταθερή τιμή την οποία μπορούμε να αποθηκεύσουμε σε μια μεταβλητή για να επισημάνουμε τη μεταβλητή ως «κενή».

Πριν ξεκινήσει ο βρόχος, η μεγαλύτερη τιμή που έχουμε δει μέχρι στιγμής είναι None, καθώς δεν έχουμε δει ακόμη τιμές. Ενώ εκτελείται ο βρόχος, εάν το μέγιστο είναι None, τότε παίρνουμε την πρώτη τιμή που βλέπουμε ως τη μεγαλύτερη μέχρι στιγμής. Μπορείτε να δείτε στην πρώτη επανάληψη πότε η τιμή του αριθμός είναι 3, αφού το μέγιστο είναι None, ορίσαμε αμέσως το μέγιστο σε 3.

Μετά την πρώτη επανάληψη, το μέγιστο δεν είναι πλέον None, επομένως ενεργοποιείται το δεύτερο μέρος της σύνθετης λογικής έκφρασης που ελέγχει το αριθμός > μέγιστο, μόνο όταν βλέπουμε μια τιμή μεγαλύτερη από τη «μεγαλύτερη μέχρι τώρα». Όταν βλέπουμε μια νέα τιμή "ακόμη μεγαλύτερη", παίρνουμε αυτή τη νέα τιμή για μέγιστο. Μπορείτε να δείτε στην έξοδο του προγράμματος ότι το μέγιστο προχωρά από 3 σε 41 σε 74.

Στο τέλος του βρόχου, έχουμε σαρώσει όλες τις τιμές και η μεταβλητή μέγιστο περιέχει τώρα τη μεγαλύτερη τιμή στη λίστα.

Για τον υπολογισμό του μικρότερου αριθμού, ο κώδικας είναι παρόμοιος με μια μικρή αλλαγή:

```
ελάχιστο = None
print('Πριν:', ελάχιστο)
for αριθμός in [3, 41, 12, 9, 74, 15]:
    if ελάχιστο is None or αριθμός < smallest:
        ελάχιστο = αριθμός
    print('Εκτέλεση:', αριθμός, ελάχιστο)
print('Ελάχιστο:', ελάχιστο)
```

Και πάλι, το ελάχιστο είναι το "μικρότερο μέχρι στιγμής" πριν, κατά τη διάρκεια και μετά την εκτέλεση του βρόχου. Όταν ολοκληρωθεί ο βρόχος, το ελάχιστο περιέχει την ελάχιστη τιμή στη λίστα.

Και πάλι, όπως και στην καταμέτρηση και την άθροιση, οι ενσωματωμένες συναρτήσεις max() και min() καθιστούν περιττή τη γραφή των παραπάνω βρόχων.

Το ακόλουθο είναι μια απλή έκδοση της ενσωματωμένης συνάρτησης min() της Python:

```
def min(τιμές):
    ελάχιστο = None
```

```
    for αριθμός in τιμές:
        if ελάχιστο is None or αριθμός < ελάχιστο:
            ελάχιστο = αριθμός
    return ελάχιστο
```

Στην εκδοχή της συνάρτησης του μικρότερου κώδικα, αφαιρέσαμε όλες τις εντολές `print` και χρησιμοποιήσαμε τα αντίστοιχα αγγλικά ονόματα μεταβλητών, ώστε να είναι ισοδύναμη με τη συνάρτηση `min` που είναι ήδη ενσωματωμένη στην Python.

Εκσφαλμάτωση

Καθώς αρχίζετε να γράφετε μεγαλύτερα προγράμματα, μπορεί να χρειαστεί να ξοδέψετε περισσότερος χρόνος εκσφαλμάτωσης. Περισσότερος κώδικας σημαίνει περισσότερες πιθανότητες να κάνετε λάθος και περισσότερα μέρη που κρύβουν σφάλματα.

Ένας τρόπος για να μειώσετε το χρόνο εντοπισμού και διόρθωσης σφαλμάτων είναι η "εκσφαλμάτωση με διχοτόμηση". Για παράδειγμα, εάν υπάρχουν 100 γραμμές στο πρόγραμμά σας και τις ελέγξετε μία κάθε φορά, θα χρειαστείτε 100 βήματα.

Αντίθετα, προσπαθήστε να σπάσετε το πρόβλημα στο μισό. Κοιτάξτε στη μέση του προγράμματος ή κοντά σε αυτήν, για μια ενδιάμεση τιμή που μπορείτε να ελέγξετε. Προσθέστε μια εντολή `print` (ή κάτι άλλο που έχει επαληθεύσιμο αποτέλεσμα) και εκτελέστε το πρόγραμμα.

Εάν ο έλεγχος στο μέσο σημείο είναι λανθασμένος, το πρόβλημα πρέπει να βρίσκεται στο πρώτο μισό του προγράμματος. Αν είναι σωστός, το πρόβλημα βρίσκεται στο δεύτερο μισό.

Κάθε φορά που εκτελείτε έναν έλεγχο όπως αυτόν, μειώνετε στο μισό τον αριθμό των γραμμών που πρέπει να ελέγξετε. Μετά από έξι βήματα (τα οποία είναι πολύ λιγότερα από 100), θα καταλήξετε σε μία ή δύο γραμμές κώδικα, τουλάχιστον θεωρητικά.

Στην πράξη δεν είναι πάντα σαφές ποιο είναι το "μέσο του προγράμματος" και δεν είναι πάντα δυνατό να το ελέγξουμε. Δεν έχει νόημα να μετράμε γραμμές και να βρίσκουμε το ακριβές μέσο. Αντίθετα, σκεφτείτε μέρη στο πρόγραμμα όπου μπορεί να υπάρχουν σφάλματα και μέρη στα οποία είναι εύκολο να βάλετε έναν έλεγχο. Στη συνέχεια, επιλέξτε ένα σημείο όπου πιστεύετε ότι οι πιθανότητες είναι περίπου οι ίσες για τις περιπτώσεις το σφάλμα να είναι πριν ή μετά τον έλεγχο.

Γλωσσάριο

αθροιστής : Μια μεταβλητή που χρησιμοποιείται σε έναν βρόχο για να υπολογίσει ένα άθροισμα

αρχικοποίηση : Μια εκχώρηση που δίνει μια αρχική τιμή σε μια μεταβλητή που θα ενημερωθεί.

ατέρμων βρόχος : Ένας βρόχος στον οποίο η συνθήκη τερματισμού δεν ικανοποιείται ποτέ ή για τον οποίο δεν υπάρχει τερματική συνθήκη.

αύξηση : Μια ενημέρωση που αυξάνει την τιμή μιας μεταβλητής (συχνά κατά ένα).

επανάληψη : Επαναλαμβανόμενη εκτέλεση ενός συνόλου εντολών χρησιμοποιώντας είτε μια συνάρτηση που καλεί τον εαυτό της είτε έναν βρόχο.

μείωση : Μια ενημέρωση που μειώνει την τιμή μιας μεταβλητής.

μετρητής : Μια μεταβλητή που χρησιμοποιείται σε έναν βρόχο για να μετρήσει πόσες φορές συνέβη κάτι. Αρχικοποιούμε έναν μετρητή με το μηδέν και μετά αυξάνουμε τον μετρητή κάθε φορά που θέλουμε να "μετρήσουμε" κάτι.

Ασκήσεις

Άσκηση 1: Γράψτε ένα πρόγραμμα, το οποίο διαβάζει επαναληπτικά αριθμούς μέχρι ο χρήστης να εισάγει τέλος. Όταν εισαχθεί τέλος, εκτυπώνει το άθροισμα, το πλήθος και τον μέσο όρο των αριθμών. Εάν ο χρήστης εισάγει οτιδήποτε άλλο εκτός από αριθμούς, εντοπίζει το λάθος με χρήση των `try` και `except`, εκτυπώνει ένα μήνυμα λάθους και ζητά τον επόμενο αριθμό.

```
Εισάγετε έναν αριθμό: 4
Εισάγετε έναν αριθμό: 5
Εισάγετε έναν αριθμό: bad data
Μη έγκυρη είσοδος
Εισάγετε έναν αριθμό: 7
Εισάγετε έναν αριθμό: τέλος
16 3 5.333333333333333
```

Άσκηση 2: Γράψτε ένα άλλο πρόγραμμα, το οποίο ζητά μια λίστα αριθμών, όπως και παραπάνω και στο τέλος εκτυπώνει το μέγιστο και το ελάχιστο όλων των αριθμών αντί για τον μέσο όρο.

Κεφάλαιο 6

Συμβολοσειρές

Μία συμβολοσειρά είναι μία ακολουθία

Μια συμβολοσειρά είναι μια *ακολουθία* χαρακτήρων. Μπορείτε να προσπελάσετε τους χαρακτήρες, έναν κάθε φορά, με τον τελεστή αγκύλης:

```
>>> φρούτο = 'banana'
>>> γράμμα = φρούτο[1]
```

Η δεύτερη πρόταση εξάγει τον χαρακτήρα στη θέση 1 από τη μεταβλητή φρούτο και τον εκχωρεί στη μεταβλητή γράμμα.

Η έκφραση σε αγκύλες ονομάζεται *δείκτης (index)*. Ο δείκτης υποδεικνύει ποιον χαρακτήρα της ακολουθίας θέλετε (εξ ου και το όνομα).

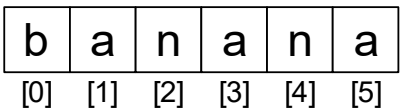
Αλλά μπορεί να μην πάρετε αυτό που περιμένετε:

```
>>> print(γράμμα)
a
```

Για τους περισσότερους ανθρώπους, το πρώτο γράμμα του "banana" είναι «b», όχι «a». Αλλά στην Python, ο δείκτης λειτουργεί κάπως διαφορετικά. Για την αρχή της συμβολοσειράς, για το πρώτο γράμμα είναι μηδέν.

```
>>> γράμμα = φρούτο[0]
>>> print(γράμμα)
b
```

Έτσι το "b" είναι το 0ό γράμμα ("μηδενικό") του "banana", το "a" είναι το 1ο γράμμα ("πρώτο"), και το "n" είναι το 2ο ("δεύτερο") γράμμα.



Εικόνα 6.1: Δείκτες Συμβολοσειρών

Μπορείτε να χρησιμοποιήσετε οποιαδήποτε έκφραση, συμπεριλαμβανομένων

μεταβλητών και τελεστών, σαν ένα δείκτη, αλλά η τιμή του δείκτη πρέπει να είναι ακέραιος αριθμός. Αλλιώς θα πάρετε:

```
>>> γράμμα = φρούτο[1.5]
TypeError: string indices must be integers
```

Λήψη του μήκους μιας συμβολοσειράς χρησιμοποιώντας το `len`.

Η `len` είναι μια ενσωματωμένη συνάρτηση που επιστρέφει τον αριθμό των χαρακτήρων σε μια συμβολοσειρά:

```
>>> φρούτο = 'banana'
>>> len(φρούτο)
6
```

Για να εξάγετε το τελευταίο γράμμα μιας συμβολοσειράς, μπορεί να μπείτε στον πειρασμό να δοκιμάσετε κάτι σαν αυτό:

```
>>> μήκος = len(φρούτο)
>>> τελευταίο = φρούτο[μήκος]
IndexError: string index out of range
```

Ο λόγος για το `IndexError` είναι ότι δεν υπάρχει γράμμα στο "banana" με δείκτη 6. Εφόσον αρχίσαμε να μετράμε από το μηδέν, τα έξι γράμματα αριθμούνται από το 0 έως το 5. Για να πάρετε τον τελευταίο χαρακτήρα, πρέπει να αφαιρέσετε 1 από το μήκος:

```
>>> τελευταίο = φρούτο[μήκος-1]
>>> print(τελευταίο)
a
```

Εναλλακτικά, μπορείτε να χρησιμοποιήσετε αρνητικούς δείκτες, οι οποίοι μετρούν αντίστροφα από το τέλος της συμβολοσειράς. Η έκφραση `φρούτο[-1]` δίνει το τελευταίο γράμμα, το `φρούτο[-2]` δίνει το δεύτερο από το τέλος και ούτω καθεξής.

Διάσχιση συμβολοσειράς με βρόχο

Πολλοί υπολογισμοί εμπλέκουν την επεξεργασία μιας συμβολοσειράς, έναν χαρακτήρα τη φορά. Συχνά ξεκινούν από την αρχή, επιλέγουν κάθε χαρακτήρα με τη σειρά, κάνουν κάτι σε αυτόν και συνεχίζουν μέχρι το τέλος. Αυτό το μοτίβο επεξεργασίας ονομάζεται *διάσχιση (traversal)*. Ένας τρόπος για να γράψετε μια διάσχιση είναι με έναν βρόχο `while`:

```
δείκτης = 0
while δείκτης < len(φρούτο):
    γράμμα = φρούτο[δείκτης]
    print(γράμμα)
    δείκτης = δείκτης + 1
```

Αυτός ο βρόχος διασχίζει τη συμβολοσειρά και εμφανίζει κάθε γράμμα σε μια γραμμή, από μόνο του. Η συνθήκη βρόχου είναι `δείκτης < len(φρούτο)`, οπότε όταν ο `δείκτης` είναι ίσος με το μήκος της συμβολοσειράς, η συνθήκη είναι ψευδής και το σώμα του βρόχου δεν εκτελείται. Ο τελευταίος χαρακτήρας που προσπελαύνεται είναι αυτός με τον δείκτη `len(fruit)-1`, που είναι και ο τελευταίος χαρακτήρας της συμβολοσειράς.

Άσκηση 1: Γράψτε έναν βρόχο `while` που ξεκινά από τον τελευταίο χαρακτήρα της συμβολοσειράς και πηγαίνει προς τα πίσω μέχρι τον πρώτο χαρακτήρα της συμβολοσειράς, τυπώνοντας κάθε γράμμα σε ξεχωριστή γραμμή, απλά με ανάποδη σειρά.

Ένας άλλος τρόπος για να γράψετε μια διάσχιση είναι με έναν βρόχο `for`:

```
for char in fruit:
    print(char)
```

Κάθε φορά που επαναλαμβάνεται ο βρόχος, ο επόμενος χαρακτήρας της συμβολοσειράς εκχωρείται στη μεταβλητή `char`. Ο βρόχος συνεχίζεται μέχρι να προσπελάσουμε όλους τους χαρακτήρες.

Διαμέριση συμβολοσειρών

Ένα *τμήμα* μιας συμβολοσειράς ονομάζεται *slice*. Η επιλογή ενός τμήματος είναι παρόμοια με την επιλογή ενός χαρακτήρα:

```
>>> s = 'Monty Python'
>>> print(s[0:5])
Monty
>>> print(s[6:12])
Python
```

Ο τελεστής `[n:m]` επιστρέφει το τμήμα της συμβολοσειράς από τον "n-οστό" χαρακτήρα μέχρι τον "m-οστό" χαρακτήρα, συμπεριλαμβανομένου του πρώτου αλλά εξαιρώντας τον τελευταίο.

Εάν παραλείψετε τον πρώτο δείκτη (πριν από την άνω και κάτω τελεία), το τμήμα που λαμβάνετε ξεκινά από την αρχή της συμβολοσειράς. Εάν παραλείψετε τον δεύτερο δείκτη, το τμήμα φτάνει μέχρι και το τέλος της συμβολοσειράς:

```
>>> φρούτο = 'banana'
>>> φρούτο[:3]
'ban'
>>> φρούτο[3:]
'ana'
```

Εάν ο πρώτος δείκτης είναι μεγαλύτερος ή ίσος με τον δεύτερο, το αποτέλεσμα είναι μια *κενή συμβολοσειρά*, που αντιπροσωπεύεται από δύο εισαγωγικά:

```
>>> φρούτο = 'banana'
>>> φρούτο[3:3]
''
```

Μια κενή συμβολοσειρά δεν περιέχει χαρακτήρες και έχει μήκος 0, αλλά εκτός από αυτό, είναι ίδια με οποιαδήποτε άλλη συμβολοσειρά.

Άσκηση 2: Δεδομένου ότι το `φρούτο` είναι μια συμβολοσειρά, τι σημαίνει `φρούτο[:]`;

Οι συμβολοσειρές είναι αμετάβλητες

Είναι δελεαστικό να χρησιμοποιήσετε τον τελεστή στην αριστερή πλευρά μιας ανάθεσης, με την πρόθεση να αλλάξετε έναν χαρακτήρα σε μια συμβολοσειρά. Για παράδειγμα:

```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: 'str' object does not support item assignment
```

Το "αντικείμενο - object" σε αυτήν την περίπτωση είναι η συμβολοσειρά και το "στοιχείο - item" είναι ο χαρακτήρας που προσπαθήσατε να τροποποιήσετε. Προς το παρόν, ένα *αντικείμενο* είναι το ίδιο πράγμα με μια τιμή, αλλά θα βελτιώσουμε αυτόν τον ορισμό αργότερα. Ένα *στοιχείο* είναι μία από τις τιμές σε μια ακολουθία.

Το σφάλμα αυτό προκλήθηκε γιατί οι συμβολοσειρές είναι *αμετάβλητες*, πράγμα που σημαίνει ότι δεν μπορείτε να τροποποιήσετε μια υπάρχουσα συμβολοσειρά. Το καλύτερο που μπορείτε να κάνετε είναι να δημιουργήσετε μια νέα συμβολοσειρά που θα

είναι μια παραλλαγή της αρχικής:

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> print(new_greeting)
Jello, world!
```

Αυτό το παράδειγμα συνενώνει ένα νέο πρώτο γράμμα με ένα τμήμα του `greeting`. Δεν έχει καμία επίδραση στην αρχική συμβολοσειρά.

Βρόχος και μέτρηση

Το παρακάτω πρόγραμμα μετράει πόσες φορές εμφανίζεται το γράμμα "a" σε μια συμβολοσειρά:

```
λέξη = 'banana'
πλήθος = 0
for γράμμα in λέξη:
    if γράμμα == 'a':
        πλήθος = πλήθος + 1
print(πλήθος)
```

Αυτό το πρόγραμμα παρουσιάζει ένα άλλο μοτίβο υπολογισμού που ονομάζεται *μετρητής*. Η μεταβλητή πλήθος αρχικοποιείται σε 0 και στη συνέχεια αυξάνεται κάθε φορά που βρίσκεται ένα "a". Όταν ο βρόχος τερματίζει, το πλήθος περιέχει το αποτέλεσμα: το πλήθος των α.

Άσκηση 3: Ενθυλακώστε αυτόν τον κώδικα σε μια συνάρτηση με όνομα `count` και γενικεύστε τον έτσι ώστε να δέχεται τη συμβολοσειρά και το γράμμα ως ορίσματα.

Ο τελεστής in

Η λέξη "in" είναι ένας λογικός τελεστής που δέχεται δύο συμβολοσειρές και επιστρέφει `True` εάν η πρώτη είναι τμήμα της δεύτερης:

```
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

Σύγκριση συμβολοσειρών

Οι τελεστές σύγκρισης λειτουργούν σε συμβολοσειρές. Για να δείτε αν δύο συμβολοσειρές είναι ίσες:

```
if λέξη == 'banana':
    print('Όλα εντάξει, bananas.')
```

Άλλοι τελεστές σύγκρισης που είναι χρήσιμοι για την τοποθέτηση λέξεων σε αλφαβητική σειρά:

```
if λέξη < 'banana':
    print('Η λέξη σου, ' + λέξη + ', προηγείται της banana.')
elif λέξη > 'banana':
    print('Η λέξη σου, ' + λέξη + ', έπεται της banana.')
else:
    print('Όλα εντάξει, bananas.')
```

Η Python δεν χειρίζεται τα κεφαλαία και τα πεζά γράμματα με τον ίδιο τρόπο που το κάνουν οι άνθρωποι. Όλα τα κεφαλαία γράμματα θεωρούνται μικρότερα από όλα τα πεζά, οπότε:

```
Η λέξη σου, Pineapple, προηγείται της banana.
```

Ένας συνηθισμένος τρόπος αντιμετώπισης αυτού του προβλήματος είναι η μετατροπή των συμβολοσειρών σε τυπική μορφή, όπως όλα πεζά, πριν από την εκτέλεση της σύγκρισης. Έχετέ το υπόψη σας, σε περίπτωση που χρειαστεί να υπερασπιστείτε τον εαυτό σας ενάντια σε έναν άνδρα οπλισμένο με έναν ανανά (Pineapple) (αμερικάνικη στρατιωτική αργκό όπου ο ανανάς σημαίνει χειροβομβίδα).

Μέθοδοι συμβολοσειρών

Οι συμβολοσειρές (str) είναι ένα παράδειγμα *αντικειμένων* Python. Ένα αντικείμενο περιέχει τόσο δεδομένα (την ίδια τη συμβολοσειρά) όσο και *μεθόδους*, οι οποίες είναι ουσιαστικά συναρτήσεις που είναι ενσωματωμένες στο αντικείμενο και είναι διαθέσιμες σε κάθε *στιγμιότυπο - instance* του αντικειμένου.

Η Python έχει μια συνάρτηση που ονομάζεται `dir`, η οποία παραθέτει τις διαθέσιμες μεθόδους για κάποιο αντικείμενο. Η συνάρτηση `type` δείχνει τον τύπο ενός αντικειμένου και η συνάρτηση `dir` δείχνει τις διαθέσιμες μεθόδους.

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
>>> help(str.capitalize)
Help on method_descriptor:

capitalize(...)
    S.capitalize() -> str

    Return a capitalized version of S, i.e. make the first character
    have upper case and the rest lower case.
>>>
```

Ενώ η συνάρτηση `dir` παραθέτει τις μεθόδους και μπορείτε να χρησιμοποιήσετε τη `help` για να λάβετε κάποια απλή τεκμηρίωση σε μια μέθοδο, μια καλύτερη πηγή τεκμηρίωσης για τις μεθόδους συμβολοσειράς θα ήταν https://docs.python.org/library/stdtypes.html#string-methods.

Η κλήση μιας *μεθόδου* είναι παρόμοια με την κλήση μιας συνάρτησης (δέχεται ορίσματα και επιστρέφει μια τιμή), αλλά η σύνταξη είναι διαφορετική. Καλούμε μια μέθοδο προσθέτοντας το όνομα της μεθόδου στο τέλος του ονόματος της μεταβλητής, χρησιμοποιώντας την τελεία ως οριοθέτη.

Για παράδειγμα, η μέθοδος `upper` δέχεται μια συμβολοσειρά και επιστρέφει μια νέα συμβολοσειρά με όλα τα γράμματα κεφαλαία:

Αντί για τη σύνταξη των συναρτήσεων `upper(word)`, χρησιμοποιεί τη σύνταξη των μεθόδων `λέξη.upper()`.

```
>>> λέξη = 'banana'
>>> νέα_λέξη = λέξη.upper()
>>> print(νέα_λέξη)
BANANA
```

Αυτή η μορφή διαχωρισμού με τελεία καθορίζει το όνομα της μεθόδου, upper και το όνομα της συμβολοσειράς λέξη, για την εφαρμογή της μεθόδου. Οι κενές παρενθέσεις υποδεικνύουν ότι αυτή η μέθοδος δεν δέχεται όρισμα.

Μια κλήση μεθόδου ονομάζεται *επίκληση - invocation*. Σε αυτή την περίπτωση, θα λέγαμε ότι επικαλούμαστε upper στην λέξη.

Για παράδειγμα, υπάρχει μια μέθοδος συμβολοσειράς με το όνομα find που αναζητά τη θέση μιας συμβολοσειράς μέσα σε μια άλλη:

```
>>> λέξη = 'banana'
>>> δείκτης = λέξη.find('a')
>>> print(δείκτης)
1
```

Σε αυτό το παράδειγμα, επικαλούμαστε την find στη λέξη και δίνουμε το γράμμα που αναζητούμε ως παράμετρο.

Η μέθοδος find μπορεί να βρει υποσυμβολοσειρές καθώς και χαρακτήρες:

```
>>> λέξη.find('na')
2
```

Μπορεί να πάρει ως δεύτερο όρισμα τον δείκτη από όπου πρέπει να ξεκινήσει:

```
>>> λέξη.find('na', 3)
4
```

Μια συνηθισμένη εργασία είναι να αφαιρέσετε τους λευκούς χαρακτήρες (κενά, tab ή νέες γραμμές) από την αρχή και το τέλος μιας συμβολοσειράς χρησιμοποιώντας τη μέθοδο strip:

```
>>> γραμμή = '  Πάμε λοιπόν  '
>>> γραμμή.strip()
'Πάμε λοιπόν'
```

Ορισμένες μέθοδοι όπως το startswith επιστρέφουν λογικές τιμές.

```
>>> γραμμή = 'Καλή σας μέρα'
>>> γραμμή.startswith('Καλή')
True
```

```
>>> γραμμή.startswith('κ')
False
```

Θα προσέξατε ότι το `startswith` κάνει διάκριση πεζών-κεφαλαίων, οπότε μερικές φορές παίρνουμε μια γραμμή και μετατρέπουμε όλα τα γράμματα σε πεζά προτού κάνουμε οποιονδήποτε έλεγχο χρησιμοποιώντας τη μέθοδο `lower`.

```
>>> γραμμή = 'Καλή σας μέρα'
>>> γραμμή.startswith('κ')
False
>>> γραμμή.lower()
'καλή σας μέρα'
>>> γραμμή.lower().startswith('κ')
True
```

Στο τελευταίο παράδειγμα, καλείται η μέθοδος `lower` και στη συνέχεια χρησιμοποιούμε τη `startswith` για να δούμε αν η πεζή συμβολοσειρά που προέκυψε ξεκινά με το γράμμα "κ". Εφόσον είμαστε προσεκτικοί με τη σειρά, μπορούμε να κάνουμε πολλές κλήσεις μεθόδων σε μία μόνο έκφραση.

Άσκηση 4: Υπάρχει μια μέθοδος συμβολοσειρών που ονομάζεται `count` που είναι παρόμοια με τη συνάρτηση που δημιουργήσατε στην προηγούμενη άσκηση. Διαβάστε την τεκμηρίωση αυτής της μεθόδου στη διεύθυνση:

https://docs.python.org/library/stdtypes.html#string-methods

Γράψτε μια κλήση της, που μετράει πόσες φορές εμφανίζεται το γράμμα "a" στο "banana".

Ανάλυση συμβολοσειρών

Συχνά, θέλουμε να ψάξουμε σε μια συμβολοσειρά και να βρούμε μια υποσυμβολοσειρά της. Για παράδειγμα, αν μας παρουσιαζόταν μια σειρά γραμμών μορφοποιημένες όλες ως εξής:

`From stephen.marquard@ uct.ac.za Sat Jan  5 09:14:16 2008`

και θέλαμε να εξάγουμε μόνο το δεύτερο μισό της διεύθυνσης (δηλαδή το `uct.ac.za`) από κάθε γραμμή, θα μπορούσαμε να το κάνουμε χρησιμοποιώντας τη μέθοδο `find` και την διαμέριση συμβολοσειράς.

Αρχικά, θα βρούμε τη θέση του συμβόλου `at ( @ )` στη συμβολοσειρά. Στη συνέχεια,

θα βρούμε τη θέση του πρώτου διαστήματος *μετά* το σύμβολο at και, στη συνέχεια, θα χρησιμοποιήσουμε τη διαμέριση συμβολοσειράς για να εξαγάγουμε το τμήμα της συμβολοσειράς που αναζητούμε.

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> θέσηat = data.find('@')
>>> print(θέσηat)
21
>>> θέση_τέλους = data.find(' ',θέσηat)
>>> print(θέση_τέλους)
31
>>> host = data[θέσηat+1:θέση_τέλους]
>>> print(host)
uct.ac.za
>>>
```

Χρησιμοποιούμε μια έκδοση της μεθόδου `find` που μας επιτρέπει να καθορίσουμε μια θέση στη συμβολοσειρά από όπου θέλουμε να αρχίσει να ψάχνει το `find`. Όταν τεμαχίζουμε, εξάγουμε τους χαρακτήρες "έναν πέρα από το σύμβολο at έως, *αλλά χωρίς* αυτόν, τον χαρακτήρα διαστήματος".

Η τεκμηρίωση για τη μέθοδο `find` είναι διαθέσιμη στη διεύθυνση

https://docs.python.org/library/stdtypes.html#string-methods.

Τελεστής μορφής

Ο *τελεστής μορφής*, % μας επιτρέπει να κατασκευάσουμε συμβολοσειρές, αντικαθιστώντας τμήματα των συμβολοσειρών με δεδομένα που είναι αποθηκευμένα σε μεταβλητές. Όταν εφαρμόζεται σε ακέραιους αριθμούς, το % είναι ο τελεστής ακέραιου υπολοίπου. Αλλά όταν ο πρώτος τελεστής είναι μια συμβολοσειρά, το % είναι ο τελεστής μορφοποίησης.

Ο πρώτος τελεστέος είναι η *συμβολοσειρά μορφής (format string)*, η οποία περιέχει μία ή περισσότερες *ακολουθίες μορφής (format sequences)* που καθορίζουν την μορφή του δεύτερου τελεστέου. Το αποτέλεσμα είναι μια συμβολοσειρά.

Για παράδειγμα, η ακολουθία μορφής %d σημαίνει ότι ο δεύτερος τελεστέος πρέπει να είναι ακέραιος (το "d" σημαίνει "decimal"):

```
>>> καμήλες = 42
>>> '%d' % καμήλες
'42'
```

Το αποτέλεσμα είναι η συμβολοσειρά '42', η οποία δεν πρέπει να συγχέεται με την ακέραια τιμή 42.

Μια ακολουθία μορφής μπορεί να εμφανιστεί οπουδήποτε μέσα στη συμβολοσειρά, ώστε να μπορείτε να ενσωματώσετε μια τιμή σε μια πρόταση:

```
>>> καμήλες = 42
>>> 'Έχω εντοπίσει %d καμήλες.' % καμήλες
'Έχω εντοπίσει 42 καμήλες.'
```

Εάν υπάρχουν περισσότερες από μία ακολουθίες μορφής στη συμβολοσειρά, το δεύτερο όρισμα πρέπει να είναι πλειάδα (tuple)[1]. Κάθε ακολουθία μορφής αντιστοιχίζεται με ένα στοιχείο της πλειάδας, με τη σειρά.

Το παρακάτω παράδειγμα χρησιμοποιεί το %d για να αναπαραστήσει έναν ακέραιο, το %g για να αναπαραστήσει έναν αριθμό κινητής υποδιαστολής (μην ρωτήσετε γιατί) και το %s για να αναπαραστήσει μια συμβολοσειρά:

```
>>> 'Σε %d χρόνια έχω εντοπίσει %g %s.' % (3, 0.1, 'καμήλες')
'Σε 3 χρόνια έχω εντοπίσει 0.1 καμήλες.'
```

Ο αριθμός των στοιχείων στην πλειάδα πρέπει να ταιριάζει με τον αριθμό των ακολουθιών μορφής στη συμβολοσειρά. Οι τύποι των στοιχείων πρέπει επίσης να ταιριάζουν με τις ακολουθίες μορφής:

```
>>> '%d %d %d' % (1, 2)
TypeError: not enough arguments for format string
>>> '%d' % 'dollars'
TypeError: %d format: a number is required, not str
```

Στο πρώτο παράδειγμα, δεν υπάρχουν αρκετά στοιχεία, στο δεύτερο, το στοιχείο είναι λάθος τύπου.

Ο τελεστής μορφής είναι ισχυρός, αλλά μπορεί να είναι δύσκολος στη χρήση του.

---

[1] Η πλειάδα είναι μια ακολουθία τιμών διαχωρισμένων με κόμμα μέσα σε ένα ζεύγος παρενθέσεων. Θα καλύψουμε τις πλειάδες στο Κεφάλαιο 10

Μπορείτε να διαβάσετε περισσότερα για αυτόν στο

https://docs.python.org/library/stdtypes.html#printf-style-string-formatting.

Εκσφαλμάτωση

Μια δεξιότητα που πρέπει να καλλιεργήσετε καθώς προγραμματίζετε είναι να ρωτάτε πάντα τον εαυτό σας: "Τι μπορεί να πάει στραβά εδώ;" ή εναλλακτικά, "Τι τρελό πράγμα μπορεί να κάνει ο χρήστης μας για να καταρρεύσει το (φαινομενικά) τέλειο πρόγραμμά μας;"

Για παράδειγμα, δείτε το πρόγραμμα που χρησιμοποιήσαμε για να εξηγήσουμε τον βρόχο `while` στο κεφάλαιο για την επανάληψη:

```
while True:
    γραμμή = input('> ')
    if γραμμή[0] == '#':
        continue
    if γραμμή == 'τέλος':
        break
    print(γραμμή)
print('Τέλος!')
```

#Code: http://www.gr.py4e.com/code3/copytildone2.py

Δείτε τι συμβαίνει όταν ο χρήστης εισάγει μια κενή γραμμή στην είσοδο:

```
> hello there
hello there
> # don't print this
> print this!
print this!
>
Traceback (most recent call last):
  File "copytildone.py", line 3, in <module>
    if γραμμή[0] == '#':
IndexError: string index out of range
```

Ο κώδικας λειτουργεί καλά μέχρι να εμφανιστεί μια κενή γραμμή. Τότε δεν υπάρχει μηδενικός χαρακτήρας, οπότε παίρνουμε ένα traceback. Υπάρχουν δύο λύσεις για να γίνει η γραμμή τρία "ασφαλής" ακόμα κι αν η γραμμή είναι άδεια.

Μια δυνατότητα είναι απλώς να χρησιμοποιήσετε τη μέθοδο `startswith` που επιστρέφει `False` εάν η συμβολοσειρά είναι κενή.

```
if γραμμή.startswith('#'):
```

Ένας άλλος τρόπος είναι να γράψετε, για ασφάλεια, την εντολή `if` χρησιμοποιώντας το μοτίβο *φύλακα (guardian pattern)* και να βεβαιωθείτε ότι η δεύτερη λογική έκφραση αξιολογείται μόνο όπου υπάρχει τουλάχιστον ένας χαρακτήρας στη συμβολοσειρά:

```
if len(γραμμή) > 0 and γραμμή[0] == '#':
```

Γλωσσάριο

flag : Μια λογική μεταβλητή που χρησιμοποιείται για να δείξει εάν μια συνθήκη είναι αληθής ή ψευδής.

ακολουθία - sequence : Ένα διατεταγμένο σύνολο, δηλαδή ένα σύνολο τιμών όπου κάθε τιμή προσδιορίζεται από έναν ακέραιο δείκτη.

ακολουθία μορφής : Μια ακολουθία χαρακτήρων σε μια συμβολοσειρά μορφής, όπως το %d, που καθορίζει τον τρόπο μορφοποίησης μιας τιμής.

αμετάβλητο : Η ιδιότητα μιας ακολουθίας της οποίας τα στοιχεία δεν μπορούν να αλλάξουν.

αναζήτηση : Ένα μοτίβο διέλευσης που σταματά όταν βρει αυτό που ψάχνει.

αντικείμενο - object : Κάτι στο οποίο μπορεί να αναφέρεται μια μεταβλητή. Προς το παρόν, μπορείτε να χρησιμοποιήσετε το "αντικείμενο" και το "τιμή" εναλλακτικά.

δείκτης : Μια ακέραια τιμή που χρησιμοποιείται για την επιλογή ενός στοιχείου από μια ακολουθία, όπως ενός χαρακτήρας από μια συμβολοσειρά.

διάσχιση : Η προσπέλαση των στοιχείων με μια σειρά, εκτελώντας μια παρόμοια λειτουργία στο καθένα.

κενή συμβολοσειρά : Μια συμβολοσειρά χωρίς χαρακτήρες και μήκος 0, που αντιπροσωπεύεται από δύο εισαγωγικά.

κλήση : Μια δήλωση που καλεί κάποια μέθοδο.

μέθοδος : Μια συνάρτηση που σχετίζεται με ένα αντικείμενο και καλείται χρησιμοποιώντας διαχωρισμός με τελεία.

μετρητής : Μια μεταβλητή που χρησιμοποιείται για να μετρήσει κάτι, συνήθως αρχικοποιείται με το μηδέν και στη συνέχεια αυξάνεται.

στοιχείο : Μία από τις τιμές μιας ακολουθίας.

συμβολοσειρά μορφής : Μια συμβολοσειρά, που χρησιμοποιείται με τον τελεστή μορφής, που περιέχει ακολουθίες μορφής.

τελεστής μορφής - format operator: Ένας τελεστής,% , που παίρνει μια συμβολοσειρά μορφής και μια πλειάδα και δημιουργεί μια συμβολοσειρά που περιλαμβάνει τα στοιχεία της πλειάδας μορφοποιημένα όπως καθορίζεται από τη συμβολοσειρά μορφής.

τμήμα - slice : Ένα μέρος μιας συμβολοσειράς που καθορίζεται από μια σειρά δεικτών.

Ασκήσεις

Άσκηση 5: Πάρτε τον ακόλουθο κώδικα Python που αποθηκεύει μια συμβολοσειρά:

```
str = 'X-DSPAM-Confidence: 0.8475'
```

Χρησιμοποιήστε `find` και διαμέριση συμβολοσειράς για να εξαγάγετε το τμήμα της συμβολοσειράς μετά τον χαρακτήρα άνω και κάτω τελείας και στη συνέχεια χρησιμοποιήστε τη συνάρτηση `float` για να μετατρέψετε τη συμβολοσειρά που εξαγάγατε σε αριθμό κινητής υποδιαστολής.

Άσκηση 6: Διαβάστε την τεκμηρίωση των μεθόδων συμβολοσειράς στο https://docs.python.org/library/stdtypes.html#string-methods **Ίσως θελήσετε να πειραματιστείτε με μερικά από αυτά για να βεβαιωθείτε ότι καταλαβαίνετε πώς λειτουργούν. Τα `strip` και `replace` είναι ιδιαίτερα χρήσιμα.**

Η τεκμηρίωση χρησιμοποιεί μια σύνταξη που μπορεί να προκαλεί σύγχυση. Για παράδειγμα, στο `find(sub[, start[, end]])`, οι αγκύλες υποδεικνύουν προαιρετικά ορίσματα. Επομένως, το `sub` απαιτείται, αλλά το `start` είναι προαιρετικό και εάν συμπεριλάβετε το `start`, τότε το `end` είναι προαιρετικό.

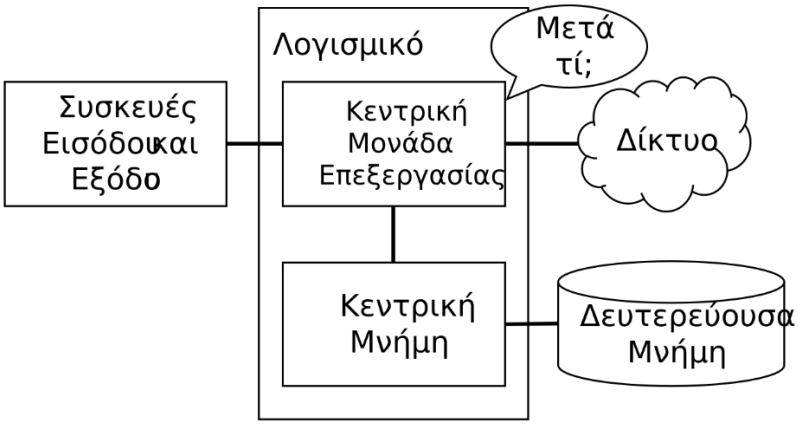
Κεφάλαιο 7

Αρχεία

Μονιμότητα

Μέχρι στιγμής, μάθαμε πώς να γράφουμε προγράμματα και να επικοινωνούμε τις προθέσεις μας στην *Κεντρική Μονάδα Επεξεργασίας* χρησιμοποιώντας δομή επιλογής, συναρτήσεις και δομή επανάληψης. Μάθαμε πώς να δημιουργούμε και να χρησιμοποιούμε δομές δεδομένων στην *Κύρια μνήμη*. Η CPU και η μνήμη είναι εκεί όπου λειτουργεί και εκτελείται το λογισμικό μας. Εκεί συμβαίνει όλη η «σκέψη».

Αλλά αν θυμάστε από τις συζητήσεις μας για την αρχιτεκτονική υλικού, μόλις απενεργοποιηθεί η τροφοδοσία, ό,τι είναι αποθηκευμένο είτε στην CPU είτε στην κύρια μνήμη διαγράφεται. Έτσι, μέχρι τώρα, τα προγράμματά μας ήταν απλώς παροδική διασκέδαση, ασκήσεις για την εκμάθηση Python.



Εικόνα 7.1: Δευτερεύουσα Μνήμη

Σε αυτό το κεφάλαιο, αρχίζουμε να εργαζόμαστε με τη *Δευτερεύουσα μνήμη* (ή με αρχεία). Η δευτερεύουσα μνήμη δεν διαγράφεται όταν η τροφοδοσία ρεύματος σταματήσει. Επίσης, στην περίπτωση μιας μονάδας flash USB, τα δεδομένα που γράφουμε με τα προγράμματά μας μπορούν να αφαιρεθούν από το σύστημά μας και να μεταφερθούν σε άλλο σύστημα.

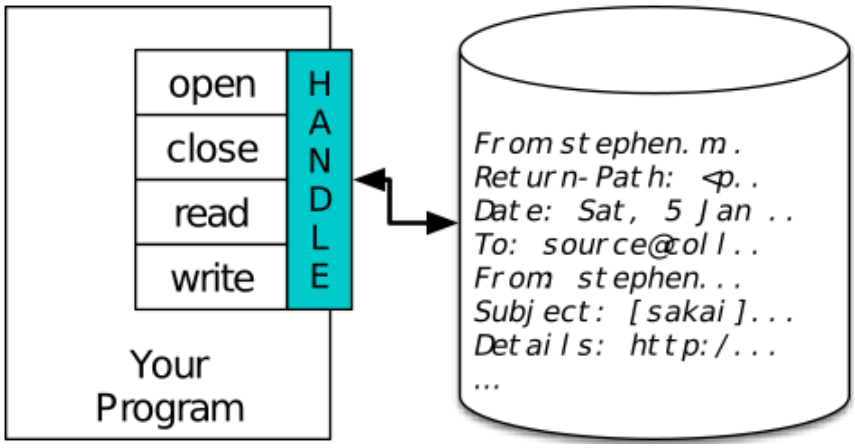
Θα επικεντρωθούμε κυρίως στην ανάγνωση και τη σύνταξη αρχείων κειμένου όπως αυτά που δημιουργούμε σε ένα πρόγραμμα επεξεργασίας κειμένου. Αργότερα θα δούμε

πώς δουλεύουμε με αρχεία βάσης δεδομένων που είναι δυαδικά αρχεία, ειδικά σχεδιασμένα για ανάγνωση και εγγραφή μέσω λογισμικού βάσης δεδομένων.

Άνοιγμα αρχείων

Όταν θέλουμε να διαβάσουμε ή να γράψουμε ένα αρχείο (ας πούμε στον σκληρό δίσκο), πρέπει πρώτα να *ανοίξουμε (open)* το αρχείο. Το άνοιγμα του αρχείου επικοινωνεί με το λειτουργικό σας σύστημα, το οποίο γνωρίζει πού αποθηκεύονται τα δεδομένα του κάθε αρχείου. Όταν ανοίγετε ένα αρχείο, ζητάτε από το λειτουργικό σύστημα να βρει το αρχείο βάση ονόματος και να βεβαιωθεί ότι το αρχείο υπάρχει. Σε αυτό το παράδειγμα, ανοίγουμε το αρχείο *mbox.txt*, το οποίο θα πρέπει να είναι αποθηκευμένο στον ίδιο φάκελο, στον οποίο βρίσκεστε όταν ξεκινάτε την Python. Μπορείτε να κάνετε λήψη αυτού του αρχείου από το www.gr.py4e.com/code3/mbox.txt

```
>>> fhand = open('mbox.txt')
>>> print(fhand)
<_io.TextIOWrapper name='mbox.txt' mode='r' encoding='cp1252'>
```



Εικόνα 7.2: Περιγραφέας Αρχείου

Εάν το open ολοκληρωθεί με επιτυχία, το λειτουργικό σύστημα μας επιστρέφει έναν *περιγραφέα αρχείου (file handle)*. Ο περιγραφέας αρχείου δεν είναι τα πραγματικά δεδομένα που περιέχονται στο αρχείο, αλλά αντίθετα είναι μια "λαβή" που μπορούμε να χρησιμοποιήσουμε για να διαβάσουμε τα δεδομένα. Σας δίνεται ένας περιγραφέας εάν υπάρχει το ζητούμενο αρχείο και έχετε τα κατάλληλα δικαιώματα για να διαβάσετε το αρχείο.

Εάν το αρχείο δεν υπάρχει, το open θα αποτύχει με ένα traceback και δεν θα δημιουργηθεί περιγραφέας για πρόσβαση στα περιεχόμενα του αρχείου:

```
>>> fhand = open('stuff.txt')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'stuff.txt'
```

Αργότερα θα χρησιμοποιήσουμε τα try και except για να αντιμετωπίσουμε, πιο χαριτωμένα, την κατάσταση όπου επιχειρούμε να ανοίξουμε ένα αρχείο που δεν υπάρχει.

Αρχεία κειμένου και γραμμές

Ένα αρχείο κειμένου μπορούν να θεωρηθεί ως μια ακολουθία γραμμών, όπως μια συμβολοσειρά Python μπορεί να θεωρηθεί ως μια ακολουθία χαρακτήρων. Για παράδειγμα, αυτό είναι ένα δείγμα αρχείου κειμένου που καταγράφει τη δραστηριότητα αλληλογραφίας από διάφορα άτομα σε μια ομάδα ανάπτυξης έργου ανοιχτού κώδικα:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
...
```

Ολόκληρο το αρχείο αλληλεπιδράσεων αλληλογραφίας είναι διαθέσιμο στο

www.gr.py4e.com/code3/mbox.txt

και μια μικρότερη έκδοση του αρχείου είναι διαθέσιμη στο

www.gr.py4e.com/code3/mbox-short.txt

Αυτά τα αρχεία είναι σε τυπική μορφή αρχείου που περιέχει πολλά μηνύματα αλληλογραφίας. Οι γραμμές που ξεκινούν με "From (Από)" διαχωρίζουν τα μηνύματα και οι γραμμές που ξεκινούν με "From:" αποτελούν μέρος των μηνυμάτων. Για περισσότερες πληροφορίες σχετικά με τη μορφή mbox, ανατρέξτε στο https://en.wikipedia.org/wiki/Mbox.

Για να χωρίσετε το αρχείο σε γραμμές, υπάρχει ένας ειδικός χαρακτήρας που αντιπροσωπεύει το "τέλος της γραμμής" που ονομάζεται χαρακτήρας *newline - νέα γραμμή*.

Στην Python, σε σταθερές συμβολοσειρών, αντιπροσωπεύουμε τον χαρακτήρα *newline* ως ανάστροφη κάθετο-n (\n). Παρόλο που αυτό μοιάζει με δύο χαρακτήρες, είναι στην πραγματικότητα ένας μόνο χαρακτήρας. Όταν εξετάζουμε τη μεταβλητή πληκτρολογώντας "stuff" στον διερμηνέα, μας δείχνει το \n στη συμβολοσειρά, αλλά όταν χρησιμοποιούμε την print για να εμφανίσουμε τη συμβολοσειρά, βλέπουμε τη συμβολοσειρά σπασμένη σε δύο γραμμές από τον χαρακτήρα νέας γραμμής.

```
>>> stuff = 'Hello\nWorld!'
>>> stuff
'Hello\nWorld!'
>>> print(stuff)
Hello
World!
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
>>> len(stuff)
3
```

Μπορείτε επίσης να δείτε ότι το μήκος της συμβολοσειράς X\nY είναι *τρεις* χαρακτήρες, επειδή ο χαρακτήρας νέας γραμμής είναι ένας χαρακτήρας.

Έτσι, όταν κοιτάμε τις γραμμές σε ένα αρχείο, πρέπει να *φανταστούμε* ότι υπάρχει ένας ειδικός, αόρατος χαρακτήρας, που ονομάζεται νέα γραμμή, στο τέλος κάθε γραμμής, που σηματοδοτεί το τέλος της γραμμής.

Ο χαρακτήρας νέας γραμμής, λοιπόν, διαχωρίζει τους χαρακτήρες του αρχείου σε γραμμές.

Ανάγνωση αρχείων

Ενώ ο *περιγραφέας αρχείου* δεν περιέχει τα δεδομένα για το αρχείο, είναι πολύ εύκολο να τον χρησιμοποιήσετε και να δημιουργήσετε έναν βρόχο for για να διαβάσετε και να μετρήσετε κάθε μία από τις γραμμές σε ένα αρχείο:

```
fhand = open('mbox-short.txt')

πλήθος = 0

for γραμμή in fhand:

    πλήθος = πλήθος + 1
```

```
print('Πλήθος γραμμών:', πλήθος)
```

#Code: http://www.gr.py4e.com/code3/open.py

Μπορούμε να χρησιμοποιήσουμε τον περιγραφέα αρχείου ως μια ακολουθία στον βρόχο `for`. Ο παραπάνω βρόχος `for`, απλώς μετράει τον αριθμό των γραμμών στο αρχείο και τις εκτυπώνει. Η κατά προσέγγιση μετάφραση του βρόχου `for` στα αγγλικά είναι, "για κάθε γραμμή στο αρχείο που αντιπροσωπεύεται από τον περιγραφέα του αρχείου, προσθέστε ένα στη μεταβλητή `count`".

Ο λόγος που η συνάρτηση `open` δεν διαβάζει ολόκληρο το αρχείο είναι ότι το αρχείο μπορεί να είναι αρκετά μεγάλο, με πολλά gigabyte δεδομένων. Η δήλωση `open` χρειάζεται τον ίδιο χρόνο ανεξάρτητα από το μέγεθος του αρχείου. Ο βρόχος `for` είναι αυτός που προκαλεί, στην πραγματικότητα, την ανάγνωση των δεδομένων από το αρχείο.

Όταν διαβάζεται το αρχείο χρησιμοποιώντας έναν βρόχο `for`, με αυτόν τον τρόπο, η Python φροντίζει να διασπάσει τα δεδομένα του αρχείου σε ξεχωριστές γραμμές χρησιμοποιώντας τον χαρακτήρα νέας γραμμής. Η Python διαβάζει κάθε γραμμή μέσω του χαρακτήρα νέας γραμμής και περιλαμβάνει το χαρακτήρα νέας γραμμής, ως τον τελευταίο χαρακτήρα, στη μεταβλητή γραμμή σε κάθε επανάληψη του βρόχου `for`.

Επειδή ο βρόχος `for` διαβάζει τα δεδομένα μία γραμμή τη φορά, μπορεί να διαβάσει και να μετρήσει αποτελεσματικά τις γραμμές σε πολύ μεγάλα αρχεία χωρίς να εξαντληθεί η κύρια μνήμη για την αποθήκευση των δεδομένων. Το παραπάνω πρόγραμμα μπορεί να μετρήσει τις γραμμές σε αρχείο οποιουδήποτε μεγέθους χρησιμοποιώντας πολύ λίγη μνήμη, αφού κάθε γραμμή διαβάζεται, μετριέται και στη συνέχεια απορρίπτεται.

Εάν γνωρίζετε ότι το αρχείο είναι σχετικά μικρό σε σύγκριση με το μέγεθος της κύριας μνήμης σας, μπορείτε να διαβάσετε ολόκληρο το αρχείο σε μία συμβολοσειρά, χρησιμοποιώντας τη μέθοδο `read` στον περιγραφέα του αρχείου.

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

Σε αυτό το παράδειγμα, ολόκληρο το περιεχόμενο (και οι 94.626 χαρακτήρες) του αρχείου *mbox-short.txt* διαβάζονται απευθείας στη μεταβλητή `inp`. Χρησιμοποιούμε διαμέριση συμβολοσειράς για να εκτυπώσουμε τους πρώτους 20 χαρακτήρες των

δεδομένων συμβολοσειράς που είναι αποθηκευμένα στο inp.

Όταν το αρχείο διαβάζεται με αυτόν τον τρόπο, όλοι οι χαρακτήρες, συμπεριλαμβανομένων όλων των γραμμών και χαρακτήρων νέας γραμμής, είναι μια μεγάλη συμβολοσειρά στη μεταβλητή inp. Είναι καλή ιδέα να αποθηκεύεται η έξοδος του read ως μεταβλητή, επειδή κάθε κλήση της read εξαντλεί τους πόρους:

```
>>> fhand = open('mbox-short.txt')
>>> print(len(fhand.read()))
94626
>>> print(len(fhand.read()))
0
```

Θυμηθείτε ότι αυτή η μορφή της συνάρτησης open θα πρέπει να χρησιμοποιείται μόνο εάν τα δεδομένα του αρχείου χωράνε άνετα στην κύρια μνήμη του υπολογιστή σας. Εάν το αρχείο είναι πολύ μεγάλο για να χωρέσει στην κύρια μνήμη, θα πρέπει να γράψετε το πρόγραμμά σας έτσι ώστε να διαβάσει το αρχείο σε κομμάτια, χρησιμοποιώντας έναν βρόχο for ή while.

Φιλτράρισμα αρχείου

Όταν κάνετε φιλτράρισμα δεδομένων σε ένα αρχείο, ένα πολύ συνηθισμένο μοτίβο είναι να διαβάζετε το αρχείο, αγνοώντας τις περισσότερες γραμμές και να επεξεργάζεστε μόνο τις γραμμές που πληρούν μια συγκεκριμένη συνθήκη. Μπορούμε να συνδυάσουμε την ανάγνωση ενός αρχείου με μεθόδους συμβολοσειράς, για να δημιουργήσουμε απλούς μηχανισμούς φιλτραρίσματος.

Για παράδειγμα, αν θέλαμε να διαβάσουμε ένα αρχείο και να εκτυπώσουμε μόνο τις γραμμές που αρχίζουν με το πρόθεμα "From:", θα μπορούσαμε να χρησιμοποιήσουμε τη μέθοδο συμβολοσειράς *startswith* για να επιλέξουμε μόνο τις γραμμές με το επιθυμητό πρόθεμα:

```
fhand = open('mbox-short.txt')
for γραμμή in fhand:
    if γραμμή.startswith('From:'):
        print(γραμμή)
```

#Code: http://www.gr.py4e.com/code3/search1.py

Όταν εκτελείται αυτό το πρόγραμμα, έχουμε την ακόλουθη έξοδο:

```
From: stephen.marquard@uct.ac.za
```

```
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
...
```

Η έξοδος φαίνεται υπέροχη αφού οι μόνες γραμμές που βλέπουμε είναι αυτές που ξεκινούν με "From:", αλλά γιατί βλέπουμε τις επιπλέον κενές γραμμές; Αυτό οφείλεται στον αόρατο χαρακτήρα *newline*. Κάθε μία από τις γραμμές τελειώνει με έναν χαρακτήρα νέας γραμμής, επομένως η δήλωση `print` εκτυπώνει τη συμβολοσειρά της μεταβλητή *γραμμή*, που όμως περιλαμβάνει έναν χαρακτήρα νέας γραμμής και στη συνέχεια η `print` προσθέτει *άλλη μία* νέα γραμμή, με αποτέλεσμα το εφέ διπλού διαστήματος που βλέπουμε.

Θα μπορούσαμε να χρησιμοποιήσουμε την διαμέριση γραμμής, για να εκτυπώσουμε τους χαρακτήρες εκτός από τον τελευταίο, αλλά μια απλούστερη προσέγγιση είναι να χρησιμοποιήσουμε τη μέθοδο *rstrip* που αφαιρεί τους λευκούς χαρακτήρες από τη δεξιά πλευρά μιας συμβολοσειράς ως εξής:

```
fhand = open('mbox-short.txt')
for γραμμή in fhand:
    γραμμή = γραμμή.rstrip()
    if γραμμή.startswith('From:'):
        print(γραμμή)
```

#Code: http://www.gr.py4e.com/code3/search2.py

Όταν εκτελείται αυτό το πρόγραμμα, έχουμε την ακόλουθη έξοδο:

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
...
```

Καθώς τα προγράμματα επεξεργασίας αρχείων σας γίνονται πιο περίπλοκα, μπορεί να θελήσετε να δομήσετε τους βρόχους αναζήτησης χρησιμοποιώντας το `continue`. Η βασική ιδέα του βρόχου αναζήτησης είναι ότι ψάχνετε για "ενδιαφέρουσες" γραμμές και ουσιαστικά παρακάμπτετε τις "αδιάφορες" γραμμές. Και μετά, όταν βρίσκουμε μια ενδιαφέρουσα γραμμή, κάνουμε κάτι με αυτή τη γραμμή.

Μπορούμε να δομήσουμε τον βρόχο για να ακολουθήσουμε το μοτίβο της παράκαμψης αδιάφορων γραμμών ως εξής:

```
fhand = open('mbox-short.txt')
for γραμμή in fhand:
    γραμμή = γραμμή.rstrip()
    # Παράκαμψη `αδιάφορων` γραμμών
    if not γραμμή.startswith('From:'):
        continue
    # Επεξεργασία 'ενδιαφερόντων' γραμμών
    print(γραμμή)
```

#Code: http://www.gr.py4e.com/code3/search3.py

Η έξοδος του προγράμματος είναι η ίδια. Στα αγγλικά, οι αδιάφορες γραμμές είναι εκείνες που δεν ξεκινούν με "From:", τις οποίες παραλείπουμε χρησιμοποιώντας το `continue`. Για τις "ενδιαφέρουσες" γραμμές (δηλαδή αυτές που ξεκινούν με "From:") εκτελούμε την επεξεργασία σε αυτές τις γραμμές.

Μπορούμε να χρησιμοποιήσουμε τη μέθοδο συμβολοσειράς `find` για να προσομοιώσουμε την αναζήτηση των προγραμμάτων επεξεργασίας κειμένου, που βρίσκει γραμμές στις οποίες περιλαμβάνεται, οπουδήποτε, η συμβολοσειρά αναζήτησης. Μιας και το `find` αναζητά την εμφάνιση μιας συμβολοσειράς μέσα σε μια άλλη συμβολοσειρά και είτε επιστρέφει τη θέση της συμβολοσειράς είτε -1, εάν η συμβολοσειρά δεν βρέθηκε, μπορούμε να γράψουμε τον ακόλουθο βρόχο για να εμφανίσουμε τις γραμμές που περιέχουν τη συμβολοσειρά "@uct.ac .za" (δηλαδή, προέρχονται από το Πανεπιστήμιο του Κέιπ Τάουν στη Νότια Αφρική):

```
fhand = open('mbox-short.txt')
for γραμμή in fhand:
    γραμμή = γραμμή.rstrip()
    if γραμμή.find('@uct.ac.za') == -1: continue
    print(γραμμή)
```

#Code: http://www.gr.py4e.com/code3/search4.py

Ο οποίος παράγει την ακόλουθη έξοδο:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using -
f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
```

```
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f
From: david.horwitz@uct.ac.za
Author: david.horwitz@uct.ac.za
...
```

Εδώ χρησιμοποιούμε επίσης τη σύντομη μορφή της εντολής `if`, όπου βάζουμε το `continue` στην ίδια γραμμή με το `if`. Αυτή η σύντομη μορφή του `if` λειτουργεί το ίδιο όπως αν το `continue` να ήταν στην επόμενη γραμμή και με εσοχή.

Επιτρέποντας στον χρήστη να επιλέξει το όνομα του αρχείου

Πραγματικά, δεν θέλουμε να πρέπει να επεξεργαζόμαστε τον κώδικά μας σε Python κάθε φορά που θέλουμε να επεξεργαστούμε ένα διαφορετικό αρχείο. Θα ήταν πιο βολικό να ζητάμε από τον χρήστη να εισάγει τη συμβολοσειρά ονόματος αρχείου, κάθε φορά που εκτελείται το πρόγραμμα, ώστε να μπορεί να χρησιμοποιήσει το πρόγραμμά μας σε διαφορετικά αρχεία, χωρίς να αλλάξει τον κώδικα της Python.

Αυτό είναι πολύ απλό να υλοποιηθεί, διαβάζοντας το όνομα του αρχείου από τον χρήστη, χρησιμοποιώντας την `input` ως εξής:

```
fname = input('Εισαγάγετε το όνομα του αρχείου: ')
fhand = open(fname)
πλήθος = 0
for γραμμή in fhand:
    if γραμμή.startswith('Subject:'):
        πλήθος = πλήθος + 1
print('Υπάρχουν ', πλήθος, ' γραμμές θέματος στο ', fname)
```

#Code: http://www.gr.py4e.com/code3/search6.py

Διαβάζουμε το όνομα του αρχείου από τον χρήστη και το τοποθετούμε σε μια μεταβλητή με το όνομα `fname` και ανοίγουμε αυτό το αρχείο. Τώρα μπορούμε να εκτελέσουμε το πρόγραμμα, επανειλημμένα, για διαφορετικά αρχεία.

```
python search6.py
Εισαγάγετε το όνομα του αρχείου: mbox.txt
Υπάρχουν 1797 γραμμές θέματος στο mbox.txt

python search6.py
```

```
Εισαγάγετε το όνομα του αρχείου: mbox-short.txt
Υπάρχουν 27 γραμμές θέματος στο mbox-short.txt
```

Πριν κρυφοκοιτάξετε την επόμενη ενότητα, ρίξτε μια ματιά στο παραπάνω πρόγραμμα και αναρωτηθείτε, "Τι θα μπορούσε να πάει στραβά εδώ;" ή "Τι μπορεί να κάνει ο φιλικός χρήστης μας, που θα έκανε το όμορφο μικρό μας πρόγραμμα να σταματήσει, άχαρα, την εκτέλεσή του με ένα traceback, κάνοντάς μας να φαινόμαστε όχι και τόσο καλοί στα μάτια των χρηστών μας;"

Χρήση try, except, και open

Σας είπα να μην κρυφοκοιτάξετε. Αυτή είναι η τελευταία σας ευκαιρία.

Τι γίνεται αν ο χρήστης μας πληκτρολογήσει κάτι που δεν είναι όνομα αρχείου;

```
python search6.py
Εισαγάγετε το όνομα του αρχείου: missing.txt
Traceback (most recent call last):
  File "search6.py", line 2, in <module>
    fhand = open(fname)
FileNotFoundError: [Errno 2] No such file or directory: 'missing.txt'

python search6.py
Εισαγάγετε το όνομα του αρχείου: na na boo boo
Traceback (most recent call last):
  File "search6.py", line 2, in <module>
    fhand = open(fname)
FileNotFoundError: [Errno 2] No such file or directory: 'na na boo boo'
```

Μην γελάτε. Οι χρήστες θα κάνουν τελικά ό,τι είναι δυνατό για να κολλήσουν τα προγράμματά σας, είτε κατά λάθος είτε με κακόβουλη πρόθεση. Στην πραγματικότητα, ένα σημαντικό μέρος οποιασδήποτε ομάδας ανάπτυξης λογισμικού είναι ένα άτομο ή μια ομάδα ατόμων, που ονομάζεται *Διασφάλιση Ποιότητας* (*Quality Assurance* ή QA για συντομία), των οποίων η ίδια δουλειά είναι να κάνουν τα πιο τρελά πράγματα σε μια προσπάθεια να "σπάσουν" το λογισμικό που ο προγραμματιστής δημιούργησε.

Η ομάδα QA είναι υπεύθυνη για την εύρεση των ελαττωμάτων στα προγράμματα προτού παραδώσουμε το πρόγραμμα στους τελικούς χρήστες, που μπορεί να αγοράσουν το λογισμικό ή να πληρώσουν το μισθό μας για τη σύνταξη του λογισμικού.

Έτσι, η ομάδα QA είναι ο καλύτερος φίλος του προγραμματιστή.

Τώρα λοιπόν που βλέπουμε το ψεγάδι στο πρόγραμμα, μπορούμε να το διορθώσουμε, κομψά, χρησιμοποιώντας τη δομή try/except. Πρέπει να αντιληφθούμε ότι η κλήση της open ενδέχεται να αποτύχει και να προσθέσουμε κατάλληλο κώδικα ανάκτησης όταν η open αποτύχει ως εξής:

```
fname = input('Εισαγάγετε το όνομα του αρχείου: ')
try:
    fhand = open(fname)
except:
    print('Δεν είναι δυνατό το άνοιγμα του αρχείου:', fname)
    exit()
πλήθος = 0
for γραμμή in fhand:
    if γραμμή.startswith('Subject:'):
        πλήθος = πλήθος + 1
print('Υπάρχουν ', πλήθος, ' γραμμές θέματος στο ', fname)
```

#Code: http://www.gr.py4e.com/code3/search7.py

Η συνάρτηση exit τερματίζει το πρόγραμμα. Είναι μια συνάρτηση που καλούμε και δεν επιστρέφει ποτέ. Τώρα, όταν ο χρήστης μας (ή η ομάδα QA) πληκτρολογεί ανόητα ή λάθος ονόματα αρχείων, τα "πιάνουμε" και τα ξεπερνάμε με χάρη:

```
python search7.py
Εισαγάγετε το όνομα του αρχείου: mbox.txt
Υπάρχουν 1797 γραμμές θέματος στο mbox.txt

python search7.py
Εισαγάγετε το όνομα του αρχείου: na na boo boo
Δεν είναι δυνατό το άνοιγμα του αρχείου: na na boo boo
```

Η προστασία της κλήσης open είναι ένα καλό παράδειγμα της σωστής χρήσης των try και except σε ένα πρόγραμμα Python. Χρησιμοποιούμε τον όρο "Pythonic" όταν κάνουμε κάτι με τον "τρόπο της Python". Θα μπορούσαμε να πούμε ότι το παραπάνω παράδειγμα είναι ο Pythonic τρόπος για να ανοίξετε ένα αρχείο.

Μόλις γίνετε πιο ειδικοί στην Python, μπορείτε να συμμετάσχετε σε μονομαχίες με άλλους προγραμματιστές Python για να αποφασίσετε ποια από τις δύο ισοδύναμες λύσεις σε ένα πρόβλημα είναι "πιο Python". Ο στόχος να είσαι «πιο Pythonic»

αντικατοπτρίζει την ιδέα ότι ο προγραμματισμός είναι εν μέρει μηχανική και εν μέρει τέχνη. Δεν μας ενδιαφέρει πάντα να κάνουμε κάτι που λειτουργεί, θέλουμε επίσης η λύση μας να είναι κομψή και να εκτιμάται ως κομψή από τους ομότιμούς μας.

Γραφή σε αρχεία

Για να γράψετε ένα αρχείο, πρέπει να το ανοίξετε με τη λειτουργία "w" ως δεύτερη παράμετρο:

```
>>> fout = open('output.txt', 'w')
>>> print(fout)
<_io.TextIOWrapper name='output.txt' mode='w' encoding='cp1252'>
```

Εάν το αρχείο υπάρχει ήδη, το άνοιγμα του σε λειτουργία εγγραφής διαγράφει τα προηγούμενα δεδομένα και τα αντικαθιστά με τα καινούρια, οπότε να είστε προσεκτικοί! Εάν το αρχείο δεν υπάρχει, δημιουργείται ένα νέο.

Η μέθοδος `write` του αντικειμένου χειρισμού αρχείου τοποθετεί δεδομένα στο αρχείο, επιστρέφοντας τον αριθμό των χαρακτήρων που γράφτηκαν. Η προεπιλεγμένη λειτουργία εγγραφής είναι κείμενο για εγγραφή (και ανάγνωση) συμβολοσειρών.

```
>>> γραμμή1 = "Αυτό το κλαδί, είναι\n"
>>> fout.write(γραμμή1)
21
```

Και πάλι, το αντικείμενο αρχείου παρακολουθεί τη θέση του, οπότε αν καλέσετε ξανά την `write`, προσθέτει τα νέα δεδομένα στο τέλος.

Πρέπει να φροντίσουμε να διαχειριζόμαστε τα άκρα των γραμμών, καθώς γράφουμε στο αρχείο, εισάγοντας ρητά τον χαρακτήρα νέας γραμμής όταν θέλουμε να τερματίσουμε μια γραμμή. Η εντολή `print` προσθέτει αυτόματα μια νέα γραμμή, αλλά η μέθοδος `write` δεν προσθέτει τη νέα γραμμή αυτόματα.

```
>>> γραμμή2 = 'το έμβλημα του τόπου μας.\n'
>>> fout.write(γραμμή2)
26
```

Όταν ολοκληρώσετε τη σύνταξη, πρέπει να κλείσετε το αρχείο για να βεβαιωθείτε ότι και το τελευταίο bit δεδομένων είναι φυσικά γραμμένο στο δίσκο, ώστε να μην χαθεί εάν διακοπεί η τροφοδοσία ρεύματος.

```
>>> fout.close()
```

Θα μπορούσαμε να κλείσουμε και τα αρχεία που ανοίγουμε για ανάγνωση, αλλά μπορεί να είμαστε και λίγο απρόσεκτοι αν μόνο ανοίγουμε κάποια αρχεία, καθώς η Python φροντίζει ώστε όλα τα ανοιχτά αρχεία να κλείνουν όταν τελειώνει το πρόγραμμα. Όταν γράφουμε όμως σε αρχεία, θέλουμε να τα κλείνουμε ξεκάθαρα, για να μην αφήνουμε τίποτα στην τύχη.

Εκσφαλμάτωση

Όταν διαβάζετε και γράφετε αρχεία, ενδέχεται να αντιμετωπίσετε προβλήματα με τους λευκούς χαρακτήρες. Αυτά τα σφάλματα μπορεί να είναι δύσκολο να εντοπιστούν, επειδή τα κενά, τα tab και οι νέες γραμμές είναι συνήθως αόρατα:

```
>>> s = '1 2\t 3\n 4'
>>> print(s)
1 2  3
 4
```

Η ενσωματωμένη συνάρτηση repr μπορεί να βοηθήσει. Λαμβάνει οποιοδήποτε αντικείμενο ως όρισμα και επιστρέφει μια παράσταση συμβολοσειράς του αντικειμένου. Για συμβολοσειρές, αναπαριστά τους χαρακτήρες κενού διαστήματος με ακολουθίες ανάστροφης κάθετης:

```
>>> print(repr(s))
'1 2\t 3\n 4'
```

Αυτό μπορεί να είναι χρήσιμο για τον εντοπισμό σφαλμάτων.

Ένα άλλο πρόβλημα που μπορεί να αντιμετωπίσετε είναι ότι διαφορετικά συστήματα χρησιμοποιούν διαφορετικούς χαρακτήρες για να υποδείξουν το τέλος μιας γραμμής. Ορισμένα συστήματα χρησιμοποιούν μια νέα γραμμή, που αναπαρίσταται με "\n". Άλλα χρησιμοποιούν έναν χαρακτήρα επιστροφής, που αναπαρίσταται με «\r». Κάποια χρησιμοποιούν και τα δύο. Εάν μετακινείτε αρχεία μεταξύ διαφορετικών συστημάτων, αυτές οι ασυνέπειες ενδέχεται να προκαλέσουν προβλήματα.

Για τα περισσότερα συστήματα, υπάρχουν εφαρμογές για μετατροπή από τη μια μορφή στην άλλη. Μπορείτε να τα βρείτε (και να διαβάσετε περισσότερα για αυτό το ζήτημα) στη διεύθυνση https://www.wikipedia.org/wiki/Newline. Ή, φυσικά, θα μπορούσατε να γράψετε μία μόνοι σας.

Γλωσσάριο

catch : Για να αποτρέψετε μια εξαίρεση από τον τερματισμό ενός προγράμματος χρησιμοποιώντας την εντολή `try` και `except`.

Pythonic : Μια τεχνική που λειτουργεί κομψά στην Python. "Η χρήση του try και except είναι ο *Pythonic* τρόπος ανάκτησης από αρχεία που λείπουν".

Quality Assurance - Διασφάλιση Ποιότητας : Ένα άτομο ή μια ομάδα που επικεντρώνεται στη διασφάλιση της συνολικής ποιότητας ενός προϊόντος λογισμικού. Το QA συχνά εμπλέκεται στη δοκιμή ενός προϊόντος και στον εντοπισμό προβλημάτων πριν από την κυκλοφορία του προϊόντος.

αρχείο κειμένου : Μια ακολουθία χαρακτήρων που είναι αποθηκευμένοι σε μονάδα μόνιμη αποθήκευση, όπως ένας σκληρός δίσκος.

νέα γραμμή : Ένας ειδικός χαρακτήρας που χρησιμοποιείται σε αρχεία και συμβολοσειρές για να υποδείξει το τέλος μιας γραμμής.

Ασκήσεις

Άσκηση 1: Γράψτε ένα πρόγραμμα για να διαβάσετε ένα αρχείο και να εκτυπώσετε τα περιεχόμενα του αρχείου (γραμμή προς γραμμή) όλα με κεφαλαία. Η εκτέλεση του προγράμματος θα έχει ως εξής:

```
python shout.py
Εισαγάγετε το όνομα του αρχείου: mbox-short.txt
FROM STEPHEN.MARQUARD@UCT.AC.ZA SAT JAN  5 09:14:16 2008
RETURN-PATH: <POSTMASTER@COLLAB.SAKAIPROJECT.ORG>
RECEIVED: FROM MURDER (MAIL.UMICH.EDU [141.211.14.90])
     BY FRANKENSTEIN.MAIL.UMICH.EDU (CYRUS V2.3.8) WITH LMTPA;
     SAT, 05 JAN 2008 09:14:16 -0500
```

Μπορείτε να κατεβάσετε το αρχείο από www.gr.py4e.com/code3/mbox-short.txt

Άσκηση 2: Γράψτε ένα πρόγραμμα που να δέχεται ένα όνομα αρχείου και, στη συνέχεια, να διαβάζει το αρχείο αυτό και να αναζητά γραμμές της μορφής:

```
X-DSPAM-Confidence: 0.8475
```

Όταν συναντήσετε μια γραμμή που ξεκινά με "X-DSPAM-Confidence:" διασπάστε τη γραμμή για να εξαγάγετε τον αριθμό κινητής υποδιαστολής στη γραμμή.

Μετρήστε αυτές τις γραμμές και στη συνέχεια υπολογίστε το σύνολο των τιμών "spam confidence" από αυτές τις γραμμές. Όταν φτάσετε στο τέλος του αρχείου, εκτυπώστε τη μέση τιμή των τιμών "spam confidence".

```
Εισαγάγετε το όνομα του αρχείου: mbox.txt
Μέσο spam confidence: 0.894128046745

Εισαγάγετε το όνομα του αρχείου: mbox-short.txt
Μέσο spam confidence: 0.750718518519
```

Δοκιμάστε το αρχείο σας στα αρχεία *mbox.txt* και *mbox-short.txt*.

Άσκηση 3: Μερικές φορές, όταν οι προγραμματιστές βαριούνται ή θέλουν να διασκεδάσουν λίγο, προσθέτουν ένα αβλαβές *Πασχαλινό Αυγό (Easter Egg)* στο πρόγραμμά τους. Τροποποιήστε το πρόγραμμα που ζητά από τον χρήστη το όνομα του αρχείου, έτσι ώστε να εκτυπώνει ένα αστείο μήνυμα όταν ο χρήστης πληκτρολογεί το ακριβές όνομα αρχείου "na na boo boo". Το πρόγραμμα θα πρέπει να συμπεριφέρεται κανονικά για όλα τα άλλα αρχεία που υπάρχουν και δεν υπάρχουν. Ακολουθεί ένα δείγμα εκτέλεσης του προγράμματος:

```
python egg.py
Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

python egg.py
Εισαγάγετε το όνομα του αρχείου: missing.tyxt
Δεν είναι δυνατό το άνοιγμα του αρχείου: missing.tyxt

python egg.py
Εισαγάγετε το όνομα του αρχείου: na na boo boo
NA NA BOO BOO TO YOU - You have been punk'd!
```

Δεν σας ενθαρρύνουμε να βάλετε τα "Easter Eggs" στα προγράμματά σας. Αυτό είναι απλώς μια άσκηση.

Κεφάλαιο 8

Λίστες

Μια λίστα είναι μια ακολουθία

Όπως και μια συμβολοσειρά, μια λίστα (*list)* είναι μια ακολουθία τιμών. Σε μια συμβολοσειρά, οι τιμές είναι χαρακτήρες ενώ σε μια λίστα, μπορούν να είναι οποιουδήποτε τύπου. Οι τιμές στη λίστα ονομάζονται *στοιχεία* (*elements* ή μερικές φορές *items*).

Υπάρχουν διάφοροι τρόποι για να δημιουργήσετε μια νέα λίστα. Ο πιο απλός είναι να περικλείσετε τα στοιχεία σε αγκύλες ("[" και "]"):

```
[10, 20, 30, 40]
['crunchy frog', 'ram bladder', 'lark vomit']
```

Το πρώτο παράδειγμα είναι μια λίστα τεσσάρων ακεραίων αριθμών. Η δεύτερη είναι μια λίστα τριών συμβολοσειρών. Τα στοιχεία μιας λίστας δεν χρειάζεται να είναι του ίδιου τύπου. Η ακόλουθη λίστα περιέχει μια συμβολοσειρά, ένα δεκαδικό, έναν ακέραιο, και (ορίστε!) μια νέα λίστα:

```
['spam', 2.0, 5, [10, 20]]
```

Μια λίστα σε μια άλλη λίστα είναι *εμφωλευμένη*.

Μια λίστα που δεν περιέχει στοιχεία ονομάζεται κενή λίστα. Μπορείτε να δημιουργήσετε μία με κενές αγκύλες, `[]`.

Όπως θα περίμενε κανείς, μπορείτε να εκχωρήσετε τιμές λίστας σε μεταβλητές:

```
>>> τυριά = ['Cheddar', 'Edam', 'Gouda']
>>> αριθμοί = [17, 123]
>>> κενή = []
>>> print(τυριά, αριθμοί, κενή)
['Cheddar', 'Edam', 'Gouda'] [17, 123] []
```

Οι λίστες είναι μεταβαλλόμενες

Η σύνταξη, για την πρόσβαση στα στοιχεία μιας λίστας, είναι η ίδια με αυτήν της πρόσβασης στους χαρακτήρες μιας συμβολοσειράς: ο τελεστής αγκύλης. Η έκφραση μέσα στις αγκύλες καθορίζει το δείκτη. Θυμηθείτε ότι οι δείκτες ξεκινούν από το 0:

```
>>> print(τυριά[0])
Cheddar
```

Σε αντίθεση με τις συμβολοσειρές, οι λίστες είναι μεταβλητές, μπορείτε δηλαδή να αλλάξετε τη σειρά των στοιχείων μιας λίστας ή να εκχωρήσετε εκ νέου ένα στοιχείο σε μια λίστα. Όταν ο τελεστής αγκύλης εμφανίζεται στην αριστερή πλευρά μιας ανάθεσης, προσδιορίζει το στοιχείο της λίστας που θα τροποποιηθεί.

```
>>> numbers = [17, 123]
>>> numbers[1] = 5
>>> print(numbers)
[17, 5]
```

Το στοιχείο στη θέση ένα της `numbers`, που ήταν 123, είναι τώρα 5.

Μπορείτε να σκεφτείτε μια λίστα ως μια σχέση μεταξύ δεικτών και στοιχείων. Αυτή η σχέση ονομάζεται *χαρτογράφηση (mapping)*. Κάθε δείκτης "αντιστοιχίζεται" σε ένα από τα στοιχεία.

Οι δείκτες λιστών λειτουργούν με τον ίδιο τρόπο όπως οι δείκτες συμβολοσειρών:

- Οποιαδήποτε ακέραια έκφραση μπορεί να χρησιμοποιηθεί ως ευρετήριο.
- Εάν προσπαθήσετε να διαβάσετε ή να γράψετε ένα στοιχείο που δεν υπάρχει, λαμβάνετε ένα `IndexError`.
- Εάν ένας δείκτης έχει αρνητική τιμή, μετράει αντίστροφα από το τέλος της λίστας.

Ο τελεστής `in` λειτουργεί και σε λίστες.

```
>>> τυριά = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in τυριά
True
>>> 'Brie' in τυριά
False
```

Διάσχιση λίστα

Ο πιο συνηθισμένος τρόπος για να διασχίσετε τα στοιχεία μιας λίστας είναι με έναν βρόχο for. Η σύνταξη είναι η ίδια με τις συμβολοσειρές:

```
for τυρί in τυριά:
    print(τυρί)
```

Αυτή η μορφή λειτουργεί καλά εάν χρειάζεται μόνο να διαβάσετε τα στοιχεία της λίστας. Αλλά αν θέλετε να γράψετε ή να ενημερώσετε τα στοιχεία, χρειάζεστε τους δείκτες. Ένας συνηθισμένος τρόπος για να γίνει αυτό είναι ο συνδυασμός των συναρτήσεων range και len:

```
for i in range(len(αριθμοί)):
    αριθμοί[i] = αριθμοί[i] * 2
```

Αυτός ο βρόχος διασχίζει τη λίστα και ενημερώνει κάθε στοιχείο. Το len επιστρέφει το πλήθος των στοιχείων της λίστας. Το range επιστρέφει μια λίστα δεικτών από 0 έως n-1, όπου το n είναι το μήκος της λίστας. Κάθε φορά μέσω του βρόχου, το i λαμβάνει τον δείκτη του επόμενου στοιχείου. Η δήλωση ανάθεσης στο σώμα χρησιμοποιεί το i για να διαβάσει την παλιά τιμή του στοιχείου και να εκχωρήσει τη νέα τιμή.

Ένας βρόχος for σε μια κενή λίστα δεν εκτελεί ποτέ το σώμα:

```
for x in κενή:
    print('Αυτό δεν εκτελείται ποτέ.')
```

Αν και μια λίστα μπορεί να περιέχει μια άλλη λίστα, η ένθετη λίστα εξακολουθεί να υπολογίζεται ως ένα μεμονωμένο στοιχείο. Το μήκος της λίστας, στο παρακάτω παράδειγμα, είναι τέσσερα:

```
['spam', 1, ['Brie', 'Roquefort', 'Pol le Veq'], [1, 2, 3]]
```

Λειτουργίες λίστας

Ο τελεστής + συνενώνει λίστες:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
```

Ομοίως, ο τελεστής * επαναλαμβάνει μια λίστα πολλές φορές:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Το πρώτο παράδειγμα επαναλαμβάνεται την πρώτη λίστα τέσσερις φορές και το δεύτερο τρεις φορές.

Διαμέριση λίστας

Ο τελεστής διαμέρισης λειτουργεί και σε λίστες:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
['b', 'c']
>>> t[:4]
['a', 'b', 'c', 'd']
>>> t[3:]
['d', 'e', 'f']
```

Εάν παραλείψετε τον πρώτο δείκτη, το τμήμα ξεκινά από την αρχή της λίστας. Αν παραλείψετε τον δεύτερο, το τμήμα φτάνει μέχρι το τέλος της λίστας. Έτσι, εάν παραλείψετε και τους δύο, το τμήμα είναι αντίγραφο ολόκληρης της λίστας.

```
>>> t[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

Δεδομένου ότι οι λίστες είναι μεταβαλλόμενες, είναι συχνά χρήσιμο να δημιουργείτε ένα αντίγραφο πριν εκτελέσετε λειτουργίες που "ανακατεύουν", "σπάνε" ή τροποποιούν τις λίστες.

Ένας τελεστής διαμέρισης στο αριστερό μέλος μιας ανάθεσης μπορεί να ενημερώσει πολλά στοιχεία ταυτόχρονα:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3] = ['x', 'y']
>>> print(t)
['a', 'x', 'y', 'd', 'e', 'f']
```

Μέθοδοι λίστας

Η Python παρέχει μεθόδους που λειτουργούν σε λίστες. Για παράδειγμα, η append προσθέτει ένα νέο στοιχείο στο τέλος μιας λίστας:

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print(t)
['a', 'b', 'c', 'd']
```

Η extend παίρνει μια λίστα ως όρισμα και προσθέτει όλα τα στοιχεία της στην λίστα στην οποία εφαρμόστηκε:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> print(t1)
['a', 'b', 'c', 'd', 'e']
```

Αυτό το παράδειγμα δεν τροποποιεί το t2.

Η sort ταξινομεί τα στοιχεία της λίστας από το μικρότερο προς το μεγαλύτερο:

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> print(t)
['a', 'b', 'c', 'd', 'e']
```

Οι περισσότερες μέθοδοι λίστας είναι κενές. Τροποποιούν τη λίστα και επιστρέφουν None. Αν κατά λάθος γράψετε `t = t.sort()`, θα απογοητευτείτε με το αποτέλεσμα.

Διαγραφή στοιχείων

Υπάρχουν διάφοροι τρόποι για να διαγράψετε στοιχεία από μια λίστα. Εάν γνωρίζετε το ευρετήριο του στοιχείου που θέλετε ν διαγράψετε, μπορείτε να χρησιμοποιήσετε την pop:

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> print(t)
['a', 'c']
>>> print(x)
b
```

Η pop τροποποιεί τη λίστα και επιστρέφει το στοιχείο που αφαιρέθηκε. Εάν δεν δώσετε κάποιον δείκτη, διαγράφει και επιστρέφει το τελευταίο στοιχείο.

Εάν δεν χρειάζεστε την καταργημένη τιμή, μπορείτε να χρησιμοποιήσετε την εντολή `del`:

```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> print(t)
['a', 'c']
```

Εάν γνωρίζετε το στοιχείο που θέλετε να αφαιρέσετε (αλλά όχι το δείκτη του), μπορείτε να χρησιμοποιήσετε το `remove`:

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> print(t)
['a', 'c']
```

Η επιστρεφόμενη τιμή του `remove` είναι `None`.

Για να αφαιρέσετε περισσότερα από ένα στοιχεία, μπορείτε να χρησιμοποιήσετε το `del` με ένα δείκτη διαμέρισης:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del t[1:5]
>>> print(t)
['a', 'f']
```

Ως συνήθως, το τμήμα, που θα διαγραφεί περιλαμβάνει όλα τα στοιχεία μέχρι και πριν το δεύτερο δείκτη.

Λίστες και συναρτήσεις

Υπάρχει ένα πλήθος ενσωματωμένων συναρτήσεων που μπορούν να χρησιμοποιηθούν σε λίστες και που σας επιτρέπουν να εξετάζεται γρήγορα μια λίστα, χωρίς να γράφετε τους δικούς σας βρόχους:

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
```

```
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25
```

Η συνάρτηση `sum()` λειτουργεί μόνο όταν τα στοιχεία της λίστας είναι αριθμοί. Οι άλλες δύο συναρτήσεις (`max()`, `len()`, etc.) λειτουργούν και με λίστες συμβολοσειρών και άλλους τύπους, που μπορούν να είναι συγκρίσιμοι.

Θα μπορούσαμε να ξαναγράψουμε ένα προηγούμενο πρόγραμμα, που υπολόγιζε τον μέσο όρο μιας λίστας αριθμών που εισήγαγε ο χρήστης, χρησιμοποιώντας μια λίστα.

Πρώτα, το πρόγραμμα για τον υπολογισμό ενός μέσου όρου χωρίς λίστα:

```
σύνολο = 0
πλήθος = 0
while (True):
    είσοδος = input('Εισαγάγετε έναν αριθμό: ')
    if είσοδος == 'τέλος': break
    τιμή = float(είσοδος)
    σύνολο = σύνολο + τιμή
    πλήθος = πλήθος + 1

μέσοςΌρος = σύνολο / πλήθος
print('Μέσος Όρος:', μέσοςΌρος)
```

#Code: http://www.gr.py4e.com/code3/avenum.py

Σε αυτό το πρόγραμμα, έχουμε τις μεταβλητές πλήθος και σύνολο για να κρατήσουμε τον πλήθος και το τρέχον σύνολο των αριθμών που εισάγονται, καθώς ζητάμε επανειλημμένα από τον χρήστη την εισαγωγή ενόν αριθμού.

Θα μπορούσαμε απλά να αποθηκεύουμε κάθε αριθμό καθώς τον εισαγάγει ο χρήστης και να χρησιμοποιήσουμε ενσωματωμένες συναρτήσεις για να υπολογίσουμε το άθροισμα και το πλήθος, στο τέλος.

```
numlist = list()
while (True):
    είσοδος = input('Εισαγάγετε έναν αριθμό: ')
    if είσοδος == 'τέλος': break
    τιμή = float(είσοδος)
```

```
    numlist.append(τιμή)

μέσοςΌρος = sum(numlist) / len(numlist)
print('Μέσος Όρος:', μέσοςΌρος)
```

#Code: http://www.gr.py4e.com/code3/avelist.py

Δημιουργούμε μια κενή λίστα πριν ξεκινήσει ο βρόχος και, στη συνέχεια, κάθε φορά που έχουμε έναν αριθμό, τον προσθέτουμε στη λίστα. Στο τέλος του προγράμματος, απλά υπολογίζουμε το άθροισμα των αριθμών στη λίστα και το διαιρούμε με το πλήθος των αριθμών στη λίστα για να καταλήξουμε στον μέσο όρο.

Λίστες και συμβολοσειρές

Μια συμβολοσειρά είναι μια ακολουθία χαρακτήρων και μια λίστα είναι μια ακολουθία τιμών, αλλά μια λίστα χαρακτήρων δεν είναι ίδια με μια συμβολοσειρά. Για να μετατρέψετε μια συμβολοσειρά σε μια λίστα χαρακτήρων, μπορείτε να χρησιμοποιήσετε τη `list`:

```
>>> s = 'spam'
>>> t = list(s)
>>> print(t)
['s', 'p', 'a', 'm']
```

Επειδή το `list` είναι το όνομα μιας ενσωματωμένης συνάρτησης, θα πρέπει να αποφύγετε τη χρήση της ως όνομα μεταβλητής. Επίσης αποφεύγω το γράμμα "l' γιατί μοιάζει πάρα πολύ με τον αριθμό "1". Γι' αυτό λοιπόν χρησιμοποιώ το "t".

Η συνάρτηση `list` σπάει μια συμβολοσειρά σε μεμονωμένα γράμματα. Εάν θέλετε να χωρίσετε μια συμβολοσειρά σε λέξεις, μπορείτε να χρησιμοποιήσετε τη μέθοδο `split`:

```
>>> s = 'pining for the fjords'
>>> t = s.split()
>>> print(t)
['pining', 'for', 'the', 'fjords']
>>> print(t[2])
the
```

Αφού χρησιμοποιήσετε το `split` για να σπάσετε τη συμβολοσειρά σε μια λίστα λέξεων, μπορείτε να χρησιμοποιήσετε τον τελεστή ευρετηρίου (τετράγωνη αγκύλη) για να δείτε μια συγκεκριμένη λέξη στη λίστα.

Μπορείτε να καλέσετε το split με ένα προαιρετικό όρισμα, που ονομάζεται *οριοθέτης (delimiter)* που καθορίζει ποιοι χαρακτήρες θα χρησιμοποιηθούν ως διαχωριστικά λέξεων. Το ακόλουθο παράδειγμα χρησιμοποιεί μια παύλα ως οριοθέτη:

```
>>> s = 'spam-spam-spam'
>>> οριοθέτης = '-'
>>> s.split(οριοθέτης)
['spam', 'spam', 'spam']
```

Η join είναι το αντίστροφο του split. Παίρνει μια λίστα με συμβολοσειρές και συνενώνει τα στοιχεία της. Το join είναι μια μέθοδος συμβολοσειράς, επομένως πρέπει να την καλέσετε στον οριοθέτη και να μεταβιβάσετε τη λίστα ως παράμετρο:

```
>>> t = ['pining', 'for', 'the', 'fjords']
>>> οριοθέτης = ' '
>>> οριοθέτης.join(t)
'pining for the fjords'
```

Σε αυτήν την περίπτωση, ο οριοθέτης είναι ένας χαρακτήρας διαστήματος, επομένως η join βάζει ένα διάστημα μεταξύ των λέξεων. Για να συνδέσετε συμβολοσειρές χωρίς κενά, μπορείτε να χρησιμοποιήσετε την κενή συμβολοσειρά "", ως οριοθέτη.

Ανάλυση γραμμών

Συνήθως όταν διαβάζουμε ένα αρχείο θέλουμε να κάνουμε κάτι στις γραμμές του, πέρα από την απλή εκτύπωση ολόκληρης της γραμμής. Συχνά θέλουμε να βρούμε τις "ενδιαφέρουσες γραμμές" και μετά *να αναλύσουμε* την κάθε μία από αυτές, για να βρούμε κάποιο ενδιαφέρον *μέρος* της γραμμής. Τι θα γινόταν αν θέλαμε να εκτυπώσουμε την ημέρα της εβδομάδας από αυτές τις γραμμές που ξεκινούν με "From";

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

Η μέθοδος split είναι πολύ αποτελεσματική, όταν αντιμετωπίζετε τέτοιου είδους προβλήματα. Μπορούμε να γράψουμε ένα μικρό πρόγραμμα που αναζητά γραμμές, που ξεκινούν με "From", να διαχωρίσουμε (split) αυτές τις γραμμές και, στη συνέχεια, να εκτυπώσουμε την τρίτη λέξη στη γραμμή:

```
fhand = open('mbox-short.txt')
for γραμμή in fhand:
    γραμμή = γραμμή.rstrip()
```

```
    if not γραμμή.startswith('From '): continue
    λέξεις = γραμμή.split()
    print(λέξεις[2])
```

#Code: http://www.gr.py4e.com/code3/search5.py

Το πρόγραμμα παράγει την ακόλουθη έξοδο:

```
    Sat
    Fri
    Fri
    Fri
    ...
```

Αργότερα, θα μάθουμε όλο και πιο εξελιγμένες τεχνικές, για να επιλέγουμε τις γραμμές στις οποίες θα δουλέψουμε και πώς θα ξεχωρίσουμε αυτές τις γραμμές για να βρούμε το ακριβές κομμάτι των πληροφοριών που αναζητούμε.

Αντικείμενα και τιμές

Εάν εκτελέσουμε αυτές τις εντολές ανάθεσης:

```
a = 'banana'
b = 'banana'
```

Γνωρίζουμε ότι το a και το b αναφέρονται και τα δύο σε μια συμβολοσειρά, αλλά δεν ξέρουμε αν αναφέρονται στην *ίδια* συμβολοσειρά. Υπάρχουν δύο πιθανές καταστάσεις:

Εικόνα 8.1: Μεταβλητές και Αντικείμενα

Στο πρώτο σχήμα, τα a και b αναφέρονται σε δύο διαφορετικά αντικείμενα, που έχουν την ίδια τιμή. Στο δεύτερο σχήμα αναφέρονται στο ίδιο αντικείμενο.

Για να ελέγξετε αν δύο μεταβλητές αναφέρονται στο ίδιο αντικείμενο, μπορείτε να χρησιμοποιήσετε τον τελεστή `is`.

```
>>> a = 'banana'
>>> b = 'banana'
>>> a is b
True
```

Σε αυτό το παράδειγμα, η Python δημιούργησε μόνο ένα αντικείμενο συμβολοσειράς και το a και το b αναφέρονται σε αυτό.

Αλλά όταν δημιουργείτε δύο λίστες, δημιουργούνται δύο αντικείμενα:

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a is b
False
```

Σε αυτή την περίπτωση θα λέγαμε ότι οι δύο λίστες είναι *ισοδύναμες*, γιατί έχουν τα ίδια στοιχεία, αλλά όχι *ταυτόσημες*, μιας και είναι δύο διαφορετικά αντικείμενα. Αν δύο αντικείμενα είναι ταυτόσημα, είναι επίσης ισοδύναμα, αλλά αν είναι ισοδύναμα, δεν είναι απαραίτητα ταυτόσημα.

Μέχρι τώρα, χρησιμοποιούσαμε το "αντικείμενο" και την "τιμή" εναλλακτικά, αλλά είναι πιο ακριβές να πούμε ότι ένα αντικείμενο έχει μια τιμή. Εάν `a = [1,2,3]`, το a αναφέρεται σε ένα αντικείμενο λίστας του οποίου η τιμή είναι μια συγκεκριμένη ακολουθία στοιχείων. Εάν μια άλλη λίστα έχει τα ίδια στοιχεία, θα λέγαμε ότι έχει την ίδια τιμή.

Ψευδωνυμία

Εάν το a αναφέρεται σε ένα αντικείμενο και εκτελέσετε `b = a`, τότε και οι δύο μεταβλητές αναφέρονται στο ίδιο αντικείμενο:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
```

Ο συσχετισμός μιας μεταβλητής με ένα αντικείμενο ονομάζεται *αναφορά*. Σε αυτό το παράδειγμα, υπάρχουν δύο αναφορές στο ίδιο αντικείμενο.

Ένα αντικείμενο με περισσότερες από μία αναφορές, έχει περισσότερα από ένα ονόματα, οπότε λέμε ότι το αντικείμενο έχει *ψευδώνυμα*.

Εάν το αντικείμενο με ψευδώνυμα είναι μεταβαλλόμενο, οι αλλαγές που γίνονται με το ένα ψευδώνυμο επηρεάζουν το άλλο:

```
>>> b[0] = 17
>>> print(a)
[17, 2, 3]
```

Αν και αυτή η συμπεριφορά μπορεί να είναι χρήσιμη, είναι επιρρεπής σε σφάλματα. Γενικά, είναι ασφαλέστερο να αποφεύγετε τη ψευδωνυμία όταν εργάζεστε με μεταβαλλόμενα αντικείμενα.

Για αμετάβλητα αντικείμενα όπως οι συμβολοσειρές, η ψευδωνυμία δεν είναι τόσο σοβαρό πρόβλημα. Σε αυτό το παράδειγμα:

```
a = 'banana'
b = 'banana'
```

Σχεδόν ποτέ δεν έχει διαφορά εάν το a και το b αναφέρονται στην ίδια συμβολοσειρά ή όχι.

Ορίσματα λίστας

Όταν μεταβιβάζετε μια λίστα σε μια συνάρτηση, η συνάρτηση λαμβάνει μια αναφορά στη λίστα. Εάν η συνάρτηση τροποποιήσει μια παράμετρο λίστας, η αλλαγή πραγματοποιείται και στην λίστα που μεταβιβάστηκε ως όρισμα. Για παράδειγμα, το delete_head αφαιρεί το πρώτο στοιχείο από μια λίστα:

```
def delete_head(t):
    del t[0]
```

Δείτε πώς χρησιμοποιείται:

```
>>> γράμματα = ['a', 'b', 'c']
>>> delete_head(γράμματα)
>>> print(γράμματα)
['b', 'c']
```

Η παράμετρος t και η μεταβλητή γράμματα είναι ψευδώνυμα για το ίδιο αντικείμενο.

Είναι σημαντικό να γίνεται διάκριση μεταξύ λειτουργιών που τροποποιούν λίστες και λειτουργιών που δημιουργούν νέες λίστες. Για παράδειγμα, η μέθοδος append τροποποιεί μια λίστα, αλλά ο τελεστής + δημιουργεί μια νέα λίστα:

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> print(t1)
[1, 2, 3]
>>> print(t2)
None
```

```
>>> t3 = t1 + [3]
>>> print(t3)
[1, 2, 3]
>>> t2 is t3
False
```

Αυτή η διαφορά είναι σημαντική όταν γράφετε συναρτήσεις που υποτίθεται ότι τροποποιούν λίστες. Για παράδειγμα, αυτή η συνάρτηση *δεν* διαγράφει το πρώτο στοιχείο (θέση 0) μιας λίστας:

```
def bad_delete_head(t):
    t = t[1:]              # WRONG!
```

Ο τελεστής διαμέρισης (slice) δημιουργεί μια νέα λίστα και η ανάθεση κάνει το `t` να αναφέρεται σε αυτήν, αλλά τίποτα από αυτά δεν έχει καμία επίδραση στη λίστα που μεταβιβάστηκε ως όρισμα.

Μια εναλλακτική είναι να γράψετε μια συνάρτηση που δημιουργεί και επιστρέφει μια νέα λίστα. Για παράδειγμα, η `tail` επιστρέφει όλα εκτός από το πρώτο στοιχείο μιας λίστας:

```
def tail(t):
    return t[1:]
```

Αυτή η συνάρτηση αφήνει την αρχική λίστα χωρίς αμετάβλητη. Δείτε πώς χρησιμοποιείται:

```
>>> γράμματα = ['a', 'b', 'c']
>>> υπόλοιπο = tail(γράμματα)
>>> print(υπόλοιπο)
['b', 'c']
```

Άσκηση 1: Γράψτε μια συνάρτηση με όνομα `chop`, που δέχεται μια λίστα και την τροποποιεί, αφαιρώντας το πρώτο και το τελευταίο στοιχείο και επιστρέφει `None`. Στη συνέχεια, γράψτε μια συνάρτηση που ονομάζεται `middle`, που δέχεται μια λίστα και επιστρέφει μια νέα λίστα που περιέχει όλα τα στοιχεία της αρχική, εκτός από το πρώτο και το τελευταίο.

Εκσφαλμάτωση

Η απρόσεκτη χρήση λιστών (και άλλων μεταβαλλόμενων αντικειμένων) μπορεί να

οδηγήσει σε πολύωρη εκσφαλμάτωση. Ακολουθούν μερικές συνήθεις παγίδες και τρόποι για να τις αποφύγετε:

1. Μην ξεχνάτε ότι οι περισσότερες μέθοδοι λίστας τροποποιούν το όρισμα και επιστρέφουν None, αντίθετα από τις μεθόδους συμβολοσειράς, οι οποίες επιστρέφουν μια νέα συμβολοσειρά και αφήνουν το πρωτότυπο αναλλοίωτο.

   Εάν έχετε συνηθίσει να γράφετε κώδικα για συμβολοσειρές ως εξής:

   ```
   word = word.strip()
   ```

   Είναι, συχνά, δελεαστικό να γράψετε αντίστοιχο κώδικα και για λίστες:

   ```
   t = t.sort()            # ΛΑΘΟΣ!
   ```

   Επειδή η sort επιστρέφει None, η επόμενη λειτουργία που θα εκτελέσετε με το t είναι πιθανό να αποτύχει. Πριν χρησιμοποιήσετε μεθόδους λίστας και τελεστές, θα πρέπει να διαβάσετε προσεκτικά την τεκμηρίωση και στη συνέχεια να κάνετε δοκιμές σε διαδραστική λειτουργία. Οι μέθοδοι και οι τελεστές που είναι κοινές σε λίστες με άλλες ακολουθίες (όπως συμβολοσειρές) τεκμηριώνονται στη διεύθυνση:

   docs.python.org/library/stdtypes.html#common-sequence-operations

   Οι μέθοδοι και οι τελεστές που ισχύουν μόνο για μεταβαλλόμενες ακολουθίες τεκμηριώνονται στη διεύθυνση:

   docs.python.org/library/stdtypes.html#mutable-sequence-types

2. Διάλεξε ένα ιδίωμα και μείνε με αυτό.

   Μέρος του προβλήματος με τις λίστες είναι ότι υπάρχουν πάρα πολλοί τρόποι για να κάνετε πράγματα. Για παράδειγμα, για να αφαιρέσετε ένα στοιχείο από μια λίστα, μπορείτε να χρησιμοποιήσετε τις pop, remove, del ή ακόμα και μια εκχώρηση με τον τελεστή διαμέρισης.

   Για να προσθέσετε ένα στοιχείο, μπορείτε να χρησιμοποιήσετε τη μέθοδο append ή τον τελεστή +. Αλλά μην ξεχνάτε ότι αυτό είναι το σωστό:

   ```
   t.append(x)
   t = t + [x]
   ```

   Και αυτό είναι λάθος:

   ```
   t.append([x])           # ΛΑΘΟΣ!
   t = t.append(x)         # ΛΑΘΟΣ!
   ```

```
t + [x]                # ΛΑΘΟΣ!
t = t + x              # ΛΑΘΟΣ!
```

Δοκιμάστε καθένα από αυτά τα παραδείγματα σε διαδραστική λειτουργία για να βεβαιωθείτε ότι καταλαβαίνετε τι κάνουν. Σημειώστε ότι μόνο το τελευταίο προκαλεί σφάλμα χρόνου εκτέλεσης, τα άλλα τρία είναι συντακτικά σωστά, αλλά δεν έχουν το επιθυμητό αποτέλεσμα.

3. Δημιουργήστε αντίγραφα για να αποφύγετε τη ψευδωνυμία

   Εάν θέλετε να χρησιμοποιήσετε μια μέθοδο όπως η `sort`, που τροποποιεί το όρισμα, αλλά πρέπει να διατηρήσετε και την αρχική λίστα, μπορείτε να δημιουργήσετε ένα αντίγραφο.

```
orig = t[:]
t.sort()
```

   Σε αυτό το παράδειγμα, θα μπορούσατε επίσης να χρησιμοποιήσετε την ενσωματωμένη συνάρτηση `sorted`, η οποία επιστρέφει μια νέα, ταξινομημένη λίστα και αφήνει το πρωτότυπο αναλλοίωτο. Σε αυτήν όμως την περίπτωση, θα πρέπει να αποφύγετε τη χρήση του `sorted` ως όνομα μεταβλητής!

4. Λίστες, `split` και αρχεία

   Όταν διαβάζουμε και αναλύουμε αρχεία, υπάρχουν πολλές πιθανότητες να συναντήσουμε είσοδο που ενδέχεται να διακόψει το πρόγραμμά μας, επομένως είναι καλή ιδέα να επανεξετάσουμε το μοτίβο *guardian (κηδεμονίας)* όταν πρόκειται να γράψουμε προγράμματα που διαβάζουν από ένα αρχείο και να αναζητήσουμε μια "βελόνα στο άχυρα".

   Ας δούμε ξανά το πρόγραμμά μας, που αναζητά την ημέρα της εβδομάδας στις γραμμές του αρχείου μας:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

   Εφόσον χωρίζουμε αυτή τη γραμμή σε λέξεις, θα μπορούσαμε να παραιτηθούμε από τη χρήση του `startswith` και απλώς να δούμε την πρώτη λέξη της γραμμής για να προσδιορίσουμε αν μας ενδιαφέρει αυτή η γραμμή ή όχι. Μπορούμε να χρησιμοποιήσουμε το `continue` για να παραλείψουμε τις γραμμές που δεν έχουν το "From" ως πρώτη λέξη ως εξής:

```
fhand = open('mbox-short.txt')
for γραμμή in fhand:
```

```
    λέξεις = γραμμή.split()
    if λέξεις[0] != 'From' : continue
    print(λέξεις[2])
```

Αυτό φαίνεται πολύ πιο απλό και δεν χρειάζεται καν να χρησιμοποιήσουμε το `rstrip` για να αφαιρέσουμε το χαρακτήρα νέας γραμμής από το τέλος του αρχείου. Είναι όμως καλύτερο;

```
python search8.py
Sat
Traceback (most recent call last):
  File "search8.py", line 5, in <module>
    if λέξεις[0] != 'From' : continue
IndexError: list index out of range
```

Λειτουργεί αρχικά και βλέπουμε την ημέρα από την πρώτη γραμμή (Sat), αλλά μετά το πρόγραμμα αποτυγχάνει με ένα σφάλμα traceback. Τι πήγε στραβά; Ποια μπερδεμένα δεδομένα προκάλεσαν την αποτυχία του κομψό, έξυπνου και πολύ Pythonic προγράμματός μας;

Θα μπορούσατε να το ελέγχετε για πολλή ώρα και να μπερδευτείτε ή να ζητήσετε βοήθεια από κάποιον, αλλά η πιο γρήγορη και έξυπνη προσέγγιση είναι να προσθέσετε μια εντολή `print`. Το καλύτερο μέρος για να προσθέσετε την εντολή εκτύπωσης είναι ακριβώς πριν από τη γραμμή όπου το πρόγραμμα απέτυχε και να εκτυπώσετε τα δεδομένα που φαίνεται να προκαλούν την αποτυχία.

Τώρα αυτή η προσέγγιση μπορεί να παράξει πολλές γραμμές εξόδου, αλλά τουλάχιστον θα έχετε αμέσως κάποιες ενδείξεις για το πρόβλημα που αντιμετωπίζετε. Έτσι, προσθέτουμε μια εκτύπωση της μεταβλητής `words`, ακριβώς πριν από τη γραμμή πέντε. Προσθέτουμε ακόμη και ένα πρόθεμα "Εντοπισμός σφαλμάτων:" στη γραμμή, ώστε να μπορούμε να διατηρήσουμε την κανονική μας έξοδο ξεχωριστά από την έξοδο εντοπισμού σφαλμάτων.

```
for γραμμή in fhand:
    λέξεις = γραμμή.split()
    print('Εντοπισμός σφαλμάτων:', λέξεις)
    if λέξεις[0] != 'From' : continue
    print(λέξεις[2])
```

Όταν εκτελούμε το πρόγραμμα, πολλές εξόδοι κυλούν στο πάνω μέρος της οθόνης,

αλλά στο τέλος, βλέπουμε την έξοδο εντοπισμού σφαλμάτων και την σφάλμα traceback, ώστε να γνωρίζουμε τι συνέβη λίγο πριν από το σφάλμα.

```
Debug: ['X-DSPAM-Confidence:', '0.8475']
Debug: ['X-DSPAM-Probability:', '0.0000']
Debug: []
Traceback (most recent call last):
  File "search9.py", line 6, in <module>
    if λέξεις[0] != 'From' : continue
IndexError: list index out of range
```

Κάθε γραμμή εντοπισμού σφαλμάτων εκτυπώνει τη λίστα των λέξεων που λαμβάνουμε από τη `split`, όταν διαχωρίζουμε τη γραμμή σε λέξεις. Το πρόγραμμα αποτυγχάνει, όταν η λίστα των λέξεων είναι κενή `[]`. Αν ανοίξουμε το αρχείο σε ένα πρόγραμμα επεξεργασίας κειμένου και το ελέγξουμε, σε εκείνο το σημείο φαίνεται ως εξής:

```
X-DSPAM-Result: Innocent
X-DSPAM-Processed: Sat Jan  5 09:14:16 2008
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000

Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

Το σφάλμα παρουσιάζεται όταν το πρόγραμμά μας συναντήσει μια κενή γραμμή! Φυσικά, υπάρχουν μηδέν `λέξεις` σε μια κενή γραμμή. Γιατί δεν το σκεφτήκαμε όταν γράφαμε τον κώδικα; Όταν ο κώδικας αναζητά την πρώτη λέξη (`λέξη[0]`) για να ελέγξει αν ταιριάζει με το "From", λαμβάνουμε το σφάλμα "index out of range".

Αυτό φυσικά είναι το τέλειο μέρος για να προσθέσετε κάποιον κώδικα *κηδεμόνα*, για να αποφύγετε τον έλεγχο της πρώτης λέξης εάν δεν υπάρχει πρώτη λέξη. Υπάρχουν πολλοί τρόποι προστασίας αυτού του κώδικα. θα επιλέξουμε να ελέγξουμε τον αριθμό των λέξεων που έχουμε πριν επιχειρήσουμε να προσπελάσουμε την πρώτη λέξη:

```
fhand = open('mbox-short.txt')
for γραμμή in fhand:
    λέξεις = γραμμή.split()
    # print('Εντοπισμός σφαλμάτων:', λέξεις)
    if len(λέξεις) == 0 : continue
```

```
    if λέξεις[0] != 'From' : continue
    print(λέξεις[2])
```

Πρώτα μετατρέψαμε σε σχόλιο την εντολή εκτύπωσης εντοπισμού σφαλμάτων, αντί να την αφαιρέσουμε, σε περίπτωση που η τροποποίησή μας αποτύχει και χρειαστεί εκσφαλμάτωση ξανά. Στη συνέχεια, προσθέσαμε μια δήλωση κηδεμόνα, που ελέγχει αν έχουμε μηδενικές λέξεις και, αν ναι, χρησιμοποιούμε το `continue` για να μεταβούμε στην επόμενη γραμμή του αρχείου.

Οι δύο δηλώσεις `continue` μας βοηθούν να φιλτράρουμε το σύνολο των γραμμών που μας "ενδιαφέρουν" και τις οποίες θέλουμε να επεξεργαστούμε λίγο περισσότερο. Μια γραμμή που δεν έχει λέξεις μας είναι "αδιάφορη", οπότε την αγνοούμε και μεταβαίνουμε στην επόμενη γραμμή. Μια γραμμή που δεν έχει ως πρώτη λέξη το "From" δεν μας ενδιαφέρει, οπότε την παρακάμπτουμε κι αυτήν.

Το πρόγραμμα όπως τροποποιήθηκε εκτελείται με επιτυχία, οπότε ίσως είναι σωστό. Η δήλωση του κηδεμόνα μας διασφαλίζει ότι το `λέξεις[0]` δεν θα αποτύχει ποτέ, αλλά ίσως δεν είναι αρκετό. Όταν προγραμματίζουμε, πρέπει πάντα να σκεφτόμαστε, "Τι μπορεί να πάει στραβά;"

Άσκηση 2: Εντοπίστε ποια γραμμή, του παραπάνω προγράμματος, εξακολουθεί να μην προστατεύεται σωστά. Δείτε εάν μπορείτε να δημιουργήσετε ένα αρχείο κειμένου που προκαλεί την αποτυχία προγράμματός μας και, στη συνέχεια, τροποποιήστε το πρόγραμμα έτσι ώστε η γραμμή να προστατεύεται σωστά και δοκιμάστε το για να βεβαιωθείτε ότι χειρίζεται χωρίς σφάλματα το νέο αρχείο κειμένου σας.

Άσκηση 3: Ξαναγράψτε τον κώδικα κηδεμόνα του παραπάνω παραδείγματος, χωρίς τις δύο εντολές `if`. Αντ' αυτών, χρησιμοποιήστε μια σύνθετη λογική έκφραση, χρησιμοποιώντας τον λογικό τελεστή `or` με μία μόνο εντολή `if`.

Γλωσσάριο

αναφορά : Η συσχέτιση μεταξύ μιας μεταβλητής και της τιμής της.

αντικείμενο : Κάτι στο οποίο μπορεί να αναφέρεται μια μεταβλητή. Ένα αντικείμενο έχει έναν τύπο και μια τιμή.

δείκτης : Μια ακέραια τιμή που υποδεικνύει ένα στοιχείο σε μια λίστα.

διάσχιση λίστας – list traversal : Η διαδοχική πρόσβαση σε κάθε στοιχείο μιας λίστας.

εμφωλευμένη λίστα : Μια λίστα που είναι στοιχείο μιας άλλης λίστας.

ισοδύναμο : Έχει την ίδια τιμή.

λίστα : Μία ακολουθία τιμών.

οριοθέτης : Ένας χαρακτήρας ή συμβολοσειρά που χρησιμοποιείται για να υποδείξει πού πρέπει να χωριστεί μια συμβολοσειρά.

στοιχείο : Μία από τις τιμές σε μια λίστα (ή άλλη ακολουθία).

ταυτόσημο : Είναι το ίδιο αντικείμενο (που συνεπάγεται ισοδύναμο).

ψευδωνυμία : Μια περίσταση όπου δύο ή περισσότερες μεταβλητές αναφέρονται στο ίδιο αντικείμενο.

Ασκήσεις

Άσκηση 4: Βρείτε όλες τις μοναδικές λέξεις σε ένα αρχείο

Ο Σαίξπηρ χρησιμοποίησε πάνω από 20.000 λέξεις στα έργα του. Πώς όμως θα το υπολόγιζες αυτό; Πώς θα δημιουργήσατε τη λίστα με όλες τις λέξεις που χρησιμοποίησε ο Σαίξπηρ; Θα κατεβάζατε όλο το έργο του, θα το διαβάσατε και θα εντοπίζατε όλες τις μοναδικές λέξεις με το χέρι;

Ας χρησιμοποιήσουμε την Python για να το πετύχουμε αυτό. Καταγράψτε όλες τις μοναδικές λέξεις, ταξινομημένες με αλφαβητική σειρά, που είναι αποθηκευμένες στο αρχείο `romeo.txt`, που περιέχει ένα υποσύνολο του έργου του Σαίξπηρ.

Για να ξεκινήσετε, κατεβάστε ένα αντίγραφο του αρχείου www.gr.py4e.com/code3/romeo.txt. Δημιουργήστε μια λίστα,στην οποία να εμφανίζεται κάθε λέξη μία μόνο φορά, η οποία θα περιέχει το τελικό αποτέλεσμα. Γράψτε ένα πρόγραμμα για να ανοίξετε το αρχείο `romeo.txt` και να το διαβάσετε γραμμή προς γραμμή. Για κάθε γραμμή, χωρίστε την σε μια λίστα λέξεων χρησιμοποιώντας τη συνάρτηση `split`. Για κάθε λέξη, ελέγξτε αν η λέξη περιέχεται ήδη στη λίστα με τις μοναδικές λέξεις. Εάν η λέξη δεν περιέχεται στη λίστα με τις μοναδικές λέξεις, προσθέστε τη στη λίστα. Όταν ολοκληρωθεί το πρόγραμμα, ταξινομήστε και εκτυπώστε τη λίστα με τις μοναδικές λέξεις, σε αλφαβητική σειρά.

```
Εισαγάγετε το αρχείο: romeo.txt
```

```
['Arise', 'But', 'It', 'Juliet', 'Who', 'already',
'and', 'breaks', 'east', 'envious', 'fair', 'grief',
'is', 'kill', 'light', 'moon', 'pale', 'sick', 'soft',
'sun', 'the', 'through', 'what', 'window',
'with', 'yonder']
```

Άσκηση 5: Μινιμαλιστικός Εξυπηρετητής Email (Email Client).

Το MBOX (mail box) είναι μια δημοφιλής μορφή αρχείου για αποθήκευση και κοινή χρήση μιας συλλογής email. Αυτό χρησιμοποιήθηκε από πρώιμους διακομιστές email και εφαρμογές επιτραπέζιου υπολογιστή. Χωρίς να μπαίνουμε σε πάρα πολλές λεπτομέρειες, το MBOX είναι ένα αρχείο κειμένου, στο οποίο αποθηκεύονται διαδοχικά email. Τα email διαχωρίζονται από μια ειδική γραμμή που ξεκινά με `From` (προσέξτε το διάστημα). Είναι σημαντικό ότι οι γραμμές που ξεκινούν με `From:` (προσέξτε την άνω και κάτω τελεία) περιγράφουν το ίδιο το email και δεν λειτουργούν ως διαχωριστικά. Φανταστείτε ότι έχετε γράψει μια μινιμαλιστική εφαρμογή email, η οποία αναφέρει τα email των αποστολέων στα Εισερχόμενα του χρήστη και μετράει τον αριθμό των email.

Γράψτε ένα πρόγραμμα που διαβάζει τα δεδομένα του γραμματοκιβωτίου και όταν εντοπίσει γραμμή που ξεκινά με "From ", χωρίζει τη γραμμή σε λέξεις, χρησιμοποιώντας τη συνάρτηση `split`. Μας ενδιαφέρει ποιος έστειλε το μήνυμα, που είναι η δεύτερη λέξη στη γραμμή From".

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

Θα αναλύει τη γραμμή From και θα εκτυπώνει τη δεύτερη λέξη κάθε γραμμής From, στη συνέχεια θα μετράει τον αριθμό των γραμμών From (όχι From:) και θα εκτυπώνει, στο τέλος, το πλήθος τους. Αυτό είναι ένα καλό δείγμα εξόδου με μερικές γραμμές που έχουν αφαιρεθεί:

```
python fromcount.py
Εισαγάγετε ένα όνομα αρχείου: mbox-short.txt
stephen.marquard@uct.ac.za
louis@media.berkeley.edu
zqian@umich.edu

[...κάποια έξοδος αφαιρέθηκε...]
```

```
ray@media.berkeley.edu
cwen@iupui.edu
cwen@iupui.edu
cwen@iupui.edu
Βρέθηκαν 27 γραμμές στο αρχείο με πρώτη λέξη το From
```

Άσκηση 6: Ξαναγράψτε το πρόγραμμα που ζητά από τον χρήστη μια λίστα με αριθμούς και εκτυπώνει το μέγιστο και το ελάχιστο των αριθμών, στο τέλος, όταν ο χρήστης εισάγει "τέλος". Τροποποιήστε το πρόγραμμα ώστε να αποθηκεύει τους αριθμούς, που εισάγει ο χρήστης, σε μια λίστα και χρησιμοποιήστε τις συναρτήσεις `max()` και `min()` για να υπολογίσετε τον μέγιστο και τον ελάχιστο αριθμό μετά την ολοκλήρωση του βρόχου.

```
Εισαγάγετε έναν αριθμό: 6
Εισαγάγετε έναν αριθμό: 2
Εισαγάγετε έναν αριθμό: 9
Εισαγάγετε έναν αριθμό: 3
Εισαγάγετε έναν αριθμό: 5
Εισαγάγετε έναν αριθμό: τέλος
Μέγιστο: 9.0
Ελάχιστο: 2.0
```

Κεφάλαιο 9

Λεξικά

Ένα *λεξικό (dictionary)* είναι σαν μια λίστα, αλλά πιο γενικό. Σε μια λίστα, οι δείκτες θέσης πρέπει να είναι ακέραιοι, ενώ σε ένα λεξικό, οι δείκτες μπορούν να είναι (σχεδόν) οποιουδήποτε τύπου.

Μπορείτε να σκεφτείτε ένα λεξικό ως μια αντιστοίχιση μεταξύ ενός συνόλου δεικτών (που ονομάζονται *κλειδιά - key*) και ενός συνόλου τιμών. Κάθε κλειδί αντιστοιχίζεται σε μια τιμή. Η συσχέτιση ενός κλειδιού και μιας τιμής ονομάζεται *ζεύγος κλειδιού-τιμής (key-value pair)* ή μερικές φορές *στοιχείο*.

Για παράδειγμα, θα δημιουργήσουμε ένα λεξικό που αντιστοιχίζει λέξεις από τα αγγλικά στα ισπανικά, έτσι ώστε τα κλειδιά και οι τιμές να είναι όλα συμβολοσειρές.

Η συνάρτηση `dict` δημιουργεί ένα νέο λεξικό χωρίς στοιχεία. Επειδή το `dict` είναι το όνομα μιας ενσωματωμένης συνάρτησης, θα πρέπει να αποφύγετε τη χρήση του ως όνομα μεταβλητής.

```
>>> eng2sp = dict()
>>> print(eng2sp)
{}
```

Τα άγκιστρα, `{}`, αντιπροσωπεύουν ένα κενό λεξικό. Για να προσθέσετε στοιχεία στο λεξικό, μπορείτε να χρησιμοποιήσετε αγκύλες:

```
>>> eng2sp['one'] = 'uno'
```

Αυτή η γραμμή δημιουργεί ένα στοιχείο που αποτελείται από το κλειδί `'one'` και την τιμή "uno". Εάν εκτυπώσουμε ξανά το λεξικό, βλέπουμε ένα ζεύγος κλειδιού-τιμής με άνω και κάτω τελεία μεταξύ του κλειδιού και της τιμής:

```
>>> print(eng2sp)
{'one': 'uno'}
```

Αυτή η μορφή εξόδου είναι επίσης μια μορφή εισόδου. Για παράδειγμα, μπορείτε να δημιουργήσετε ένα νέο λεξικό με τρία στοιχεία. Αλλά αν εκτυπώσετε «eng2sp», ίσως εκπλαγείτε:

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> print(eng2sp)
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

Η σειρά των ζευγών κλειδιού-τιμής δεν είναι η ίδια. Στην πραγματικότητα, αν πληκτρολογήσετε το ίδιο παράδειγμα στον υπολογιστή σας, ενδέχεται να έχετε διαφορετικό αποτέλεσμα. Γενικά, η σειρά των στοιχείων σε ένα λεξικό είναι απρόβλεπτη.

Αλλά αυτό δεν είναι πρόβλημα γιατί τα στοιχεία ενός λεξικού δεν χρησιμοποιούν ποτέ ευρετήριο με ακέραιους δείκτες. Αντίθετα, χρησιμοποιείτε τα κλειδιά για να αναζητήσετε τις αντίστοιχες τιμές:

```
>>> print(eng2sp['two'])
'dos'
```

Το κλειδί `'two'` αντιστοιχεί πάντα στην τιμή "dos", οπότε η σειρά των στοιχείων δεν έχει σημασία.

Εάν το κλειδί δεν υπάρχει στο λεξικό, λαμβάνετε μια εξαίρεση:

```
>>> print(eng2sp['four'])
KeyError: 'four'
```

Η συνάρτηση `len` λειτουργεί σε λεξικά. Επιστρέφει το πλήθος των ζευγών κλειδιού-τιμής:

```
>>> len(eng2sp)
3
```

Ο τελεστής `in` λειτουργεί σε λεξικά. Σας λέει εάν κάτι εμφανίζεται ως *κλειδί* στο λεξικό (η εμφάνιση ως τιμή δεν αρκεί).

```
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
False
```

Για να δείτε εάν κάτι εμφανίζεται ως τιμή σε ένα λεξικό, μπορείτε να χρησιμοποιήσετε τη μέθοδο `values`, η οποία επιστρέφει τις τιμές ως έναν τύπο dict_values, που μπορεί να μετατραπεί σε λίστα και, στη συνέχεια, μπορείτε να χρησιμοποιήσετε τον τελεστή `in` (την Python 2 η μέθοδος `values` επιστρέφει μία λίστα των τιμών):

```
>>> vals = list(eng2sp.values())
>>> 'uno' in vals
True
```

Ο τελεστής `in` χρησιμοποιεί διαφορετικούς αλγόριθμους για λίστες και για λεξικά. Για λίστες, χρησιμοποιεί έναν αλγόριθμο γραμμικής αναζήτησης. Καθώς η λίστα μεγαλώνει, ο χρόνος αναζήτησης μεγαλώνει σε ευθεία αναλογία με το μήκος της λίστας. Για τα λεξικά, η Python χρησιμοποιεί έναν αλγόριθμο που ονομάζεται *πίνακας κατακερματισμού (hash table)* που έχει μια αξιοσημείωτη ιδιότητα: ο τελεστής `in` χρειάζεται περίπου τον ίδιο χρόνο ανεξάρτητα από το πόσα στοιχεία περιέχονται σε ένα λεξικό. Δεν θα εξηγήσω γιατί οι συναρτήσεις κατακερματισμού είναι τόσο μαγικές, αλλά μπορείτε να διαβάσετε περισσότερα για αυτό στο wikipedia.org/wiki/Hash_table.

Άσκηση 1: Κατεβάστε ένα αντίγραφο του αρχείου
www.gr.py4e.com/code3/words.txt

Γράψτε ένα πρόγραμμα που να διαβάζει τις λέξεις στο *words.txt* και να τις αποθηκεύει ως κλειδιά σε ένα λεξικό. Δεν έχει σημασία ποιες είναι οι τιμές. Στη συνέχεια, μπορείτε να χρησιμοποιήσετε τον τελεστή `in` ως έναν γρήγορο τρόπο για να ελέγξετε εάν μια συμβολοσειρά υπάρχει στο λεξικό.

Το λεξικό ως σύνολο μετρητών

Ας υποθέσουμε ότι σας δίνεται μια συμβολοσειρά και θέλετε να μετρήσετε πόσες φορές εμφανίζεται κάθε γράμμα. Υπάρχουν διάφοροι τρόποι για να το κάνετε:

1. Θα μπορούσατε να δημιουργήσετε 26 μεταβλητές, μία για κάθε γράμμα του αλφαβήτου. Στη συνέχεια, θα μπορούσατε να διασχίσετε τη συμβολοσειρά και, για κάθε χαρακτήρα, να αυξήσετε τον αντίστοιχο μετρητή, πιθανώς χρησιμοποιώντας μια πολλαπλή συνθήκη.

2. Θα μπορούσατε να δημιουργήσετε μια λίστα με 26 στοιχεία. Στη συνέχεια, να μετατρέψετε κάθε χαρακτήρα σε έναν αριθμό (χρησιμοποιώντας την ενσωματωμένη συνάρτηση `ord`), να χρησιμοποιήσετε τον αριθμό ως δείκτη στη λίστα και να αυξήσετε τον κατάλληλο μετρητή.

3. Θα μπορούσατε να δημιουργήσετε ένα λεξικό με χαρακτήρες ως κλειδιά και μετρητές ως τις αντίστοιχες τιμές. Την πρώτη φορά που εντοπίζετε έναν

χαρακτήρα, προσθέτετε ένα στοιχείο στο λεξικό. Στη συνέχεια θα αυξάνετε την τιμή του υπάρχοντος στοιχείου.

Κάθε μία από αυτές τις επιλογές εκτελεί τον ίδιο υπολογισμό, αλλά καθεμία από αυτές υλοποιεί τον υπολογισμό με διαφορετικό τρόπο.

Μια *υλοποίηση* είναι ένας τρόπος εκτέλεσης ενός υπολογισμού. Ορισμένες υλοποιήσεις είναι καλύτερες από άλλες. Για παράδειγμα, ένα πλεονέκτημα της εφαρμογής του λεξικού είναι ότι δεν χρειάζεται να γνωρίζουμε εκ των προτέρων ποια γράμματα εμφανίζονται στη συμβολοσειρά και δημιουργούμε θέσεις μόνο για τα γράμματα που εμφανίζονται.

Εδώ είναι μια πιθανή εκδοχή του κώδικα:

```
word = 'brontosaurus'
d = dict()
for c in word:
    if c not in d:
        d[c] = 1
    else:
        d[c] = d[c] + 1
print(d)
```

Υπολογίζουμε ουσιαστικά ένα *ιστόγραμμα*, το οποίο είναι ένας στατιστικός όρος για ένα σύνολο μετρητών (ή συχνοτήτων).

Ο βρόχος `for` διασχίζει τη συμβολοσειρά. Κάθε φορά μέσω του βρόχου, εάν ο χαρακτήρας `c` δεν υπάρχει στο λεξικό, δημιουργούμε ένα νέο στοιχείο με το κλειδί `c` και την αρχική τιμή 1 (αφού έχουμε δει αυτό το γράμμα μία φορά). Εάν το `c` είναι ήδη στο λεξικό, αυξάνουμε το `d[c]`.

Ακολουθεί η έξοδος του προγράμματος:

```
{'a': 1, 'b': 1, 'o': 2, 'n': 1, 's': 2, 'r': 2, 'u': 2, 't': 1}
```

Το ιστόγραμμα δείχνει ότι τα γράμματα "a" και "b" εμφανίζονται από μία φορά. Το "o" εμφανίζεται δύο φορές και ούτω καθεξής.

Τα λεξικά έχουν μια μέθοδο που ονομάζεται `get`, που παίρνει ένα κλειδί και μια προεπιλεγμένη τιμή. Εάν το κλειδί εμφανίζεται στο λεξικό, το `get` επιστρέφει την αντίστοιχη τιμή, διαφορετικά επιστρέφει την προεπιλεγμένη τιμή. Για παράδειγμα:

```
>>> counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
```

```
>>> print(counts.get('jan', 0))
100
>>> print(counts.get('tim', 0))
0
```

Μπορούμε να χρησιμοποιήσουμε την get για να γράψουμε πιο συνοπτικά τον βρόχο ιστογράμματος. Επειδή η μέθοδος get χειρίζεται αυτόματα την περίπτωση που ένα κλειδί δεν περιέχεται στο λεξικό, μπορούμε να αντικαταστήσουμε τέσσερις γραμμές με μία και να εξαλείψουμε την εντολή `if`.

```
word = 'brontosaurus'
d = dict()
for c in word:
    d[c] = d.get(c,0) + 1
print(d)
```

Η χρήση της μεθόδου get, για την απλοποίηση αυτού του βρόχου μέτρησης καταλήγει σε ένα πολύ συχνά χρησιμοποιούμενο "ιδίωμα" στην Python που θα το χρησιμοποιήσουμε πολλές φορές στο υπόλοιπο βιβλίο. Θα πρέπει λοιπόν να αφιερώσετε λίγο χρόνο και να συγκρίνετε τον βρόχο που χρησιμοποιεί την εντολή `if` και τον τελεστή `in` με τον βρόχο που χρησιμοποιεί τη μέθοδο get. Κάνουν ακριβώς το ίδιο πράγμα, αλλά ο δεύτερος είναι συνοπτικότερος.

Λεξικά και αρχεία

Μία από τις συνηθισμένες χρήσεις ενός λεξικού είναι να μετράει την εμφάνιση λέξεων σε ένα αρχείο, που περιέχει κείμενο. Ας ξεκινήσουμε με ένα πολύ απλό αρχείο λέξεων, βγαλμένων από το κείμενο του *Ρωμαίος και Ιουλιέτα (Romeo and Juliet)*.

Για το πρώτο σύνολο παραδειγμάτων, θα χρησιμοποιήσουμε μια σύντομη και απλοποιημένη έκδοση του κειμένου, χωρίς σημεία στίξης. Αργότερα θα δουλέψουμε με το κείμενο της σκηνής με σημεία στίξης.

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Θα γράψουμε ένα πρόγραμμα Python για να διαβάσουμε τις γραμμές του αρχείου, θα διασπάσουμε κάθε γραμμή σε μια λίστα λέξεων και, στη συνέχεια, με βρόχο θα

διατρέξουμε κάθε μία από τις λέξεις της γραμμής και θα μετρήσουμε κάθε λέξη χρησιμοποιώντας ένα λεξικό.

Θα δείτε ότι έχουμε δύο βρόχους for. Ο εξωτερικός βρόχος διαβάζει τις γραμμές του αρχείου και ο εσωτερικός βρόχος διατρέχει κάθε μία από τις λέξεις στη συγκεκριμένη γραμμή. Αυτό είναι ένα παράδειγμα ενός μοτίβου που ονομάζεται *εμφωλευμένοι βρόχοι* επειδή ένας από τους βρόχους είναι ο *εξωτερικός* βρόχος και ο άλλος βρόχος είναι ο *εσωτερικός*.

Επειδή ο εσωτερικός βρόχος εκτελεί όλες τις επαναλήψεις του κάθε φορά που ο εξωτερικός βρόχος κάνει μία επανάληψη, θεωρούμε ότι ο εσωτερικός βρόχος επαναλαμβάνεται "πιο γρήγορα" και ο εξωτερικός βρόχος επαναλαμβάνεται πιο αργά.

Ο συνδυασμός των δύο εμφωλευμένων βρόχων διασφαλίζει ότι θα μετράμε κάθε λέξη, σε κάθε γραμμή του αρχείου εισόδου.

```
fname = input('Εισαγάγετε το όνομα του αρχείου: ')
try:
    fhand = open(fname)
except:
    print('Δεν είναι δυνατό το άνοιγμα του αρχείου:', fname)
    exit()

πλήθη = dict()
for γραμμή in fhand:
    λέξεις = γραμμή.split()
    for λέξη in λέξεις:
        if λέξη not in πλήθη:
            πλήθη[λέξη] = 1
        else:
            πλήθη[λέξη] += 1

print(πλήθη)
```

#Code: http://www.gr.py4e.com/code3/count1.py

Στη δήλωση else, χρησιμοποιούμε την πιο συμπαγή εναλλακτική για την αύξηση μιας μεταβλητής. Το counts[word] += 1 ισοδυναμεί με counts[word] = counts[word] + 1. Και οι δύο τρόποι μπορούν να χρησιμοποιηθούν για την αλλαγή της τιμής μιας μεταβλητής κατά οποιοδήποτε, επιθυμητό, ποσό. Παρόμοιες εναλλακτικές υπάρχουν για τα -=, *= και /=.

Όταν εκτελούμε το πρόγραμμα, βλέπουμε μια ακατέργαστη έξοδο όλων των μετρήσεων, σε μη ταξινομημένη σειρά. (Το αρχείο *romeo.txt* είναι διαθέσιμο στη διεύθυνση www.py4e.com/code3/romeo.txt)

```
python count1.py
Εισαγάγετε το όνομα του αρχείου: romeo.txt
{'and': 3, 'envious': 1, 'already': 1, 'fair': 1,
'is': 3, 'through': 1, 'pale': 1, 'yonder': 1,
'what': 1, 'sun': 2, 'Who': 1, 'But': 1, 'moon': 1,
'window': 1, 'sick': 1, 'east': 1, 'breaks': 1,
'grief': 1, 'with': 1, 'light': 1, 'It': 1, 'Arise': 1,
'kill': 1, 'the': 3, 'soft': 1, 'Juliet': 1}
```

Είναι λίγο άβολο να ψάξουμε μέσα στο λεξικό για να βρούμε τις πιο συχνά επαναλαμβανόμενες λέξεις και τον πλήθος των εμφανίσεών τους, γι' αυτό πρέπει να προσθέσουμε λίγο ακόμα κώδικα Python για να έχουμε μια έξοδο που θα είναι πιο εύχρηστη.

Βρόχοι και λεξικά

Εάν χρησιμοποιείτε ένα λεξικό ως ακολουθία μιας εντολής `for`, τότε αυτή διασχίζει τα κλειδιά του λεξικού. Για παράδειγμα, αυτός ο βρόχος εκτυπώνει κάθε κλειδί και την αντίστοιχη τιμή:

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for key in counts:
    print(key, counts[key])
```

Δείτε τί παράγεται στην έξοδο:

```
jan 100
chuck 1
annie 42
```

Και πάλι, τα κλειδιά δεν είναι ταξινομημένα.

Μπορούμε να χρησιμοποιήσουμε αυτό το μοτίβο για να εφαρμόσουμε τα διάφορα ιδιώματα βρόχου που περιγράψαμε νωρίτερα. Για παράδειγμα, αν θέλαμε να βρούμε όλες τις εγγραφές σε ένα λεξικό με τιμή πάνω από δέκα, θα μπορούσαμε να γράψουμε τον ακόλουθο κώδικα:

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
```

```
for key in counts:
    if counts[key] > 10 :
        print(key, counts[key])
```

Ο βρόχος for επαναλαμβάνεται μέσω των *κλειδιών* του λεξικού, επομένως πρέπει να χρησιμοποιήσουμε τον τελεστή ευρετηρίου για να ανακτήσουμε την αντίστοιχη *τιμή* για κάθε κλειδί. Δείτε πώς φαίνεται η έξοδος:

```
jan 100
annie 42
```

Βλέπουμε μόνο τις εγγραφές με τιμή πάνω από 10.

Εάν θέλετε να εκτυπώσετε τα κλειδιά με αλφαβητική σειρά, πρέπει να κάνετε πρώτα μια λίστα με τα κλειδιά του λεξικού, χρησιμοποιώντας τη μέθοδο keys, που είναι διαθέσιμη για αντικείμενα λεξικού και, στη συνέχεια, να ταξινομήστε τη λίστα και να τη διασχίσετε, αναζητώντας κάθε κλειδί και να εκτυπώσετε ζεύγη κλειδιών-τιμών ταξινομημένα, ως εξής:

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
lst = list(counts.keys())
print(lst)
lst.sort()
for key in lst:
    print(key, counts[key])
```

Δείτε πώς φαίνεται η έξοδος:

```
['jan', 'chuck', 'annie']
annie 42
chuck 1
jan 100
```

Πρώτα βλέπετε τη λίστα των κλειδιών σε μη ταξινομημένη σειρά, όπως τη λαμβάνουμε από τη μέθοδο keys. Στη συνέχεια, βλέπουμε τα ζεύγη κλειδιού-τιμής ταξινομημένα από τον βρόχο for.

Προχωρημένη ανάλυση κειμένου

Στο παραπάνω παράδειγμα, χρησιμοποιώντας το αρχείο *romeo.txt*, απλοποιήσαμε το αρχείο, αφαιρώντας όλα τα σημεία στίξης με το χέρι. Το πραγματικό κείμενο έχει πολλά σημεία στίξης, όπως φαίνεται παρακάτω.

```
But, soft! what light through yonder window breaks?
It is the east, and Juliet is the sun.
Arise, fair sun, and kill the envious moon,
Who is already sick and pale with grief,
```

Μιας και η συνάρτηση `split`, της Python, αναζητά κενά και αντιμετωπίζει τις λέξεις ως διακριτικά που χωρίζονται με κενά, θα αντιμετωπίζαμε τις λέξεις "soft!" και "soft" ως *διαφορετικές* λέξεις και θα δημιουργούνταν μια ξεχωριστή καταχώρηση λεξικού για κάθε μια από αυτές τις λέξεις.

Επίσης, δεδομένου ότι το αρχείο έχει κεφαλαία και πεζά γράμματα, θα αντιμετωπίζαμε το "who" και το "Who" ως διαφορετικές λέξεις, με διαφορετικές μετρήσεις.

Μπορούμε να λύσουμε και τα δύο αυτά προβλήματα χρησιμοποιώντας τις μεθόδους συμβολοσειράς `lower`, `punctuation` και `translate`. Η `translate` είναι η πιο λεπτή από τις μεθόδους. Ακολουθεί η τεκμηρίωση για την `translate`:

`line.translate(str.maketrans(fromstr, tostr, deletestr))`

Αντικαταστήστε τους χαρακτήρες που περιλαμβάνονται στο `fromstr` με τον αντίστοιχο χαρακτήρα του `tostr` και διαγράψτε όλους τους χαρακτήρες που βρίσκονται στο `deletestr`. Το `fromstr` και το `tostr` μπορεί να είναι κενές συμβολοσειρές και η παράμετρος `deletestr` μπορεί να παραλειφθεί.

Δεν θα καθορίσουμε το `tostr`, αλλά θα χρησιμοποιήσουμε την παράμετρο `deletestr` για να διαγράψουμε όλα τα σημεία στίξης. Ακόμη, θα αφήσουμε την Python να μας πει τη λίστα των χαρακτήρων, που θεωρεί "σημεία στίξης":

```
>>> import string
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

Οι παράμετροι της `translate` ήταν διαφορετικές στην Python 2.0.

Κάνουμε τις ακόλουθες τροποποιήσεις στο πρόγραμμά μας:

```
import string
fname = input('Εισαγάγετε το όνομα του αρχείου: ')
try:
    fhand = open(fname)
except:
    print('Δεν είναι δυνατό το άνοιγμα του αρχείου:', fname)
    exit()
```

```
πλήθη = dict()
for γραμμή in fhand:
    γραμμή = γραμμή.rstrip()
    γραμμή = γραμμή.translate(γραμμή.maketrans('', '', string.punctuation))
    γραμμή = γραμμή.lower()
    λέξεις = γραμμή.split()
    for λέξη in λέξεις:
        if λέξη not in πλήθη:
            πλήθη[λέξη] = 1
        else:
            πλήθη[λέξη] += 1
print(πλήθη)
```

#Code: http://www.gr.py4e.com/code3/count2.py

Μέρος της εκμάθησης της "Τέχνης της Python" ή της "Python-ικής Σκέψης" είναι η συνειδητοποίηση ότι η Python έχει συχνά ενσωματωμένες δυνατότητες για πολλά κοινά προβλήματα ανάλυσης δεδομένων. Με τον καιρό, θα δείτε αρκετά παραδείγματα κώδικα και θα διαβάσετε αρκετά την τεκμηρίωση, για να ξέρετε πού να ψάξετε, προκειμένου να ελέγξετε εάν κάποιος έχει ήδη γράψει κάτι που κάνει τη δουλειά σας πολύ πιο εύκολη.

Η παρακάτω είναι μια συντομευμένη έκδοση της εξόδου:

```
Εισαγάγετε το όνομα του αρχείου: romeo-full.txt
{'swearst': 1, 'all': 6, 'afeard': 1, 'leave': 2, 'these': 2,
'kinsmen': 2, 'what': 11, 'thinkst': 1, 'love': 24, 'cloak': 1,
a': 24, 'orchard': 2, 'light': 5, 'lovers': 2, 'romeo': 40,
'maiden': 1, 'whiteupturned': 1, 'juliet': 32, 'gentleman': 1,
'it': 22, 'leans': 1, 'canst': 1, 'having': 1, ...}
```

Η αναζήτηση σε αυτήν της εξόδου εξακολουθεί να είναι δυσκίνητη. Μπορούμε όμως να χρησιμοποιήσουμε την Python για να μας δώσει ακριβώς αυτό που ψάχνουμε, αλλά για να το κάνουμε αυτό, πρέπει να μάθουμε για τις *πλειάδες* της Python. Θα συνεχίσουμε αυτό το παράδειγμα μόλις μάθουμε για τις πλειάδες.

Εκσφαλμάτωση

Καθώς εργάζεστε με μεγαλύτερα σύνολα δεδομένων, ο εντοπισμός σφαλμάτων μπορεί να γίνει δύσκολος με την εκτύπωση και τον έλεγχο των δεδομένων με το χειροκίνητα.

Ακολουθούν ορισμένες προτάσεις για τον εντοπισμό σφαλμάτων σε μεγάλα σύνολα δεδομένων:

Μειώστε την είσοδο : Εάν είναι δυνατόν, μειώστε το μέγεθος του συνόλου δεδομένων. Για παράδειγμα, εάν το πρόγραμμα διαβάζει ένα αρχείο κειμένου, ξεκινήστε μόνο με τις πρώτες 10 γραμμές ή με ένα μικρότερο παράδειγμα, που μπορεί να βρείτε. Μπορείτε είτε να επεξεργαστείτε τα ίδια τα αρχεία, είτε (καλύτερα) να τροποποιήσετε το πρόγραμμα ώστε να διαβάζει μόνο τις πρώτες n γραμμές.

```
Εάν υπάρχει σφάλμα, μπορείτε να μειώσετε το `n` στη μικρότερη τιμή που
εμφανίζει το σφάλμα και, στη συνέχεια, να το αυξήσετε σταδιακά, καθώς
βρίσκετε και διορθώνετε τα σφάλματα.
```

Ελέγξτε τις περιλήψεις και τους τύπους : Αντί να εκτυπώσετε και να ελέγξετε ολόκληρο το σύνολο δεδομένων, σκεφτείτε να εκτυπώσετε περιλήψεις των δεδομένων: για παράδειγμα, τον αριθμό των στοιχείων σε ένα λεξικό ή το σύνολο μιας λίστας αριθμών.

```
Μια κοινή αιτία σφαλμάτων χρόνου εκτέλεσης είναι μια τιμή που δεν είναι
του σωστού τύπου. Για τον εντοπισμό σφαλμάτων αυτού του είδους, συχνά
αρκεί να εκτυπώσετε τον τύπο μιας τιμής.
```

Γράψτε αυτοελέγχους : Μερικές φορές μπορείτε να γράψετε κώδικα για να ελέγξετε αυτόματα για σφάλματα. Για παράδειγμα, εάν υπολογίζετε τον μέσο όρο μιας λίστας αριθμών, μπορείτε να ελέγξετε ότι το αποτέλεσμα δεν είναι μεγαλύτερο από το μέγιστο στοιχείο στη λίστα ή μικρότερο από το ελάχιστο. Αυτό ονομάζεται "έλεγχος λογικής" (sanity check), επειδή εντοπίζει αποτελέσματα που είναι "παράλογα".

```
Ένα άλλο είδος ελέγχου συγκρίνει τα αποτελέσματα δύο διαφορετικών
υπολογισμών για να δει αν είναι συνεπή. Αυτό ονομάζεται «έλεγχος
συνέπειας».
```

Εκτυπώστε όμορφα το αποτέλεσμα (pprint) : Η μορφοποίηση της εξόδου εντοπισμού σφαλμάτων μπορεί να διευκολύνει τον εντοπισμό ενός σφάλματος.

Και πάλι, ο χρόνος που αφιερώνετε στην μελέτη της κατασκευής του σκελετού του προγράμματος μπορεί να μειώσει τον χρόνο που αφιερώνετε στην αποσφαλμάτωση.

Γλωσσάριο

αναζήτηση : Μια λειτουργία λεξικού που παίρνει ένα κλειδί και βρίσκει την αντίστοιχη τιμή.

εμφωλευμένοι βρόχοι : Όταν υπάρχουν ένας ή περισσότεροι βρόχοι ο ένας "μέσα" στον άλλο βρόχο. Ο εσωτερικός βρόχος εκτελείτε μέχρι να ολοκληρωθεί σε κάθε εκτέλεση του εξωτερικού βρόχου.

ζεύγος κλειδιού-τιμής : Η αναπαράσταση της αντιστοίχισης ενός κλειδιού σε μια τιμή.

ιστόγραμμα : Ένα σύνολο μετρητών.

κλειδί : Ένα αντικείμενο που εμφανίζεται σε ένα λεξικό ως το πρώτο μέρος ενός ζεύγους κλειδιού-τιμής.

λεξικό : Μια αντιστοίχιση από ένα σύνολο κλειδιών στις αντίστοιχες τιμές τους.

πίνακας κατακερματισμού - hashtable : Ο αλγόριθμος που χρησιμοποιείται για την υλοποίηση λεξικών στην Python.

στοιχείο : Ένα άλλο όνομα για ένα ζεύγος κλειδιού-τιμής.

συνάρτηση κατακερματισμού - hash function : Μια συνάρτηση που χρησιμοποιείται από έναν πίνακα κατακερματισμού για τον υπολογισμό της θέσης ενός κλειδιού.

τιμή : Ένα αντικείμενο που εμφανίζεται σε ένα λεξικό ως το δεύτερο μέρος ενός ζεύγους κλειδιού-τιμής. Αυτό είναι πιο συγκεκριμένο από την προηγούμενη χρήση της λέξης "τιμή".

υλοποίηση : Ένας τρόπος εκτέλεσης ενός υπολογισμού.

Ασκήσεις

Άσκηση 2: Γράψτε ένα πρόγραμμα που ταξινομεί κάθε μήνυμα αλληλογραφίας με βάση την ημέρα της εβδομάδας που ολοκληρώθηκε η παράδοση. Για να το κάνετε αυτό, αναζητήστε γραμμές που ξεκινούν με "From", στη συνέχεια αναζητήστε την τρίτη λέξη και καταμετρήστε την κάθε μία από τις ημέρες της εβδομάδας. Στο τέλος του προγράμματος εκτυπώστε τα περιεχόμενα του λεξικού σας (η σειρά δεν έχει σημασία).

Δείγμα Γραμμής:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

Δείγμα Εκτέλεσης:

```
python dow.py
```

```
Εισαγάγετε το όνομα του αρχείου: mbox-short.txt
{'Fri': 20, 'Thu': 6, 'Sat': 1}
```

Άσκηση 3: Γράψτε ένα πρόγραμμα για να διαβάσετε ένα αρχείο καταγραφής αλληλογραφίας, δημιουργήστε ένα ιστόγραμμα χρησιμοποιώντας ένα λεξικό για να μετρήσετε πόσα μηνύματα έχουν προέλθει από κάθε διεύθυνση email και εκτυπώστε το λεξικό.

```
Εισαγάγετε το όνομα του αρχείου: mbox-short.txt
{'gopal.ramasammycook@gmail.com': 1, 'louis@media.berkeley.edu': 3,
'cwen@iupui.edu': 5, 'antranig@caret.cam.ac.uk': 1,
'rjlowe@iupui.edu': 2, 'gsilver@umich.edu': 3,
'david.horwitz@uct.ac.za': 4, 'wagnermr@iupui.edu': 1,
'zqian@umich.edu': 4, 'stephen.marquard@uct.ac.za': 2,
'ray@media.berkeley.edu': 1}
```

Άσκηση 4: Προσθέστε κώδικα στο προηγούμενο πρόγραμμα για να καταλάβετε ποιος έχει τα περισσότερα μηνύματα στο αρχείο. Αφού διαβάσετε όλα τα δεδομένα και το δημιουργήσετε το λεξικό, κοιτάξτε στο λεξικό χρησιμοποιώντας έναν βρόχο μέγιστου (βλ. Κεφάλαιο 5: Βρόχος μέγιστου και ελάχιστου), για να βρείτε ποιος έχει τα περισσότερα μηνύματα και να εκτυπώσετε πόσα μηνύματα έχει το άτομο αυτό.

```
Εισαγάγετε το όνομα του αρχείου: mbox-short.txt
cwen@iupui.edu 5

Εισαγάγετε το όνομα του αρχείου: mbox.txt
zqian@umich.edu 195
```

Άσκηση 5: Αυτό το πρόγραμμα καταγράφει το όνομα τομέα (αντί για τη διεύθυνση) από όπου στάλθηκε το μήνυμα, αντί από ποιον προήλθε το μήνυμα (δηλαδή ολόκληρη η διεύθυνση email) και μετρά το πλήθος των μηνυμάτων από τον τομέα αυτό. Στο τέλος του προγράμματος, εκτυπώστε τα περιεχόμενα του λεξικού σας.

```
python schoolcount.py
Εισαγάγετε το όνομα του αρχείου: mbox-short.txt
{'media.berkeley.edu': 4, 'uct.ac.za': 6, 'umich.edu': 7,
'gmail.com': 1, 'caret.cam.ac.uk': 1, 'iupui.edu': 8}
```

Κεφάλαιο 10

Πλειάδες

Οι πλειάδες είναι αμετάβλητες

Μια πλειάδα[1] (tuple) είναι μια ακολουθία τιμών που μοιάζει πολύ με μια λίστα. Οι τιμές που είναι αποθηκευμένες σε μια πλειάδα μπορούν να είναι οποιουδήποτε τύπου και έχουν δείκτες ακέραιους αριθμούς. Η σημαντική διαφορά είναι ότι οι πλειάδες είναι *αμετάβλητες*. Οι πλειάδες είναι επίσης *συγκρίσιμες* και *κατακερματισμένες (hashable)*, ώστε να μπορούμε να ταξινομήσουμε λίστες τους και να χρησιμοποιήσουμε πλειάδες ως κλειδιά - τιμές σε λεξικά της Python.

Συντακτικά, μια πλειάδα είναι μια λίστα τιμών διαχωρισμένη με κόμματα:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Αν και δεν είναι απαραίτητο, είναι σύνηθες να περικλείουμε τις πλειάδες σε παρενθέσεις, για να βοηθηθούμε στο να αναγνωρίσουμε γρήγορα τις πλειάδες, όταν κοιτάμε τον κώδικα Python:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

Για να δημιουργήσετε μια πλειάδα με ένα μόνο στοιχείο, πρέπει να συμπεριλάβετε το τελικό κόμμα:

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

Χωρίς κόμμα, η Python αντιμετωπίζει το `('a')` ως έκφραση με μια συμβολοσειρά σε παρένθεση, που αποτιμάται σε μια συμβολοσειρά:

```
>>> t2 = ('a')
```

---

[1] Ενδιαφέρουσα πληροφορία: Η λέξη "tuple (πλειάδα)" προέρχεται από τα ονόματα που δίνονται σε ακολουθίες αριθμών διαφορετικού μήκους: μονή, διπλή, τριπλή, τετραπλή, πενταπλή (quintuple), εξάπλη (sextuple), επταπλή (septuple), κ.λπ.

```
>>> type(t2)
<type 'str'>
```

Ένας άλλος τρόπος κατασκευής πλειάδας είναι η ενσωματωμένη συνάρτηση `tuple`. Χωρίς όρισμα, δημιουργεί μια κενή πλειάδα:

```
>>> t = tuple()
>>> print(t)
()
```

Εάν το όρισμα είναι μια ακολουθία (συμβολοσειρά, λίστα ή πλειάδα), το αποτέλεσμα της κλήσης της `tuple` είναι μια πλειάδα με τα στοιχεία της ακολουθίας:

```
>>> t = tuple('lupins')
>>> print(t)
('l', 'u', 'p', 'i', 'n', 's')
```

Επειδή το `tuple` είναι το όνομα ενός κατασκευαστή, θα πρέπει να αποφύγετε τη χρήση του ως όνομα μεταβλητής.

Οι περισσότεροι τελεστές λιστών λειτουργούν και σε πλειάδες. Ο τελεστής αγκύλη ευρετηριάζει ένα στοιχείο:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print(t[0])
'a'
```

Και ο τελεστής διαμέρισης επιλέγει μια σειρά στοιχείων.

```
>>> print(t[1:3])
('b', 'c')
```

Αλλά αν προσπαθήσετε να τροποποιήσετε ένα από τα στοιχεία της πλειάδας, λαμβάνετε ένα σφάλμα:

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

Δεν μπορείτε να τροποποιήσετε τα στοιχεία μιας πλειάδας, αλλά μπορείτε να αντικαταστήσετε μια πλειάδα με μια άλλη:

```
>>> t = ('A',) + t[1:]
>>> print(t)
('A', 'b', 'c', 'd', 'e')
```

Σύγκριση πλειάδων

Οι τελεστές σύγκρισης λειτουργούν με πλειάδες και άλλες ακολουθίες. Η Python ξεκινά συγκρίνοντας το πρώτο στοιχείο από κάθε ακολουθία. Αν είναι ίσα, πηγαίνει στο επόμενο στοιχείο και ούτω καθεξής, μέχρι να βρει στοιχεία που διαφέρουν. Τα επόμενα στοιχεία δεν λαμβάνονται υπόψη (ακόμα και αν είναι πραγματικά μεγάλα).

```
>>> (0, 1, 2) < (0, 3, 4)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
```

Η μέθοδος sort λειτουργεί με τον ίδιο τρόπο. Ταξινομεί κυρίως κατά το πρώτο στοιχείο, αλλά στην περίπτωση ισοπαλίας, ταξινομεί κατά το δεύτερο στοιχείο και ούτω καθεξής.

Αυτό το χαρακτηριστικό προσφέρεται σε ένα μοτίβο που ονομάζεται *DSU*

Decorate - Διακοσμώ : μια ακολουθία δημιουργώντας μια λίστα πλειάδων με ένα ή περισσότερα κλειδιά ταξινόμησης που προηγούνται των στοιχείων των πλειάδων,

Sort - Ταξινομώ : τη λίστα των πλειάδων, χρησιμοποιώντας την ενσωματωμένη sort της Python και

Undecorate - Αφαιρώ τη διακόσμηση : εξάγοντας τα ταξινομημένα στοιχεία της ακολουθίας.

Για παράδειγμα, ας υποθέσουμε ότι έχετε μια λίστα λέξεων και θέλετε να τις ταξινομήσετε από αυτή με το μεγαλύτερο μήκος προς στη συντομότερη:

```
κείμενο = 'but soft what light in yonder window breaks'
λέξεις = κείμενο.split()
t = list()
for λέξη in λέξεις:
    t.append((len(λέξη), λέξη))

t.sort(reverse=True)

res = list()
for μήκος, λέξη in t:
    res.append(λέξη)

print(res)
```

#Code: http://www.gr.py4e.com/code3/soft.py

Ο πρώτος βρόχος δημιουργεί μια λίστα με πλειάδες, όπου κάθε πλειάδα είναι μια λέξη με προπορευόμενο το μήκος της.

Η `sort` συγκρίνει τα πρώτα στοιχεία των πλειάδων, το μήκος. Το όρισμα δεμευμένης λέξης `reverse=True` λέει στην `sort` να λειτουργήσει με φθίνουσα σειρά.

Ο δεύτερος βρόχος διασχίζει τη λίστα των πλειάδων και δημιουργεί μια λίστα λέξεων με φθίνουσα σειρά μήκους. Οι λέξεις των τεσσάρων χαρακτήρων ταξινομούνται με *αντίστροφη* αλφαβητική σειρά, επομένως το "what" εμφανίζεται πριν από το "soft" στην παρακάτω λίστα.

Η έξοδος του προγράμματος είναι η εξής:

```
['yonder', 'window', 'breaks', 'light', 'what', 'soft', 'but', 'in']
```

Φυσικά η γραμμή χάνει μεγάλο μέρος του ποιητικού της αντίκτυπου όταν μετατραπεί σε λίστα Python και ταξινομηθεί σε φθίνουσα σειρά μήκους λέξεων.

Εκχώρηση τιμής σε πλειάδα

Ένα από τα μοναδικά συντακτικά χαρακτηριστικά της γλώσσας Python είναι η δυνατότητα να υπάρχει πλειάδα στην αριστερή πλευρά μιας δήλωσης ανάθεσης και μια ακολουθία τιμών στη δεξιά πλευρά. Αυτό σας επιτρέπει να αναθέσετε τιμή σε περισσότερες από μία μεταβλητές τη φορά, χρησιμοποιώντας τη δοθείσα ακολουθία.

Σε αυτό το παράδειγμα έχουμε μια λίστα δύο στοιχείων (η οποία είναι μια ακολουθία) και εκχωρούμε το πρώτο και το δεύτερο στοιχείο της ακολουθίας στις μεταβλητές `x` και `y` σε μία μόνο πρόταση.

```
>>> m = [ 'have', 'fun' ]
>>> x, y = m
>>> x
'have'
>>> y
'fun'
>>>
```

Δεν είναι μαγικό, η Python *χονδρικά* μεταφράζει τη σύνταξη της πολλαπλής

ανάθεσης ως εξής[1]:

```
>>> m = [ 'have', 'fun' ]
>>> x = m[0]
>>> y = m[1]
>>> x
'have'
>>> y
'fun'
>>>
```

Στυλιστικά, όταν χρησιμοποιούμε πλειάδα στο αριστερό μέλος της εντολής εκχώρησης, παραλείπουμε τις παρενθέσεις, αλλά το ακόλουθο αποτελεί μια εξίσου έγκυρη σύνταξη:

```
>>> m = [ 'have', 'fun' ]
>>> (x, y) = m
>>> x
'have'
>>> y
'fun'
>>>
```

Μια ιδιαίτερα έξυπνη εφαρμογή της εκχώρησης σε πλειάδα μας επιτρέπει να *αντιμεταθέσουμε* τις τιμές δύο μεταβλητών σε μια μόνο πρόταση:

```
>>> a, b = b, a
```

Και στα δύο μέλη αυτής της εντολής έχουμε πλειάδες, αλλά στο αριστερό μέλος είναι μια πλειάδα μεταβλητών. Στο δεξί μέλος έχουμε μια πλειάδα εκφράσεων. Κάθε τιμή στο δεξί μέλος εκχωρείται στην αντίστοιχη μεταβλητή του αριστερού μέλους. Όλες οι εκφράσεις στη δεξιά πλευρά αξιολογούνται πριν από οποιαδήποτε από τις εκχωρήσεις.

Ο αριθμός των μεταβλητών στα αριστερά και ο αριθμός των τιμών στα δεξιά πρέπει να είναι ο ίδιος:

```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

[1] Η Python δεν μεταφράζει τη σύνταξη κυριολεκτικά. Για παράδειγμα, εάν το δοκιμάσετε με ένα λεξικό, δεν θα λειτουργήσει όπως θα περιμένατε.

Γενικότερα, στο δεξί μέλος μπορεί να υπάρχει κάθε είδους ακολουθία (συμβολοσειρά, λίστα ή πλειάδα). Για παράδειγμα, για να χωρίσετε μια διεύθυνση email σε όνομα χρήστη και τομέα, θα μπορούσατε να γράψετε:

```
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
```

Η επιστρεφόμενη τιμή από το `split` είναι μια λίστα με δύο στοιχεία. Το πρώτο στοιχείο εκχωρείται στο `uname`, το δεύτερο στο `domain`.

```
>>> print(uname)
monty
>>> print(domain)
python.org
```

Λεξικά και πλειάδες

Τα λεξικά έχουν μια μέθοδο που ονομάζεται `items`, που επιστρέφει μια λίστα πλειάδων, όπου κάθε πλειάδα είναι ένα ζεύγος κλειδιού-τιμής:

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = list(d.items())
>>> print(t)
[('b', 1), ('a', 10), ('c', 22)]
```

Όπως θα έπρεπε να περιμένετε από ένα λεξικό, τα στοιχεία δεν είναι σε σειρά.

Ωστόσο, δεδομένου ότι η λίστα των πλειάδων είναι μια λίστα και οι πλειάδες είναι συγκρίσιμες, μπορούμε τώρα να ταξινομήσουμε τη λίστα των πλειάδων. Η μετατροπή ενός λεξικού σε λίστα πλειάδων είναι ένας τρόπος για να εξάγουμε τα περιεχόμενα ενός λεξικού ταξινομημένα κατά κλειδί:

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = list(d.items())
>>> t
[('b', 1), ('a', 10), ('c', 22)]
>>> t.sort()
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

Η νέα λίστα ταξινομείται με αύξουσα αλφαβητική σειρά με βάση την τιμή κλειδιού.

Πολλαπλές εκχωρήσεις με λεξικά

Συνδυάζοντας `items`, εκχώρηση σε πλειάδα και `for`, μπορείτε να δημιουργήσετε ένα ωραίο μοτίβο κώδικα για τη προσπέλαση των κλειδιών και των τιμών ενός λεξικού με έναν μόνο βρόχο:

```
for κλειδί, τιμή in list(d.items()):
    print(τιμή, κλειδί)
```

Αυτός ο βρόχος έχει δύο *μεταβλητές επανάληψης* επειδή το `items` επιστρέφει μια λίστα πλειάδων και το `key, val` είναι μια ανάθεση πλειάδας, που επαναλαμβάνεται διαδοχικά μέσω καθενός από τα ζεύγη κλειδιού-τιμής στο λεξικό.

Για κάθε επανάληψη του βρόχου, τόσο στο `key` όσο και στη `value` ανατίθεται το επόμενο ζεύγος κλειδιού-τιμής του λεξικό (με τη σειρά κατακερματισμού).

Η έξοδος αυτού του βρόχου είναι:

```
10 a
22 c
1 b
```

Και πάλι, είναι σε σειρά κατακερματισμού κλειδιών (δηλαδή, χωρίς συγκεκριμένη σειρά).

Εάν συνδυάσουμε αυτές τις δύο τεχνικές, μπορούμε να εκτυπώσουμε τα περιεχόμενα ενός λεξικού ταξινομημένα με βάση την *τιμή* που είναι αποθηκευμένη σε κάθε ζεύγος κλειδιού-τιμής.

Για να γίνει αυτό, φτιάχνουμε πρώτα μια λίστα με πλειάδες, όπου κάθε πλειάδα είναι `(value, key)`. Η μέθοδος `items` θα μας έδινε μια λίστα με πλειάδες `(key, value)`, αλλά αυτή τη φορά θέλουμε να ταξινομήσουμε κατά τιμή και όχι κατά κλειδί. Αφού δημιουργήσουμε τη λίστα με τις πλειάδες της τιμής - κλειδιού, είναι απλό να ταξινομήσουμε τη λίστα με αντίστροφη σειρά και να εκτυπώσουμε τη νέα, ταξινομημένη λίστα.

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> l = list()
>>> for key, val in d.items() :
...     l.append( (val, key) )
...
>>> l
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> l.sort(reverse=True)
```

```
>>> l
[(22, 'c'), (10, 'a'), (1, 'b')]
>>>
```

Κατασκευάζοντας προσεκτικά τη λίστα των πλειάδων ώστε να έχει την τιμή ως το πρώτο στοιχείο κάθε πλειάδας, μπορούμε να ταξινομήσουμε τη λίστα των πλειάδων και να πάρουμε τα περιεχόμενα του λεξικού μας ταξινομημένα κατά τιμή.

Οι πιο συνηθισμένες λέξεις

Επιστρέφοντας στο προηγούμενο παράδειγμα του κειμένου από το *Ρωμαίος και Ιουλιέτα* Πράξη 2, Σκηνή 2, μπορούμε να επεκτείνουμε το πρόγραμμά μας για να χρησιμοποιήσουμε αυτήν την τεχνική για να εκτυπώσουμε τις δέκα πιο συνηθισμένες λέξεις στο κείμενο ως εξής:

```
import string
fhand = open('romeo-full.txt')
πλήθη = dict()
for γραμμή in fhand:
    γραμμή = γραμμή.translate(str.maketrans('', '', string.punctuation))
    γραμμή = γραμμή.lower()
    λέξεις = γραμμή.split()
    for λέξη in λέξεις:
        if λέξη not in πλήθη:
            πλήθη[λέξη] = 1
        else:
            πλήθη[λέξη] += 1

# Sort the dictionary by value
λίστα = list()
for κλειδί, τιμή in list(πλήθη.items()):
    λίστα.append((τιμή, κλειδί))

λίστα.sort(reverse=True)

for κλειδί, τιμή in λίστα[:10]:
    print(κλειδί, τιμή)
```

#Code: http://www.gr.py4e.com/code3/count3.py

Το πρώτο μέρος του προγράμματος που διαβάζει το αρχείο και δημιουργεί το λεξικό, που αντιστοιχίζει κάθε λέξη με το πλήθος εμφάνισης των λέξεων στο έγγραφο, δεν έχει αλλάξει. Αλλά, αντί να εκτυπώνουμε απλώς το πλήθη και να τερματίζουμε το πρόγραμμα, κατασκευάζουμε μια λίστα με πλειάδες `(τιμή, κλειδί)` και στη συνέχεια ταξινομούμε τη λίστα με αντίστροφη σειρά.

Από τη στιγμή που η τιμή είναι πρώτη, θα χρησιμοποιηθεί για τις συγκρίσεις. Εάν υπάρχουν περισσότερες από μία πλειάδες με την ίδια τιμή, θα κοιτάξει το δεύτερο στοιχείο (το κλειδί), επομένως οι πλειάδες των οποίων η τιμή είναι ίδια θα ταξινομηθούν περαιτέρω σε αλφαβητική σειρά του κλειδιού.

Στο τέλος γράφουμε έναν ωραίο βρόχο `for` που κάνει μια επανάληψη πολλαπλής εκχώρησης και εκτυπώνει τις δέκα πιο συνηθισμένες λέξεις, διατρέχοντας ένα τμήμα της λίστας (`lst[:10]`).

Έτσι τώρα η έξοδος μοιάζει, τελικά, με αυτό που θέλαμε για την ανάλυση συχνότητας λέξεων.

```
61 i
42 and
40 romeo
34 to
34 the
32 thou
32 juliet
30 that
29 my
24 thee
```

Το γεγονός ότι αυτή η περίπλοκη ανάλυση δεδομένων μπορεί να γίνει με ένα εύκολο στην κατανόηση πρόγραμμα Python, 19 γραμμών, είναι ένας λόγος για τον οποίο η Python είναι μια καλή επιλογή ως γλώσσα για την εξερεύνηση πληροφοριών.

Χρήση πλειάδων ως κλειδιών στα λεξικά

Επειδή οι πλειάδες είναι *κατακερματίσιμες* και οι λίστες όχι, αν θέλουμε να δημιουργήσουμε ένα *σύνθετο* κλειδί για χρήση σε ένα λεξικό, πρέπει να χρησιμοποιήσουμε ως κλειδί μια πλειάδα.

Θα χρειαζόμασταν ένα σύνθετο κλειδί εάν θέλαμε να δημιουργήσουμε έναν τηλεφωνικό κατάλογο, που αντιστοιχίζει ζεύγη επωνύμων, ονομάτων σε αριθμούς τηλεφώνου. Υποθέτοντας ότι έχουμε ορίσει τις μεταβλητές επώνυμο, όνομα και αριθμός, θα μπορούσαμε να γράψουμε μια εντολή εκχώρησης στο λεξικό ως εξής:

```
ευρετήριο[επώνυμο,όνομα] = αριθμός
```

Η έκφραση μέσα στις αγκύλες είναι πλειάδα. Θα μπορούσαμε να χρησιμοποιήσουμε την ανάθεση πλειάδας σε έναν βρόχο for, για να διασχίσουμε αυτό το λεξικό.

```
for επώνυμο, όνομα in ευρετήριο:
    print(όνομα, επώνυμο, ευρετήριο[επώνυμο,όνομα])
```

Αυτός ο βρόχος διασχίζει τα κλειδιά στο ευρετήριο, τα οποία είναι πλειάδες. Εκχωρεί τα στοιχεία κάθε πλειάδας στα επώνυμο και όνομα και στη συνέχεια εκτυπώνει το όνομα και τον αντίστοιχο αριθμό τηλεφώνου.

Ακολουθίες: συμβολοσειρές, λίστες και πλειάδες (- Oh My!)

Έχω επικεντρωθεί σε λίστες πλειάδων, αλλά σχεδόν όλα τα παραδείγματα σε αυτό το κεφάλαιο λειτουργούν επίσης με λίστες λιστών, πλειάδες πλειάδων και πλειάδες λιστών. Για να αποφευχθεί η απαρίθμηση των πιθανών συνδυασμών, μερικές φορές είναι πιο εύκολο να μιλάμε για ακολουθίες ακολουθιών.

Σε πολλά περιβάλλοντα, τα διαφορετικά είδη ακολουθιών (συμβολοσειρές, λίστες και πλειάδες) μπορούν να χρησιμοποιηθούν εναλλακτικά. Λοιπόν, πώς και γιατί επιλέγετε κάποιο αντί του άλλου ;

Για να ξεκινήσουμε με το προφανές, οι συμβολοσειρές είναι πιο περιορισμένες από τις άλλες ακολουθίες, επειδή τα στοιχεία πρέπει να είναι χαρακτήρες. Είναι επίσης αμετάβλητες. Εάν χρειάζεστε τη δυνατότητα να αλλάξετε τους χαρακτήρες μιας συμβολοσειράς (αντί για τη δημιουργία μιας νέας συμβολοσειράς), ίσως θελήσετε να χρησιμοποιήσετε μια λίστα χαρακτήρων.

Οι λίστες είναι πιο συχνά χρησιμοποιούμενες από τις πλειάδες, κυρίως επειδή είναι μεταβλητές. Αλλά υπάρχουν μερικές περιπτώσεις όπου μπορεί να προτιμήσετε τις πλειάδες:

1. Σε ορισμένες περιπτώσεις, όπως μια δήλωση return, είναι συντακτικά πιο απλό να δημιουργήσετε μια πλειάδα παρά μια λίστα. Σε άλλες περιπτώσεις, μπορεί να προτιμήσετε μια λίστα.

2. Εάν θέλετε να χρησιμοποιήσετε μια ακολουθία ως κλειδί λεξικού, πρέπει να χρησιμοποιήσετε έναν αμετάβλητο τύπο όπως πλειάδα ή συμβολοσειρά.

3. Εάν μεταβιβάζετε μια ακολουθία ως όρισμα σε μια συνάρτηση, η χρήση πλειάδων μειώνει την πιθανότητα απροσδόκητης συμπεριφοράς λόγω ψευδωνυμίας.

Επειδή οι πλειάδες είναι αμετάβλητες, δεν παρέχουν μεθόδους όπως `sort` και `reverse`, οι οποίες τροποποιούν τις υπάρχουσες λίστες. Ωστόσο, η Python παρέχει τις ενσωματωμένες συναρτήσεις `sorted` και `reversed`, οι οποίες λαμβάνουν οποιαδήποτε ακολουθία ως παράμετρο και επιστρέφουν μια νέα ακολουθία με τα ίδια στοιχεία με διαφορετική σειρά.

List comprehension (Κατανόηση λίστας)

Μερικές φορές θέλετε να δημιουργήσετε μια ακολουθία χρησιμοποιώντας δεδομένα από μια άλλη ακολουθία. Μπορείτε να το πετύχετε γράφοντας έναν βρόχο for και προσαρτώντας ένα στοιχείο κάθε φορά. Για παράδειγμα, αν θέλατε να μετατρέψετε μια λίστα συμβολοσειρών -- κάθε συμβολοσειρά αποθηκεύει ψηφία -- σε αριθμούς που μπορείτε να αθροίσετε, θα γράφατε:

```
list_of_ints_in_strings = ['42', '65', '12']
list_of_ints = []
for x in list_of_ints_in_strings:
    list_of_ints.append(int(x))

print(sum(list_of_ints))
```

Με την list comprehension, ο παραπάνω κώδικας μπορεί να γραφτεί με πιο συμπαγή τρόπο:

```
list_of_ints_in_strings = ['42', '65', '12']
list_of_ints = [ int(x) for x in list_of_ints_in_strings ]
print(sum(list_of_ints))
```

Εκσφαλμάτωση

Οι λίστες, τα λεξικά και οι πλειάδες είναι γνωστά γενικά ως *δομές δεδομένων*. Σε αυτό το κεφάλαιο αρχίζουμε να βλέπουμε σύνθετες δομές δεδομένων, όπως λίστες πλειάδων και λεξικά που περιέχουν πλειάδες ως κλειδιά και λίστες ως τιμές. Οι σύνθετες δομές

δεδομένων είναι χρήσιμες, αλλά είναι επιρρεπείς σε αυτό που αποκαλώ *σφάλματα σχήματος (shape errors)*. Δηλαδή, σφάλματα που προκαλούνται όταν μια δομή δεδομένων έχει λάθος τύπο, μέγεθος ή σύνθεση ή ίσως γράψετε κάποιον κώδικα και ξεχάσετε το σχήμα των δεδομένων σας και προκαλέσετε ένα σφάλμα. Για παράδειγμα, αν περιμένετε μια λίστα με έναν ακέραιο και σας δώσω έναν απλό ακέραιο (όχι σε λίστα), δεν θα λειτουργήσει.

Γλωσσάριο

DSU : Συντομογραφία του "decorate-sort-undecorate", ένα μοτίβο που περιλαμβάνει τη δημιουργία μιας λίστας πλειάδων, την ταξινόμηση και την εξαγωγή μέρους του αποτελέσματος.

gather : Η λειτουργία της συναρμολόγησης μιας πλειάδας ορίσματος μεταβλητού μήκους.

scatter : Η λειτουργία της αντιμετώπισης μιας ακολουθίας ως λίστας ορισμάτων.

singleton - μεμονωμένο : Μια λίστα (ή άλλη ακολουθία) με ένα μόνο στοιχείο.

δομή δεδομένων : Μια συλλογή σχετικών τιμών, συχνά οργανωμένη σε λίστες, λεξικά, πλειάδες κ.λπ.

κατακερματιζόμενος - hashable : Ένας τύπος που έχει συνάρτηση κατακερματισμού. Οι αμετάβλητοι τύποι όπως ακέραιοι, κινητής υποδιαστολής (float) και συμβολοσειρές μπορούν να κατακερματιστούν, μεταβλητοί τύποι όπως λίστες και λεξικά όχι.

πλειάδα : Μια αμετάβλητη ακολουθία στοιχείων.

πλειάδα σε ανάθεση : Μια εκχώρηση με μια ακολουθία στο δεξί μέλος και μια πλειάδα μεταβλητών στο αριστερό. Το δεξί μέλος αξιολογείται και στη συνέχεια τα στοιχεία του αντιστοιχίζονται στις μεταβλητές στα αριστερά.

συγκρίσιμος : Ένας τύπος όπου μια τιμή μπορεί να ελεγχθεί για να διαπιστωθεί εάν είναι μεγαλύτερη από, μικρότερη από ή ίση με μια άλλη τιμή του ίδιου τύπου. Οι τύποι που είναι συγκρίσιμοι μπορούν να τοποθετηθούν σε μια λίστα και να ταξινομηθούν.

σχήμα (μιας δομής δεδομένων) - shape : Σύνοψη του τύπου, του μεγέθους και της σύνθεσης μιας δομής δεδομένων.

Ασκήσεις

Άσκηση 1: Αναθεωρήστε ένα προηγούμενο πρόγραμμα ως εξής: Διαβάστε και αναλύστε τις γραμμές "From " και ανακτήστε τις διευθύνσεις από την κάθε γραμμή. Μετρήστε τον αριθμό των μηνυμάτων από κάθε άτομο χρησιμοποιώντας ένα λεξικό.

** Αφού διαβάσετε όλα τα δεδομένα, δημιουργήστε μια λίστα με πλειάδες (πλήθος, email) από το λεξικό. Στη συνέχεια, ταξινομήστε τη λίστα με αντίστροφη σειρά και εκτυπώστε το άτομο με τα περισσότερα μηνύματα.**

```
Δείγμα γραμμής:
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

Εισαγάγετε ένα όνομα αρχείου: mbox-short.txt
cwen@iupui.edu 5

Εισαγάγετε ένα όνομα αρχείου: mbox.txt
zqian@umich.edu 195
```

Άσκηση 2: Αυτό το πρόγραμμα μετράει την κατανομή της ώρας της ημέρας για κάθε ένα από τα μηνύματα. Μπορείτε να τραβήξετε την ώρα από τη γραμμή "From ", βρίσκοντας τη συμβολοσειρά χρόνου και, στη συνέχεια, χωρίζοντας τη συμβολοσειρά σε μέρη, χρησιμοποιώντας τον χαρακτήρα άνω και κάτω τελείας. Αφού υπολογίσετε τα πλήθη για κάθε ώρα, εκτυπώστε τα, ένα ανά γραμμή, ταξινομημένα ανά ώρα, όπως φαίνεται παρακάτω.

```
python timeofday.py
Εισαγάγετε ένα όνομα αρχείου: mbox-short.txt
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
```

```
18 1
19 1
```

Άσκηση 3: Γράψτε ένα πρόγραμμα που διαβάζει ένα αρχείο και τυπώνει τα *γράμματα* με φθίνουσα σειρά συχνότητας. Το πρόγραμμά σας θα πρέπει να μετατρέψει όλη την είσοδο σε πεζά και να μετράει μόνο τα γράμματα a-z. Το πρόγραμμά σας δεν πρέπει να μετράει κενά, ψηφία, σημεία στίξης ή οτιδήποτε άλλο εκτός από τα γράμματα a-z. Βρείτε δείγματα κειμένου από πολλές διαφορετικές γλώσσες και δείτε πώς η συχνότητα των γραμμάτων ποικίλλει μεταξύ των γλωσσών. Συγκρίνετε τα αποτελέσματά σας με τους πίνακες στο https://wikipedia.org/wiki/Letter_frequencies**.**

Κεφάλαιο 11

Κανονικές εκφράσεις (Regular expressions)

Μέχρι στιγμής διαβάζαμε αρχεία, αναζητούσαμε μοτίβα και εξαγάγαμε διάφορα κομμάτια γραμμών, που θεωρούσαμε ενδιαφέροντα. Χρησιμοποιήσαμε μεθόδους συμβολοσειρών, όπως `split` και `find` και χρησιμοποιούσαμε λίστες και διαμέριση συμβολοσειρών για να εξαγάγουμε τμήματα των γραμμών.

Αυτή η εργασία αναζήτησης και εξαγωγής είναι τόσο συνηθισμένη, που η Python έχει ένα πολύ ισχυρό άρθρωμα, που ονομάζεται *κανονικές εκφράσεις*, ή για συντομία *regex*, και χειρίζεται πολλές από αυτές τις εργασίες ιδιαίτερα έξυπνα. Ο λόγος που δεν έχουμε αναφερθεί στις κανονικές εκφράσεις νωρίτερα στο βιβλίο είναι επειδή, ενώ είναι πολύ ισχυρές, είναι λίγο περίπλοκες και η σύνταξή τους χρειάζεται μια κάποια εξοικείωση.

Οι κανονικές εκφράσεις είναι σχεδόν, από μόνες τους, μικρή γλώσσα προγραμματισμού για αναζήτηση και ανάλυση συμβολοσειρών. Στην πραγματικότητα, έχουν γραφτεί ολόκληρα βιβλία με θέμα τις κανονικές εκφράσεις. Σε αυτό το κεφάλαιο, θα καλύψουμε μόνο τα βασικά των κανονικών εκφράσεων. Για περισσότερες λεπτομέρειες σχετικά με τις κανονικές εκφράσεις, δείτε:

https://en.wikipedia.org/wiki/Regular_expression
https://docs.python.org/library/re.html

Το άρθρωμα κανονικών εκφράσεων `re` πρέπει να εισαχθεί στο πρόγραμμά σας για να μπορέσετε να το χρησιμοποιήσετε. Η απλούστερη χρήση του αρθρώματος κανονικών εκφράσεων είναι η συνάρτηση `search()`. Το παρακάτω πρόγραμμα δείχνει μια απλή χρήση της λειτουργίας αναζήτησης.

```
# Αναζητάμε τις γραμμές που περιέχουν τη λέξη 'From'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line):
        print(line)
```

#Code: http://www.gr.py4e.com/code3/re01.py

Ανοίγουμε το αρχείο, με βρόχο διατρέχουμε κάθε γραμμή και χρησιμοποιούμε την κανονική έκφραση `search()` για να εκτυπώνουμε μόνο τις γραμμές που περιέχουν τη συμβολοσειρά "From:". Αυτό το πρόγραμμα δεν χρησιμοποιεί την πραγματική ισχύ των κανονικών εκφράσεων, αφού θα μπορούσαμε να χρησιμοποιήσουμε εξίσου εύκολα το `line.find()` για να επιτύχουμε το ίδιο αποτέλεσμα.

Η ισχύς των κανονικών εκφράσεων αποκαλύπτεται όταν προσθέτουμε ειδικούς χαρακτήρες στη συμβολοσειρά αναζήτησης, που μας επιτρέπουν να ελέγχουμε με μεγαλύτερη ακρίβεια ποιες γραμμές ταιριάζουν με τη συμβολοσειρά. Η προσθήκη αυτών των ειδικών χαρακτήρων στην κανονική μας έκφραση μας επιτρέπει να κάνουμε πολύπλοκη αντιστοίχιση και εξαγωγή ενώ γράφουμε πολύ λίγο κώδικα.

Για παράδειγμα, ο χαρακτήρας περίφλεξης (^) χρησιμοποιείται σε κανονικές εκφράσεις για να ταιριάζει με την "αρχή" μιας γραμμής. Μπορούσαμε να αλλάξουμε το πρόγραμμά μας ώστε να εντοπίζει μόνο τις γραμμές στις οποίες το "From:" ήταν στην αρχή της γραμμής ως εξής:

```
# Αναζητάμε τις γραμμές που αρχίζουν με 'From'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line):
        print(line)
```

#Code: http://www.gr.py4e.com/code3/re02.py

Με αυτόν τον τρόπο θα εντοπίσουμε μόνο γραμμές που *ξεκινούν με* τη συμβολοσειρά "From:". Αυτό είναι ένα ακόμη πολύ απλό παράδειγμα, που θα μπορούσαμε να είχαμε κάνει, ισοδύναμα, με τη μέθοδο `startswith()`, από τη βιβλιοθήκη συμβολοσειρών. Αλλά χρησιμεύει για να εμπεδώσουμε το γεγονός ότι οι κανονικές εκφράσεις περιέχουν ειδικούς χαρακτήρες ενεργειών, που μας δίνουν περισσότερο έλεγχο ως προς το τι θα ταιριάζει με την κανονική έκφραση.

Ταίριασμα χαρακτήρων σε κανονικές εκφράσεις

Υπάρχει ένα πλήθος άλλων ειδικών χαρακτήρων μπαλαντέρ, που μας επιτρέπουν να δημιουργήσουμε ακόμη πιο ισχυρές κανονικές εκφράσεις. Ο πιο συχνά χρησιμοποιούμενος ειδικός χαρακτήρας είναι η τελεία, που ταιριάζει με οποιονδήποτε χαρακτήρα.

Στο ακόλουθο παράδειγμα, η κανονική έκφραση `F..m:` θα ταιριάζει με οποιαδήποτε από τις συμβολοσειρές "From:", "Fxxm:", "F12m:" ή "F!@m:" καθώς οι χαρακτήρες τελείας στην τυπική έκφραση ταιριάζουν με οποιονδήποτε χαρακτήρα.

```
# Αναζητάμε τις γραμμές που αρχίζουν με 'F', ακολουθούμενο
# από 2 χαρακτήρες, ακολουθούμενους από 'm:'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^F..m:', line):
        print(line)
```

#Code: http://www.gr.py4e.com/code3/re03.py

Το σύμβολο της τελείας είναι ιδιαίτερα ισχυρό όταν συνδυάζεται με τη δυνατότητα να υποδεικνύει ότι ένας χαρακτήρας μπορεί να επαναληφθεί όσες φορές χρειαστεί, χρησιμοποιώντας τους χαρακτήρες * ή + στην κανονική σας έκφραση. Αυτοί οι ειδικοί χαρακτήρες σημαίνουν ότι αντί να ταιριάζουν με έναν χαρακτήρα στη συμβολοσειρά αναζήτησης, ταιριάζουν με κανέναν ή περισσότερους χαρακτήρες (στην περίπτωση του αστερίσκου) ή έναν ή περισσότερους χαρακτήρες (στην περίπτωση του συμβόλου συν).

Μπορούμε να περιορίσουμε περαιτέρω τις γραμμές που ταιριάζουν χρησιμοποιώντας έναν επαναλαμβανόμενο χαρακτήρα *μπαλαντέρ* όπως στο ακόλουθο παράδειγμα:

```
# Αναζητάμε τις γραμμές που αρχίζουν με 'From'
# και περιέχουν ένα σύμβολο at
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:.+@', line):
        print(line)
```

#Code: http://www.gr.py4e.com/code3/re04.py

Η συμβολοσειρά αναζήτησης `^From:.+@` θα ταιριάξει με επιτυχία με τις γραμμές που ξεκινούν με "From:", ακολουθούμενο από έναν ή περισσότερους χαρακτήρες (`.+`), ακολουθούμενων από ένα σύμβολο at. Άρα αυτό θα ταιριάξει με την ακόλουθη γραμμή:

```
From: stephen.marquard@uct.ac.za
```

Μπορούμε να πούμε ότι ο χαρακτήρας μπαλαντέρ `.+` επεκτείνεται για να ταιριάζει με όλους τους χαρακτήρες μεταξύ του χαρακτήρα άνω και κάτω τελείας και του συμβόλου at.

```
From:.+@
```

Είναι καλό να σκεφτόμαστε τους χαρακτήρες συν και αστερίσκο ως "άπληστους". Για παράδειγμα, η ακόλουθη συμβολοσειρά θα ταιριάζει με το τελευταίο στο σύμβολο at της συμβολοσειράς καθώς το `.+` ωθεί προς τα έξω, όπως φαίνεται παρακάτω:

```
From: stephen.marquard@uct.ac.za, csev@umich.edu, and cwen @iupui.edu
```

Είναι δυνατόν να πείτε σε έναν αστερίσκο ή ένα σύμβολο συν να μην είναι τόσο "άπληστο" προσθέτοντας έναν επιπλέον χαρακτήρα. Δείτε τη λεπτομερή τεκμηρίωση για πληροφορίες σχετικά με την απενεργοποίηση της άπληστης συμπεριφοράς.

Εξαγωγή δεδομένων με χρήση κανονικών εκφράσεων

Εάν θέλουμε να εξαγάγουμε δεδομένα από μια συμβολοσειρά στην Python, μπορούμε να χρησιμοποιήσουμε τη μέθοδο `findall()`, για να εξαγάγουμε όλες τις υποσυμβολοσειρές που ταιριάζουν με μια κανονική έκφραση. Ας χρησιμοποιήσουμε ως παράδειγμα το να εξαγάγουμε οτιδήποτε μοιάζει με διεύθυνση email, από οποιαδήποτε γραμμή, ανεξαρτήτως μορφής. Για παράδειγμα, θέλουμε να τραβήξουμε τις διευθύνσεις email από καθεμία από τις ακόλουθες γραμμές:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
          for <source@collab.sakaiproject.org>;
Received: (from apache@localhost)
Author: stephen.marquard@uct.ac.za
```

Δεν θέλουμε να γράψουμε κώδικα για κάθε έναν από τους τύπους γραμμών, χωρίζοντας και τεμαχίζοντας διαφορετικά για κάθε γραμμή. Το πρόγραμμα που ακολουθεί χρησιμοποιεί το `findall()` για να βρει τις γραμμές με διευθύνσεις email και να εξαγάγει μία ή περισσότερες διευθύνσεις από καθεμία από αυτές τις γραμμές.

```
import re
s = 'A message from csev@umich.edu to cwen@iupui.edu about meeting @2PM'
lst = re.findall('\S+@\S+', s)
print(lst)
```

#Code: http://www.gr.py4e.com/code3/re05.py

Η μέθοδος `findall()` αναζητά τη συμβολοσειρά στο δεύτερο όρισμα και επιστρέφει μια λίστα με όλες τις συμβολοσειρές που μοιάζουν με διευθύνσεις email. Χρησιμοποιούμε μια ακολουθία δύο χαρακτήρων που ταιριάζει με έναν μη λευκό χαρακτήρα (`\S`).

Η έξοδος του προγράμματος θα είναι:

```
['csev@umich.edu', 'cwen@iupui.edu']
```

Μεταφράζοντας της κανονικής έκφρασης, λέμε, αναζητούμε υποσυμβολοσειρές που έχουν τουλάχιστον έναν μη λευκό χαρακτήρα, ακολουθούμενο από ένα σύμβολο at, ακολουθούμενο από τουλάχιστον έναν ακόμη μη λευκό χαρακτήρα. Το `\S+` ταιριάζει με όσο το δυνατόν περισσότερους μη κενούς χαρακτήρες.

Η κανονική έκφραση θα ταιριάξει δύο φορές (csev@umich.edu και cwen@iupui.edu), αλλά δεν θα ταιριάξει με τη συμβολοσειρά "@2PM" επειδή δεν υπάρχουν μη κενοί χαρακτήρες *πριν* από το σύμβολο at. Μπορούμε να χρησιμοποιήσουμε αυτήν την κανονική έκφραση σε ένα πρόγραμμα για να διαβάσουμε όλες τις γραμμές ενός αρχείου και να εκτυπώσουμε οτιδήποτε μοιάζει με διεύθυνση email ως εξής:

```
# Αναζητάμε τις γραμμές που περιέχουν ένα σύμβολο at μεταξύ χαρακτήρων
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('\S+@\S+', line)
    if len(x) > 0:
        print(x)
```

#Code: http://www.gr.py4e.com/code3/re06.py

Διαβάζουμε κάθε γραμμή και μετά εξάγουμε όλες τις υποσυμβολοσειρές που ταιριάζουν με την κανονική μας έκφραση. Μιας και το `findall()` επιστρέφει μια λίστα, απλώς ελέγχουμε εάν ο αριθμός των στοιχείων στη λίστα που επιστρέφεται είναι μεγαλύτερος από το μηδέν για να εκτυπώσουμε μόνο γραμμές όπου βρήκαμε τουλάχιστον μία υποσυμβολοσειρά που μοιάζει με διεύθυνση email.

Αν τρέξουμε το πρόγραμμα στο *mbox-short.txt* θα έχουμε την ακόλουθη έξοδο:

```
...
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
```

```
['apache@localhost)']
['source@collab.sakaiproject.org;']
['cwen@iupui.edu']
['source@collab.sakaiproject.org']
['cwen@iupui.edu']
['cwen@iupui.edu']
['wagnermr@iupui.edu']
```

Ορισμένες από τις διευθύνσεις ηλεκτρονικού ταχυδρομείου μας έχουν λανθασμένους χαρακτήρες, όπως "<" ή ";" στην αρχή ή στο τέλος. Ας δηλώσουμε ότι μας ενδιαφέρει μόνο το τμήμα της συμβολοσειράς που αρχίζει και τελειώνει με ένα γράμμα ή έναν αριθμό.

Για να το κάνουμε αυτό, χρησιμοποιούμε ένα άλλο χαρακτήρα των κανονικών εκφράσεων. Οι αγκύλες χρησιμοποιούνται για να υποδείξουν ένα σύνολο πολλών αποδεκτών χαρακτήρων. Κατά μία έννοια, το `\S` ζητά να ταιριάζει με το σύνολο των "μη λευκών χαρακτήρων". Τώρα θα είμαστε λίγο πιο σαφείς ως προς τους χαρακτήρες που θα ταιριάξουμε.

Εδώ είναι η νέα μας κανονική έκφραση:

```
[a-zA-Z0-9]\S*@\S*[a-zA-Z]
```

Αυτό γίνεται λίγο περίπλοκο και μπορείτε να αρχίσετε να βλέπετε γιατί είπαμε ότι οι κανονικές εκφράσεις αποτελούν μια ξεχωριστή, μικρή γλώσσα. Μεταφράζοντας αυτήν την κανονική έκφραση, αναζητούμε υποσυμβολοσειρές που ξεκινούν με ένα *μόνο* πεζό γράμμα, κεφαλαίο γράμμα ή αριθμό "[a-zA-Z0-9]", ακολουθούμενο από κανέναν ή περισσότερους μη λευκούς χαρακτήρες (`\S *`), ακολουθούμενων από ένα σύμβολο at, ακολουθούμενο από κανέναν ή περισσότερους μη λευκούς χαρακτήρες (`\S*`), ακολουθούμενων από ένα κεφαλαίο ή πεζό γράμμα. Σημειώστε ότι αλλάξαμε από + σε * για να υποδείξουμε κανέναν ή περισσότερους μη λευκούς χαρακτήρες, καθώς το `[a-zA-Z0-9]` είναι ήδη ένας μη κενός χαρακτήρας. Θυμηθείτε ότι το * ή + ισχύει για τον μεμονωμένο χαρακτήρα που βρίσκεται ακριβώς στα αριστερά του συν ή του αστερίσκου.

Εάν χρησιμοποιήσουμε αυτήν την έκφραση στο πρόγραμμά μας, τα δεδομένα που προκύπτουν είναι πολύ πιο καθαρά:

```
# Αναζητάμε τις γραμμές που περιέχουν ένα σύμβολο at μεταξύ χαρακτήρων.
# Οι χαρακτήρες πρέπει να είναι γράμματα ή αριθμοί
```

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('[a-zA-Z0-9]\S*@\S*[a-zA-Z]', line)
    if len(x) > 0:
        print(x)
```

#Code: http://www.gr.py4e.com/code3/re07.py

```
...
['wagnermr@iupui.edu']
['cwen@iupui.edu']
['postmaster@collab.sakaiproject.org']
['200801032122.m03LMFo4005148@nakamura.uits.iupui.edu']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['apache@localhost']
```

Παρατηρήστε ότι στις γραμμές `source@collab.sakaiproject.org`, η κανονική μας έκφραση απάλειψε δύο γράμματα στο τέλος της συμβολοσειράς (">;"). Αυτό συνέβη επειδή προσθέτοντας το `[a-zA-Z]` στο τέλος της κανονικής μας έκφρασης, απαιτούμε ότι οποιαδήποτε συμβολοσειρά βρίσκει ο αναλυτής κανονικής έκφρασης πρέπει να τελειώνει με ένα γράμμα. Έτσι, όταν βλέπει το ">" στο τέλος του "sakaiproject.org>;" απλά σταματά στο τελευταίο "ταιριαστό" γράμμα που βρήκε (δηλαδή, το "g" ήταν το τελευταίο καλό ταίριασμα).

Σημειώστε επίσης ότι η έξοδος του προγράμματος είναι μια λίστα Python που έχει μια συμβολοσειρά ως μοναδικό στοιχείο στη λίστα.

Συνδυασμός αναζήτησης και εξαγωγής

Αν θέλουμε να βρούμε αριθμούς σε γραμμές που ξεκινούν με τη συμβολοσειρά "X-", όπως:

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
```

δεν θέλουμε απλώς αριθμούς κινητής υποδιαστολής από οποιαδήποτε γραμμή. Θέλουμε να εξαγάγουμε αριθμούς μόνο από γραμμές που έχουν την παραπάνω σύνταξη.

Μπορούμε να κατασκευάσουμε την ακόλουθη κανονική έκφραση για να επιλέξουμε τις γραμμές:

```
^X-.*: [0-9.]+
```

Μεταφράζοντάς το, λέμε, θέλουμε γραμμές που ξεκινούν με `X-`, ακολουθούμενο από κανέναν ή περισσότερους χαρακτήρες (`.*`), ακολουθούμενων από άνω και κάτω τελεία (`:`) και μετά ένα κενό. Μετά το κενό αναζητούμε έναν ή περισσότερους χαρακτήρες που είναι είτε ψηφίο (0-9) είτε τελεία `[0-9.]+`. Σημειώστε ότι μέσα στις αγκύλες, η τελεία ταιριάζει με μια πραγματική τελεία (δηλαδή, δεν είναι χαρακτήρας μπαλαντέρ μεταξύ των αγκύλων).

Αυτή είναι μια πολύ αυστηρή έκφραση, που θα ταιριάξει μόνο με τις γραμμές που μας ενδιαφέρουν, ως εξής:

```
# Αναζητάμε τις γραμμές που ξεκινούν με `X`,
# ακολουθούμενο από μη λευκούς χαρακτήρες,
# ":", ακολουθούμενο από ένα κενό και
# οποιονδήποτε αριθμό.
# Ο αριθμός μπορεί να περιλαμβάνει υποδιαστολή.

import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^X\S*: [0-9.]+', line):
        print(line)
```

#Code: http://www.gr.py4e.com/code3/re10.py

Όταν εκτελούμε το πρόγραμμα, βλέπουμε τα δεδομένα, όμορφα φιλτραρισμένα, για να εμφανιστούν μόνο τις γραμμές που αναζητούμε.

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.6178
X-DSPAM-Probability: 0.0000
...
```

Αλλά τώρα πρέπει να λύσουμε άλλο ένα πρόβλημα, της εξαγωγής των αριθμών. Αν και θα ήταν αρκετά απλό να χρησιμοποιήσουμε το `split`, μπορούμε να χρησιμοποιήσουμε μια άλλη δυνατότητα κανονικών εκφράσεων, για αναζήτηση και ανάλυση της γραμμής, ταυτόχρονα.

Οι παρενθέσεις είναι άλλος ένας ειδικός χαρακτήρας των κανονικών εκφράσεων. Όταν προσθέτετε παρενθέσεις σε μια κανονική έκφραση, αυτές αγνοούνται όταν ταιριάζουν με τη συμβολοσειρά. Αλλά όταν χρησιμοποιείτε `findall()`, οι παρενθέσεις υποδεικνύουν ότι, ενώ θέλετε να ταιριάζει ολόκληρη η έκφραση, σας ενδιαφέρει να εξαγάγετε μόνο το τμήμα της υποσυμβολοσειράς, που ταιριάζει με την κανονική έκφραση που περιέχεται στις παρενθέσεις.

Κάνουμε λοιπόν την εξής αλλαγή στο πρόγραμμά μας:

```
# Αναζητάμε τις γραμμές που ξεκινούν με `X`,
# ακολουθούμενο από μη λευκούς χαρακτήρες,
# ":", ακολουθούμενο από ένα κενό και
# οποιονδήποτε αριθμό.
# Ο αριθμός μπορεί να περιλαμβάνει υποδιαστολή.

import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^X\S*: ([0-9.]+)', line)
    if len(x) > 0:
        print(x)
```

#Code: http://www.gr.py4e.com/code3/re11.py

Αντί να καλέσουμε τη `search()`, προσθέτουμε παρενθέσεις γύρω από το τμήμα της κανονικής έκφρασης που αντιπροσωπεύει τον αριθμό κινητής υποδιαστολής, για να υποδείξουμε ότι θέλουμε το `findall()` να μας δώσει πίσω μόνο το τμήμα αριθμού κινητής υποδιαστολής, της συμβολοσειράς, που ταιριάζει .

Η έξοδος από αυτό το πρόγραμμα είναι η εξής:

```
['0.8475']
['0.0000']
['0.6178']
['0.0000']
['0.6961']
['0.0000']
...
```

Οι αριθμοί εξακολουθούν να βρίσκονται σε μια λίστα και να πρέπει να μετατραπούν από συμβολοσειρές σε κινητή υποδιαστολή, αλλά χρησιμοποιήσαμε τη δύναμη των

κανονικών εκφράσεων για αναζήτηση και εξαγωγή των πληροφοριών που θωρούμε ενδιαφέρουσες.

Ως ένα άλλο παράδειγμα αυτής της τεχνικής, αν κοιτάξετε το αρχείο, υπάρχει ένας αριθμός γραμμών της μορφής:

```
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

Εάν θέλαμε να εξαγάγουμε όλους τους αριθμούς αναθεώρησης (τον ακέραιο αριθμό στο τέλος αυτών των γραμμών) χρησιμοποιώντας την ίδια τεχνική όπως παραπάνω, θα μπορούσαμε να γράψουμε το ακόλουθο πρόγραμμα:

```
# Αναζητάμε τις γραμμές που ξεκινούν με 'Details: rev='
# ακολουθούμενο από ψηφία.
# Στη συνέχεια εκτυπώνουμε τον αριθμό, αν βρεθεί.
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^Details:.*rev=([0-9]+)', line)
    if len(x) > 0:
        print(x)
```

#Code: http://www.gr.py4e.com/code3/re12.py

Μεταφράζοντας την κανονική μας έκφραση, αναζητούμε γραμμές που ξεκινούν με `Details:`, ακολουθούμενο από οποιονδήποτε αριθμό χαρακτήρων (`.*`), ακολουθούμενων από `rev=` και μετά από ένα ή περισσότερα ψηφία. Θέλουμε να βρούμε γραμμές που ταιριάζουν με ολόκληρη την έκφραση, αλλά θέλουμε να εξαγάγουμε μόνο τον ακέραιο αριθμό στο τέλος της γραμμής, επομένως περιβάλλουμε το `[0-9]+` με παρενθέσεις.

Όταν εκτελούμε το πρόγραμμα, έχουμε την ακόλουθη έξοδο:

```
['39772']
['39771']
['39770']
['39769']
...
```

Θυμηθείτε ότι το `[0-9]+` είναι "άπληστο" και προσπαθεί να δημιουργήσει όσο το δυνατόν μεγαλύτερη σειρά ψηφίων πριν εξαγάγει αυτά τα ψηφία. Αυτή η "άπληστη" συμπεριφορά είναι ο λόγος που παίρνουμε και τα πέντε ψηφία για κάθε αριθμό. Η

μονάδα κανονικής έκφρασης επεκτείνεται και προς τις δύο κατευθύνσεις μέχρι να συναντήσει ένα μη ψηφίο ή την αρχή ή το τέλος μιας γραμμής.

Τώρα μπορούμε να χρησιμοποιήσουμε κανονικές εκφράσεις για να επαναλάβουμε μια άσκηση από προηγούμενη ενότητα του βιβλίου, όπου μας ενδιέφερε η ώρα της ημέρας κάθε μηνύματος αλληλογραφίας. Αναζητούσαμε γραμμές της μορφής:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

και θέλαμε να εξαγάγουμε την ώρα της ημέρας από αυτές τις γραμμές. Προηγουμένως το υλοποιήσαμε με δύο κλήσεις του `split`. Πρώτα η γραμμή χωρίστηκε σε λέξεις και μετά βγάλαμε την πέμπτη λέξη και τη χωρίσαμε ξανά στον χαρακτήρα άνω και κάτω τελείας, για να βγάλουμε τους δύο χαρακτήρες που μας ενδιέφεραν.

Ενώ αυτό λειτούργησε, στην πραγματικότητα οδηγεί σε αρκετά εύθραυστο κώδικα που υποθέτει ότι οι γραμμές είναι σωστά διαμορφωμένες. Εάν επρόκειτο να προσθέσετε έλεγχο σφαλμάτων (ή ένα μεγάλο μπλοκ try/except), για να διασφαλίσετε ότι το πρόγραμμά σας δεν αποτυγχάνει όταν αντιμετωπίσει εσφαλμένα μορφοποιημένες γραμμές, ο κώδικας θα μεταφερόταν σε 10-15 γραμμές κώδικα, που θα ήταν αρκετά δύσκολο να διαβαστούν.

Μπορούμε να το κάνουμε αυτό με πολύ πιο απλό τρόπο, με την ακόλουθη κανονική έκφραση:

```
^From .* [0-9][0-9]:
```

Η μετάφραση αυτής της κανονικής έκφρασης είναι ότι αναζητούμε γραμμές που ξεκινούν με `From ` (προσέξτε το κενό διάστημα), ακολουθούμενο από οποιονδήποτε αριθμό χαρακτήρων (`.*`), ακολουθούμενων από ένα κενό, ακολουθούμενο από δύο ψηφία `[0 -9][0-9]`, ακολουθούμενα από χαρακτήρα άνω και κάτω τελείας. Αυτός είναι ο ορισμός των ειδών γραμμών που αναζητούμε.

Για να εξάγουμε μόνο την ώρα χρησιμοποιώντας το `findall()`, προσθέτουμε παρενθέσεις γύρω από τα δύο ψηφία ως εξής:

```
^From .* ([0-9][0-9]):
```

Αυτό έχει ως αποτέλεσμα το ακόλουθο πρόγραμμα:

```
# Αναζητάμε τις γραμμές που ξεκινούν με 'From ' και ένα σύνολο χαρακτήρων
# ακολουθούμενων από δύο ψηφία, ακολουθούμενα από ':'
# Στη συνέχεια εκτυπώνουμε τα ψηφία, εάν βρεθούν
import re
```

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^From .* ([0-9][0-9]):', line)
    if len(x) > 0: print(x)
```

#Code: http://www.gr.py4e.com/code3/re13.py

Όταν το πρόγραμμα εκτελείται, παράγει την ακόλουθη έξοδο:

```
['09']
['18']
['16']
['15']
...
```

Χαρακτήρας διαφυγής

Όταν χρησιμοποιούμε ειδικούς χαρακτήρες σε κανονικές εκφράσεις, όχι ως σύμβολα αλλά με την πραγματική τους αξία, για να ταιριάξουμε την αρχή ή το τέλος μιας γραμμής ή να καθορίσουμε μπαλαντέρ, χρειαζόμαστε έναν τρόπο για να υποδείξουμε ότι αυτοί οι χαρακτήρες είναι "κανονικοί" και θέλουμε να ταιριάξουμε τον πραγματικό χαρακτήρα, όπως ένα σύμβολο του δολαρίου ή έναν χαρακτήρα περίφλεξης (^) .

Μπορούμε να υποδείξουμε πως θέλουμε απλώς να ταιριάξουμε έναν χαρακτήρα προσθέτοντας ως πρόθεμα αυτού του χαρακτήρα μια ανάστροφη κάθετο. Για παράδειγμα, μπορούμε να βρούμε χρηματικά ποσά με την ακόλουθη κανονική έκφραση.

```
import re
x = 'We just received $10.00 for cookies.'
y = re.findall('\$[0-9.]+',x)
```

Εφόσον δίνουμε το πρόθεμα της ανάστροφης κάθετου πριν το σύμβολο του δολαρίου, η κανονική έκφραση το ταιριάζει με το σύμβολο του δολαρίου, στη συμβολοσειρά εισόδου, αντί να το ταιριάξει με το "τέλος γραμμής" και η υπόλοιπη τυπική έκφραση ταιριάζει με ένα ή περισσότερα ψηφία ή τον χαρακτήρα τελείας.

Σημείωση: Μέσα στις αγκύλες, οι χαρακτήρες δεν είναι "ειδικοί". Έτσι, όταν λέμε `[0-9.]`, σημαίνει πραγματικά ψηφία ή τελεία. Έξω από αγκύλες, η τελεία είναι ο χαρακτήρας "μπαλαντέρ" και ταιριάζει με οποιονδήποτε χαρακτήρα. Μέσα σε αγκύλες, η τελεία είναι τελεία.

Περίληψη

Αν και αυτά που αναφέραμε αγγίζουν μόνο την επιφάνεια της έννοιας των κανονικών εκφράσεων, μάθαμε λίγα πράγματα για τη γλώσσα των κανονικών εκφράσεων. Είναι συμβολοσειρές αναζήτησης με ειδικούς χαρακτήρες μέσα τους που μεταφέρουν τις επιθυμίες σας στο σύστημα κανονικής έκφρασης, ως προς το τι πρέπει να "ταιριάξει" και τι να εξάγεται από τις αντιστοιχισμένες συμβολοσειρές. Ακολουθούν μερικοί από αυτούς τους ειδικούς χαρακτήρες και τις ακολουθίες χαρακτήρων:

`^` Ταιριάζει την αρχή μιας γραμμής.

`$` Ταιριάζει το τέλος μιας γραμμής

`.` Ταιριάζει οποιονδήποτε χαρακτήρα (ένα μπαλαντέρ).

`\s` Ταιριάζει ένα λευκό χαρακτήρα (μη ορατό χαρακτήρα).

`\S` Ταιριάζει ένα μη λευκό χαρακτήρα (ορατό χαρακτήρα) (αντίθετο του \s).

`*` Επαναλαμβάνει τον/τους αμέσως προηγούμενο/ους χαρακτήρα/ες καμία ή περισσότερες φορές.

`*?` Επαναλαμβάνει τον/τους αμέσως προηγούμενο/ους χαρακτήρα/ες καμία ή περισσότερες φορές "μη-άπληστα".

`+` Επαναλαμβάνει τον/τους αμέσως προηγούμενο/ους χαρακτήρα/ες μία ή περισσότερες φορές.

`+?` Επαναλαμβάνει τον/τους αμέσως προηγούμενο/ους χαρακτήρα/ες μία ή περισσότερες φορές "μη-άπληστα".

`?` Επαναλαμβάνει τον/τους αμέσως προηγούμενο/ους χαρακτήρα/ες καμία ή μία φορά.

`??` Επαναλαμβάνει τον/τους αμέσως προηγούμενο/ους χαρακτήρα/ες καμία ή μία φορά "μη-άπληστα".

`[aeiou]` Ταιριάζει έναν μόνο χαρακτήρα από το δοθέν σύνολο. Στο παράδειγμα, θα ταιριάξει κάποιο από τα "a", "e", "i", "o" ή "u", αλλά όχι κάποιον άλλο χαρακτήρα.

`[a-z0-9]` Μπορείτε να καθορίσετε εύρος χαρακτήρων χρησιμοποιώντας το σύμβολο μείον. Αυτό το παράδειγμα αντιπροσωπεύει έναν μεμονωμένο χαρακτήρα, που πρέπει να είναι πεζός ή ψηφίο.

`[^A-Za-z]` Όταν ο πρώτος χαρακτήρας του συνόλου είναι η περίφλεξη, αντιστρέφει τη λογική. Αυτό το παράδειγμα ταιριάζει με έναν μεμονωμένο χαρακτήρα που είναι οτιδήποτε *εκτός* από ένα κεφαλαίο ή πεζό γράμμα.

`( )` Όταν προστίθενται παρενθέσεις σε μια κανονική έκφραση, αγνοούνται από την αντιστοίχιση, αλλά σας επιτρέπουν, όταν χρησιμοποιείτε το `findall()`, να εξαγάγετε ένα συγκεκριμένο υποσύνολο της αντιστοιχισμένης συμβολοσειράς αντί ολόκληρης της συμβολοσειράς .

`\b` Ταιριάζει με την κενή συμβολοσειρά, αλλά μόνο στην αρχή ή στο τέλος μιας λέξης.

`\B` Ταιριάζει με την κενή συμβολοσειρά, αλλά όχι στην αρχή ή στο τέλος μιας λέξης.

`\d` Ταιριάζει με οποιοδήποτε δεκαδικό ψηφίο. Ισοδύναμο με το σύνολο [0-9].

`\D` Ταιριάζει με οποιονδήποτε χαρακτήρα, μη-ψηφίο. Ισοδύναμο με το σύνολο [^0-9].

Μπόνους ενότητα για χρήστες Unix / Linux

Η υποστήριξη για αναζήτηση αρχείων με χρήση κανονικών εκφράσεων ενσωματώθηκε στο λειτουργικό σύστημα Unix από τη δεκαετία του 1960 και είναι διαθέσιμη σε όλες σχεδόν τις γλώσσες προγραμματισμού με τη μία ή την άλλη μορφή.

Στην πραγματικότητα, υπάρχει ένα πρόγραμμα γραμμής εντολών, ενσωματωμένο στο Unix που ονομάζεται *grep* (Generalized Regular Expression Parser) που κάνει σχεδόν ότι και τα παραδείγματα με τη `search()` σε αυτό το κεφάλαιο. Επομένως, εάν έχετε σύστημα Macintosh ή Linux, μπορείτε να δοκιμάσετε τις ακόλουθες εντολές στο παράθυρο της γραμμής εντολών σας.

```
$ grep '^From:' mbox-short.txt
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
```

Αυτό λέει στο `grep` να σας δείξει τις γραμμές που ξεκινούν με τη συμβολοσειρά "From:" στο αρχείο *mbox-short.txt*. Εάν πειραματιστείτε λίγο με την εντολή `grep` και διαβάσετε την τεκμηρίωση για το `grep`, θα βρείτε κάποιες ανεπαίσθητες διαφορές μεταξύ της υποστήριξης κανονικών εκφράσεων στην Python και της υποστήριξης κανονικών εκφράσεων στο `grep`. Για παράδειγμα, το `grep` δεν υποστηρίζει τον μη λευκό χαρακτήρα `\S`, επομένως θα χρειαστεί να χρησιμοποιήσετε τον ελαφρώς πιο περίπλοκο

συμβολισμό συνόλου `[^[:space:]]`, που σημαίνει απλώς αντιστοίχιση ενός μη λευκού χαρακτήρα.

Εκσφαλμάτωση

Η Python έχει κάποια απλή και στοιχειώδη ενσωματωμένη τεκμηρίωση που μπορεί να είναι αρκετά χρήσιμη εάν χρειάζεστε ένα γρήγορο φρεσκάρισμα της μνήμης σας, σχετικά με το ακριβές όνομα μιας συγκεκριμένης μεθόδου. Αυτή η τεκμηρίωση μπορεί να προβληθεί στον διερμηνέα Python σε διαδραστική λειτουργία.

Μπορείτε να εμφανίσετε ένα διαδραστικό σύστημα βοήθειας χρησιμοποιώντας το `help()`.

```
>>> help()

help> modules
```

Εάν γνωρίζετε ποιο άρθρωμα θέλετε να χρησιμοποιήσετε, μπορείτε να χρησιμοποιήσετε την εντολή `dir()` για να βρείτε τις μεθόδους στη λειτουργική μονάδα ως εξής:

```
>>> import re
>>> dir(re)
[.. 'compile', 'copy_reg', 'error', 'escape', 'findall',
'finditer', 'match', 'purge', 'search', 'split', 'sre_compile',
'sre_parse', 'sub', 'subn', 'sys', 'template']
```

Μπορείτε επίσης να λάβετε μια μικρή τεκμηρίωσης για μια συγκεκριμένη μέθοδο χρησιμοποιώντας την εντολή dir.

```
>>> help (re.search)
Help on function search in module re:

search(pattern, string, flags=0)
    Scan through string looking for a match to the pattern, returning
    a match object, or None if no match was found.
>>>
```

Η ενσωματωμένη τεκμηρίωση δεν είναι ιδιαίτερα εκτενής, αλλά μπορεί να είναι χρήσιμη όταν βιάζεστε ή δεν έχετε πρόσβαση σε πρόγραμμα περιήγησης ιστού ή μηχανή αναζήτησης.

Γλωσσάριο

brittle code - εύθραυστος κώδικας : Κωδικός που λειτουργεί όταν τα δεδομένα εισόδου είναι σε συγκεκριμένη μορφή, αλλά είναι επιρρεπής σε σφάλματα εκτέλεσης, εάν υπάρχει κάποια απόκλιση από τη σωστή μορφή. Αυτό το ονομάζουμε "εύθραυστος κώδικας" γιατί "σπάει" εύκολα.

grep : Μια εντολή διαθέσιμη στα περισσότερα συστήματα Unix, που αναζητά μέσα σε αρχεία κειμένου γραμμές, που ταιριάζουν με κανονικές εκφράσεις. Το όνομα της εντολής σημαίνει "Generalized Regular Expression Parser - Αναλυτής Γενικοποιημένης Κανονική Έκφραση".

άπληστο ταίριασμα : Η έννοια ότι οι χαρακτήρες + και * σε μια κανονική έκφραση επεκτείνονται προς τα έξω για να ταιριάζουν με τη μεγαλύτερη δυνατή συμβολοσειρά.

κανονική έκφραση - regular expression : Μια γλώσσα για την έκφραση πιο σύνθετων συμβολοσειρών αναζήτησης. Μια κανονική έκφραση μπορεί να περιέχει ειδικούς χαρακτήρες, που υποδεικνύουν ότι μια αναζήτηση ταιριάζει μόνο στην αρχή ή στο τέλος μιας γραμμής ή πολλές άλλες παρόμοιες δυνατότητες. \

μπαλαντέρ - wild card : Ένας ειδικός χαρακτήρας, που ταιριάζει με κάθε χαρακτήρα. Στις κανονικές εκφράσεις ο χαρακτήρας μπαλαντέρ είναι η τελεία.

Ασκήσεις

Άσκηση 1: Γράψτε ένα απλό πρόγραμμα για την προσομοίωση της λειτουργίας της εντολής grep στο Unix. Ζητήστε από τον χρήστη να εισαγάγει μια κανονική έκφραση και μετρήστε τον αριθμό των γραμμών του αρχείου mbox.txt, που ταιριάζουν με την κανονική έκφραση:

```
$ python grep.py
Εισαγάγετε μια κανονική έκφραση: ^Author
mbox.txt had 1798 lines that matched ^Author

$ python grep.py
Εισαγάγετε μια κανονική έκφραση: ^X-
mbox.txt had 14368 lines that matched ^X-
```

```
$ python grep.py
Εισαγάγετε μια κανονική έκφραση: java$
mbox.txt had 4175 lines that matched java$
```

Άσκηση 2: Γράψτε ένα πρόγραμμα για να αναζητήσετε γραμμές της μορφής:

```
New Revision: 39772
```

** Εξάγετε τον αριθμό, από κάθε γραμμή, χρησιμοποιώντας μια κανονική έκφραση και τη μέθοδο `findall()`. Υπολογίστε τον μέσο όρο των αριθμών και εκτυπώστε τον μέσο όρο ως ακέραιο.**

```
Εισαγάγετε το αρχείο:mbox.txt
38549

Εισαγάγετε το αρχείο:mbox-short.txt
39756
```

Κεφάλαιο 12

Δικτυακά προγράμματα

Ενώ πολλά από τα παραδείγματα αυτού του βιβλίου έχουν επικεντρωθεί στην ανάγνωση αρχείων και στην αναζήτηση δεδομένων σε αυτά τα αρχεία, υπάρχουν πολλές διαφορετικές πηγές πληροφοριών, αν λάβουμε υπόψιν μας και το Διαδίκτυο.

Σε αυτό το κεφάλαιο θα προσποιηθούμε ότι είμαστε ένα πρόγραμμα περιήγησης ιστού και θα ανακτήσουμε ιστοσελίδες, χρησιμοποιώντας το πρωτόκολλο μεταφοράς υπερκειμένου (HTTP). Στη συνέχεια θα διαβάσουμε τα δεδομένα της ιστοσελίδας και θα τα αναλύσουμε.

Πρωτόκολλο μεταφοράς υπερκειμένου - HTTP

Το πρωτόκολλο δικτύου, που κινεί τον Ιστό, είναι στην πραγματικότητα πολύ απλό και υπάρχει ενσωματωμένη υποστήριξη στην Python, που ονομάζεται `socket`, η οποία καθιστά πολύ εύκολη τη δημιουργία συνδέσεων δικτύου και την ανάκτηση δεδομένων μέσω αυτών των υποδοχών σε ένα πρόγραμμα Python.

Μια *υποδοχή* μοιάζει πολύ με ένα αρχείο, εκτός από το ότι μια μεμονωμένη υποδοχή παρέχει αμφίδρομη σύνδεση μεταξύ δύο προγραμμάτων. Μπορείτε να διαβάζετε και να γράφετε στην ίδια υποδοχή. Εάν γράψετε κάτι σε μια υποδοχή, αποστέλλεται στην εφαρμογή στο άλλο άκρο της υποδοχής. Εάν διαβάζετε από την υποδοχή, σας δίνονται τα δεδομένα που έχει στείλει η άλλη εφαρμογή.

Αλλά αν προσπαθήσετε να διαβάσετε μια υποδοχή όταν το πρόγραμμα στην άλλη άκρη της υποδοχής δεν έχει στείλει δεδομένα, απλά κάθεστε και περιμένετε. Εάν τα προγράμματα και στα δύο άκρα της υποδοχής απλώς περιμένουν κάποια δεδομένα χωρίς να στείλουν τίποτα, θα περιμένουν για πολύ μεγάλο χρονικό διάστημα, επομένως ένα σημαντικό μέρος των προγραμμάτων που επικοινωνούν μέσω Διαδικτύου είναι να έχουν κάποιο είδος πρωτοκόλλου.

Ένα πρωτόκολλο είναι ένα σύνολο από ακριβείς κανόνες που καθορίζουν ποιος θα ξεκινήσει πρώτος, τι πρέπει να κάνουν και, στη συνέχεια, ποιες είναι οι απαντήσεις σε αυτό το μήνυμα και ποιος στέλνει στη συνέχεια και ούτω καθεξής. Κατά μία έννοια, οι

δύο εφαρμογές σε κάθε άκρο της υποδοχής χορεύουν και φροντίζουν να μην πατούν η μία στα δάχτυλα των ποδιών της άλλης.

Υπάρχουν πολλά έγγραφα που περιγράφουν αυτά τα πρωτόκολλα δικτύου. Το πρωτόκολλο μεταφοράς υπερκειμένου περιγράφεται στο ακόλουθο έγγραφο:

https://www.w3.org/Protocols/rfc2616/rfc2616.txt

Αυτό είναι ένα μεγάλο και περίπλοκο έγγραφο 176 σελίδων με πολλές λεπτομέρειες. Αν το βρίσκετε ενδιαφέρον, μη διστάσετε να το διαβάσετε όλο. Αλλά αν ρίξετε μια ματιά στη σελίδα 36 του RFC2616, θα βρείτε τη σύνταξη για το αίτημα GET. Για να ζητήσουμε ένα έγγραφο από έναν διακομιστή web, κάνουμε μια σύνδεση με τον διακομιστή `www.pr4e.org` στη θύρα 80 και, στη συνέχεια, στέλνουμε μια γραμμή της μορφής

```
GET http://data.pr4e.org/romeo.txt HTTP/1.0
```

όπου η δεύτερη παράμετρος είναι η ιστοσελίδα που ζητάμε και στη συνέχεια στέλνουμε και μια κενή γραμμή. Ο διακομιστής ιστού θα απαντήσει με ορισμένες πληροφορίες κεφαλίδας για το έγγραφο και μια κενή γραμμή ακολουθούμενη από το περιεχόμενο του εγγράφου.

Το απλούστερο πρόγραμμα περιήγησης ιστού στον κόσμο

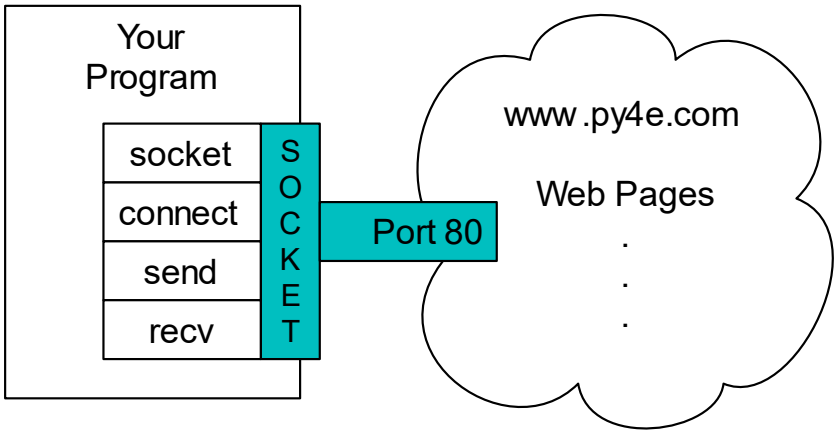
Ίσως ο ευκολότερος τρόπος για να σας δείξω πώς λειτουργεί το πρωτόκολλο HTTP είναι να γράψουμε ένα πολύ απλό πρόγραμμα Python, που πραγματοποιεί μια σύνδεση με έναν διακομιστή ιστού και ακολουθώντας τους κανόνες του πρωτοκόλλου HTTP ζητά ένα έγγραφο και εμφανίζει αυτό που του στέλνει ο διακομιστής.

```
import socket
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if len(data) < 1:
        break
    print(data.decode(),end='')
mysock.close()
```

#Code: http://www.gr.py4e.com/code3/socket1.py

Πρώτα το πρόγραμμα κάνει μια σύνδεση στη θύρα 80 του διακομιστή www.py4e.com. Δεδομένου ότι το πρόγραμμά μας παίζει το ρόλο του "περιηγητή Ιστού", το πρωτόκολλο HTTP λέει ότι πρέπει να στείλουμε την εντολή GET ακολουθούμενη από μια κενή γραμμή. Το `\r\n` σημαίνει EOL (τέλος γραμμής), οπότε το `\r\n\r\n` σημαίνει τίποτα ανάμεσα σε δύο ακολουθίες EOL. Αυτό είναι το ισοδύναμο μιας κενή γραμμής.



Εικόνα 12.1: Μια σύνδεση Υποδοχής - Socket

Μόλις στείλουμε αυτήν την κενή γραμμή, γράφουμε έναν βρόχο που λαμβάνει δεδομένα σε κομμάτια 512 χαρακτήρων από την υποδοχή και εκτυπώνει τα δεδομένα, μέχρι να μην υπάρχουν άλλα δεδομένα για ανάγνωση (δηλαδή, η recv() επιστρέφει μια κενή συμβολοσειρά).

Το πρόγραμμα παράγει την ακόλουθη έξοδο:

```
HTTP/1.1 200 OK
Date: Wed, 11 Apr 2018 18:52:55 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Sat, 13 May 2017 11:22:22 GMT
ETag: "a7-54f6609245537"
Accept-Ranges: bytes
Content-Length: 167
Cache-Control: max-age=0, no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Connection: close
Content-Type: text/plain

But soft what light through yonder window breaks
```

```
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Η έξοδος ξεκινά με κεφαλίδες που στέλνει ο διακομιστής ιστού, για να περιγράψει το έγγραφο. Για παράδειγμα, η κεφαλίδα `Content-Type` υποδεικνύει ότι το έγγραφο είναι έγγραφο απλού κειμένου (`text/plain`).

Αφού ο διακομιστής μας στείλει τις κεφαλίδες, προσθέτει μια κενή γραμμή, για να υποδείξει το τέλος των κεφαλίδων και, στη συνέχεια, στέλνει τα πραγματικά δεδομένα του αρχείου *romeo.txt*.

Αυτό το παράδειγμα δείχνει πώς να κάνετε μια σύνδεση δικτύου χαμηλού επιπέδου με υποδοχές. Οι υποδοχές μπορούν να χρησιμοποιηθούν για την επικοινωνία με έναν διακομιστή ιστού ή με έναν διακομιστή αλληλογραφίας ή και με πολλά άλλα είδη διακομιστών. Το μόνο που χρειάζεται είναι να βρείτε το έγγραφο που περιγράφει το πρωτόκολλο και να γράψετε τον κώδικα για να στείλετε και να λάβετε τα δεδομένα σύμφωνα με το πρωτόκολλο αυτό.

Ωστόσο, δεδομένου ότι το πρωτόκολλο που χρησιμοποιούμε πιο συχνά είναι το πρωτόκολλο ιστού HTTP, η Python έχει μια ειδική βιβλιοθήκη, ειδικά σχεδιασμένη ώστε να υποστηρίζει το πρωτόκολλο HTTP για την ανάκτηση εγγράφων και δεδομένων μέσω του ιστού.

Μία από τις απαιτήσεις για τη χρήση του πρωτοκόλλου HTTP είναι η ανάγκη αποστολής και λήψης των δεδομένων ως αντικείμενα bytes, αντί των συμβολοσειρών. Στο προηγούμενο παράδειγμα, οι μέθοδοι `encode()` και `decode()` μετατρέπουν τις συμβολοσειρές σε αντικείμενα bytes και το αντίστροφο.

Το επόμενο παράδειγμα χρησιμοποιεί τη σημείωση `b''` για να καθορίσει ότι μια μεταβλητή θα πρέπει να αποθηκευτεί ως αντικείμενο bytes. Το `encode()` και το `b''` είναι ισοδύναμα.

```
>>> b'Hello world'
b'Hello world'
>>> 'Hello world'.encode()
b'Hello world'
```

Ανάκτηση μιας εικόνας μέσω HTTP

Στο παραπάνω παράδειγμα, ανακτήσαμε ένα απλό αρχείο κειμένου, που είχε νέες γραμμές στο αρχείο και απλώς αντιγράψαμε τα δεδομένα στην οθόνη, καθώς το πρόγραμμα εκτελούνταν. Μπορούμε να χρησιμοποιήσουμε ένα παρόμοιο πρόγραμμα για να ανακτήσουμε μια εικόνα, χρησιμοποιώντας το HTTP. Αντί να αντιγράψουμε τα δεδομένα στην οθόνη, καθώς εκτελείται το πρόγραμμα, συγκεντρώνουμε τα δεδομένα σε μια συμβολοσειρά, κόβουμε τις κεφαλίδες και, στη συνέχεια, αποθηκεύουμε τα δεδομένα εικόνας σε ένα αρχείο ως εξής:

```
import socket
import time

HOST = 'data.pr4e.org'
PORT = 80
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect((HOST, PORT))
mysock.sendall(b'GET http://data.pr4e.org/cover3.jpg HTTP/1.0\r\n\r\n')
count = 0
picture = b""

while True:
    data = mysock.recv(5120)
    if len(data) < 1: break
    #time.sleep(0.25)
    count = count + len(data)
    print(len(data), count)
    picture = picture + data

mysock.close()

# Αναζήτησε το τέλος της κεφαλίδας (2 CRLF)
pos = picture.find(b"\r\n\r\n")
print('Header length', pos)
print(picture[:pos].decode())

# Προσπέρασε την κεφαλίδα και αποθήκευσε τα δεδομένα της εικόνας
picture = picture[pos+4:]
fhand = open("stuff.jpg", "wb")
```

```
fhand.write(picture)
fhand.close()
```

#Code: http://www.gr.py4e.com/code3/urljpeg.py

Όταν το πρόγραμμα εκτελείται, παράγει την ακόλουθη έξοδο:

```
$ python urljpeg.py
5120 5120
5120 10240
4240 14480
5120 19600
...
5120 214000
3200 217200
5120 222320
5120 227440
3167 230607
Header length 393
HTTP/1.1 200 OK
Date: Wed, 11 Apr 2018 18:54:09 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Mon, 15 May 2017 12:27:40 GMT
ETag: "38342-54f8f2e5b6277"
Accept-Ranges: bytes
Content-Length: 230210
Vary: Accept-Encoding
Cache-Control: max-age=0, no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Connection: close
Content-Type: image/jpeg
```

Μπορείτε να δείτε ότι για αυτήν τη διεύθυνση url, η κεφαλίδα `Content-Type` υποδεικνύει ότι το σώμα του εγγράφου είναι μια εικόνα (`image/jpeg`). Μόλις το πρόγραμμα ολοκληρωθεί, μπορείτε να προβάλετε τα δεδομένα της εικόνας ανοίγοντας το αρχείο `stuff.jpg` σε ένα πρόγραμμα προβολής εικόνων.

Καθώς εκτελείται το πρόγραμμα, μπορείτε να δείτε ότι δεν λαμβάνουμε 5120 χαρακτήρες, κάθε φορά που καλούμε τη μέθοδο `recv()`. τη στιγμή που καλούμε `recv()`

λαμβάνουμε όσους χαρακτήρες έχουν μεταφερθεί από τον διακομιστή ιστού, μέσω του δικτύου σε εμάς. Σε αυτό το παράδειγμα με κάθε μας αίτημα, λαμβάνουμε από μόλις 3200 χαρακτήρες έως και 5120 χαρακτήρες δεδομένων.

Τα αποτελέσματά σας μπορεί να διαφέρουν ανάλογα με την ταχύτητα του δικτύου σας. Σημειώστε επίσης ότι στην τελευταία κλήση στο `recv()` λαμβάνουμε 3167 byte, που είναι το τέλος της ροής, και στην επόμενη κλήση στο `recv()` λαμβάνουμε μια συμβολοσειρά μηδενικού μήκους, που μας ενημερώνει ότι ο διακομιστής έχει καλέσει την `close()` στο δικό του άκρο της υποδοχής πράγμα που σημαίνει ότι δεν υπάρχουν άλλα δεδομένα.

Μπορούμε να επιβραδύνουμε τις διαδοχικές μας κλήσεις της `recv()` αφαιρώντας το σχολιασμό από την κλήση της `time.sleep()`. Με αυτόν τον τρόπο, περιμένουμε ένα τέταρτο του δευτερολέπτου μετά από κάθε κλήση, ώστε ο διακομιστής να μπορεί να μας "προλάβει" και να μας στείλει περισσότερα δεδομένα πριν καλέσουμε ξανά τη `recv()`. Προσθέτοντας την καθυστέρηση αυτή, το πρόγραμμα εκτελείται ως εξής:

```
$ python urljpeg.py
5120 5120
5120 10240
5120 15360
...
5120 225280
5120 230400
207 230607
Header length 393
HTTP/1.1 200 OK
Date: Wed, 11 Apr 2018 21:42:08 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Mon, 15 May 2017 12:27:40 GMT
ETag: "38342-54f8f2e5b6277"
Accept-Ranges: bytes
Content-Length: 230210
Vary: Accept-Encoding
Cache-Control: max-age=0, no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Connection: close
Content-Type: image/jpeg
```

Τώρα, εκτός από την πρώτη και την τελευταία κλήση στο `recv()`, λαμβάνουμε πλέον 5120 χαρακτήρες, κάθε φορά που ζητάμε νέα δεδομένα.

Υπάρχει ένα buffer μεταξύ του διακομιστή που κάνει αιτήματα `send()` και της εφαρμογής μας που κάνει το αιτήματα `recv()`. Όταν εκτελούμε το πρόγραμμα με ενεργοποιημένη την καθυστέρηση, κάποια στιγμή ο διακομιστής μπορεί να γεμίσει το buffer στην υποδοχή και να αναγκαστεί να σταματήσει μέχρι το πρόγραμμά μας να αρχίσει να αδειάζει το buffer. Η παύση είτε της εφαρμογής αποστολής είτε της εφαρμογής λήψης ονομάζεται "έλεγχος ροής" (flow control).

Ανάκτηση ιστοσελίδων με την `urllib`

Ενώ μπορούμε να στείλουμε και να λάβουμε με μη αυτόματο τρόπο δεδομένα μέσω HTTP, χρησιμοποιώντας τη βιβλιοθήκη socket, υπάρχει ένας πολύ απλούστερος τρόπος για να εκτελέσετε αυτήν την κοινή εργασία στην Python, χρησιμοποιώντας τη βιβλιοθήκη `urllib`.

Χρησιμοποιώντας το `urllib`, μπορείτε να μεταχειριστείτε μια ιστοσελίδα σαν αρχείο. Απλώς υποδεικνύετε ποια ιστοσελίδα θέλετε να ανακτήσετε και το `urllib` χειρίζεται όλα τα στοιχεία του πρωτοκόλλου HTTP και της κεφαλίδας.

Ο ισοδύναμος κώδικας για την ανάγνωση του αρχείου *romeo.txt* από τον ιστό, χρησιμοποιώντας το `urllib` είναι ο εξής:

```
import urllib.request

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
for line in fhand:
    print(line.decode().strip())
```

#Code: http://www.gr.py4e.com/code3/urllib1.py

Μόλις ανοίξει η ιστοσελίδα με το `urllib.request.urlopen`, μπορούμε να την αντιμετωπίσουμε σαν αρχείο και να την διαβάσουμε χρησιμοποιώντας έναν βρόχο `for`.

Όταν εκτελείται το πρόγραμμα, βλέπουμε μόνο την έξοδο των περιεχομένων του αρχείου. Οι κεφαλίδες εξακολουθούν να αποστέλλονται, αλλά ο κωδικός `urllib` δεσμεύει τις κεφαλίδες και επιστρέφει μόνο τα δεδομένα σε εμάς.

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Για παράδειγμα, μπορούμε να γράψουμε ένα πρόγραμμα για να ανακτήσουμε τα δεδομένα για το «romeo.txt» και να υπολογίσουμε τη συχνότητα κάθε λέξης στο αρχείο ως εξής:

```
import urllib.request, urllib.parse, urllib.error

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')

πλήθη = dict()
for γραμμή in fhand:
    λέξεις = γραμμή.decode().split()
    for λέξη in λέξεις:
        πλήθη[λέξη] = πλήθη.get(λέξη, 0) + 1
print(πλήθη)
```

#Code: http://www.gr.py4e.com/code3/urlwords.py

Και πάλι, μόλις ανοίξουμε την ιστοσελίδα, μπορούμε να τη διαβάσουμε σαν τοπικό αρχείο.

Ανάγνωση δυαδικών αρχείων χρησιμοποιώντας το `urllib`

Μερικές φορές θέλετε να ανακτήσετε ένα αρχείο μη-κειμένου (ή δυαδικό - binary), όπως ένα αρχείο εικόνας ή βίντεο. Τα δεδομένα σε αυτά τα αρχεία γενικά δεν έχουν νόημα να εκτυπωθούν, αλλά με τη βοήθειά τους και χρησιμοποιώντας το `urllib` μπορείτε εύκολα να δημιουργήσετε ένα αντίγραφο μιας διεύθυνσης URL σε ένα τοπικό αρχείο, στον σκληρό σας δίσκο.

Το μοτίβο είναι να ανοίξετε τη διεύθυνση URL και να χρησιμοποιήσετε το `read` για να κατεβάσετε ολόκληρο το περιεχόμενο του εγγράφου, σε μια μεταβλητή συμβολοσειράς (`img`) και στη συνέχεια να γράψετε αυτές τις πληροφορίες σε ένα τοπικό αρχείο ως εξής:

```
import urllib.request, urllib.parse, urllib.error

img = urllib.request.urlopen('http://data.pr4e.org/cover3.jpg').read()
fhand = open('cover3.jpg', 'wb')
fhand.write(img)
fhand.close()
```

#Code: http://www.gr.py4e.com/code3/curl1.py

Αυτό το πρόγραμμα διαβάζει όλα τα δεδομένα ταυτόχρονα, μέσω του δικτύου, και τα αποθηκεύει στη μεταβλητή `img`, στην κύρια μνήμη του υπολογιστή σας, στη συνέχεια ανοίγει το αρχείο `cover3.jpg` και εγγράφει τα δεδομένα στο δίσκο σας. Το όρισμα `wb`, στο `open()`, ανοίγει ένα δυαδικό αρχείο μόνο για εγγραφή. Αυτό το πρόγραμμα θα λειτουργήσει μόν εάν το μέγεθος του αρχείου είναι μικρότερο από το μέγεθος της μνήμης του υπολογιστή σας.

Ωστόσο, εάν πρόκειται για μεγάλο αρχείο ήχου ή βίντεο, αυτό το πρόγραμμα ενδέχεται να διακοπεί ή τουλάχιστον να εκτελείται εξαιρετικά αργά όταν η μνήμη του υπολογιστή σας εξαντληθεί. Προκειμένου να αποφευχθεί η εξάντληση της μνήμης, ανακτούμε τα δεδομένα σε μπλοκ (ή buffers) και στη συνέχεια γράφουμε κάθε μπλοκ στον δίσκο μας, πριν ανακτήσουμε το επόμενο μπλοκ. Με αυτόν τον τρόπο το πρόγραμμα μπορεί να διαβάσει αρχεία οποιουδήποτε μεγέθους χωρίς να χρησιμοποιεί όλη τη μνήμη που έχει ο υπολογιστή σας.

```
import urllib.request, urllib.parse, urllib.error

img = urllib.request.urlopen('http://data.pr4e.org/cover3.jpg')
fhand = open('cover3.jpg', 'wb')
size = 0
while True:
    info = img.read(100000)
    if len(info) < 1: break
    size = size + len(info)
    fhand.write(info)

print(size, 'χαρακτήρες αντιγράφηκαν.')
fhand.close()
```

#Code: http://www.gr.py4e.com/code3/curl2.py

Σε αυτό το παράδειγμα, διαβάζουμε μόνο 100.000 χαρακτήρες κάθε φορά και στη συνέχεια γράφουμε αυτούς τους χαρακτήρες στο αρχείο `cover3.jpg`, πριν ανακτήσουμε τους επόμενους 100.000 χαρακτήρες δεδομένων από τον ιστό.

Αυτό το πρόγραμμα εκτελείται ως εξής:

```
python curl2.py
230210 χαρακτήρες αντιγράφηκαν.
```

Ανάλυση HTML και web scraping (ιστοσυγκομιδή)

Μία από τις κοινές χρήσεις της δυνατότητας `urllib` στην Python είναι η *ιστοσυγκομιδή (web scraping)*. Η ιστοσυγκομιδή είναι όταν γράφουμε ένα πρόγραμμα που προσποιείται ότι είναι πρόγραμμα περιήγησης ιστού, ανακτά σελίδες και, στη συνέχεια, εξετάζει τα δεδομένα σε αυτές τις σελίδες αναζητώντας μοτίβα.

Για παράδειγμα, μια μηχανή αναζήτησης, όπως η Google θα εξετάσει την πηγή μιας ιστοσελίδας, θα εξαγάγει τους συνδέσμους προς άλλες σελίδες και θα ανακτήσει αυτές τις σελίδες, θα εξάγει συνδέσμους και ούτω καθεξής. Χρησιμοποιώντας αυτήν την τεχνική, το Google *ανιχνεύει*, περνάει, σχεδόν από όλες τις σελίδες του ιστού.

Η Google χρησιμοποιεί επίσης τη συχνότητα των συνδέσμων, από σελίδες που βρίσκει προς μια συγκεκριμένη σελίδα, ως μέτρο του πόσο "σημαντική" είναι μια σελίδα και πόσο ψηλά πρέπει να εμφανίζεται η σελίδα αυτή στα αποτελέσματα αναζήτησής της.

Ανάλυση HTML χρησιμοποιώντας κανονικές εκφράσεις

Ένας απλός τρόπος ανάλυσης HTML είναι η χρήση κανονικών εκφράσεων για επανειλημμένη αναζήτηση και εξαγωγή υποσυμβολοσειρών που ταιριάζουν με ένα συγκεκριμένο μοτίβο.

Εδώ είναι μια απλή ιστοσελίδα:

```
<h1>The First Page</h1>
<p>
If you like, you can switch to the
<a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>.
</p>
```

Μπορούμε να κατασκευάσουμε μια καλοσχηματισμένη κανονική έκφραση, για να ταιριάξουμε και να εξαγάγουμε τους συνδέσμου; από το παραπάνω κείμενο ως εξής:

```
href="http[s]?://.+?"
```

Η κανονική έκφρασή μας αναζητά συμβολοσειρές που ξεκινούν με *href="http://* ή *href="https://*, ακολουθούμενες από έναν ή περισσότερους χαρακτήρες (`.+?`), ακολουθούμενες από ένα άλλο διπλό εισαγωγικό . Το ερωτηματικό πίσω από το `[s]?` υποδηλώνει αναζήτηση για τη συμβολοσειρά "http" ακολουθούμενη από κανένα ή ένα "s".

Το ερωτηματικό που προστέθηκε στο `.+?` υποδηλώνει ότι το ταίριασμα πρέπει να γίνει με "μη άπληστο" τρόπο και όχι με "άπληστο". Ένα μη άπληστο ταίριασμα προσπαθεί να βρει τη *μικρότερη* δυνατή συμβολοσειρά που ταιριάζει και ένα άπληστο ταίριασμα προσπαθεί να βρει τη *μεγαλύτερη* δυνατή συμβολοσειρά.

Προσθέτουμε παρενθέσεις στην κανονική μας έκφραση για να υποδείξουμε ποιο τμήμα της αντιστοιχισμένης συμβολοσειράς θα θέλαμε να εξαγάγουμε και παράγουμε το ακόλουθο πρόγραμμα:

```
# Αναζήτηση συνδέσμων εντός του URL εισόδου
import urllib.request, urllib.parse, urllib.error
import re
import ssl

# Αγνόηση των σφαλμάτων πιστοποιητικού SSL
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Εισάγεται - ')
html = urllib.request.urlopen(url, context=ctx).read()
links = re.findall(b'href="(http[s]?://.*?)"', html)
for link in links:
    print(link.decode())
```

#Code: http://www.gr.py4e.com/code3/urlregex.py

Η βιβλιοθήκη `ssl` επιτρέπει σε αυτό το πρόγραμμα να έχει πρόσβαση σε ιστότοπους που επιβάλλουν αυστηρά το HTTPS. Η μέθοδος `read` επιστρέφει τον πηγαίο κώδικα HTML ως αντικείμενο bytes αντί να επιστρέφει ένα αντικείμενο HTTPResponse. Η μέθοδος κανονικής έκφρασης `findall` θα μας δώσει μια λίστα με όλες τις συμβολοσειρές που ταιριάζουν με την κανονική μας έκφραση, επιστρέφοντας μόνο το κείμενο σύνδεσης μεταξύ των διπλών εισαγωγικών.

Όταν εκτελούμε το πρόγραμμα και εισάγουμε μια διεύθυνση URL, έχουμε την ακόλουθη έξοδο:

```
Εισάγετε - https://docs.python.org
https://docs.python.org/3/index.html
https://www.python.org/
https://docs.python.org/3.8/
```

```
https://docs.python.org/3.7/
https://docs.python.org/3.5/
https://docs.python.org/2.7/
https://www.python.org/doc/versions/
https://www.python.org/dev/peps/
https://wiki.python.org/moin/BeginnersGuide
https://wiki.python.org/moin/PythonBooks
https://www.python.org/doc/av/
https://www.python.org/
https://www.python.org/psf/donations/
http://sphinx.pocoo.org/
```

Οι κανονικές εκφράσεις λειτουργούν πολύ καλά όταν το HTML σας είναι καλά μορφοποιημένο και προβλέψιμο. Αλλά επειδή υπάρχουν πολλές "σπασμένες" σελίδες HTML εκεί έξω, μια λύση που χρησιμοποιεί μόνο κανονικές εκφράσεις μπορεί είτε να χάσει ορισμένους έγκυρους συνδέσμους είτε να καταλήξει με κατεστραμμένα δεδομένα.

Αυτό μπορεί να λυθεί χρησιμοποιώντας μια ισχυρή βιβλιοθήκη ανάλυσης HTML.

Ανάλυση HTML χρησιμοποιώντας το BeautifulSoup

Παρόλο που η HTML μοιάζει με την XML[1] και ορισμένες σελίδες έχουν κατασκευαστεί προσεκτικά ώστε να είναι XML, οι περισσότερες σελίδες HTML γενικά περιέχουν σφάλματα τέτοια που αναγκάζουν έναν αναλυτή XML να απορρίψει ολόκληρη τη σελίδα του HTML ως ακατάλληλα σχηματισμένη.

Υπάρχει ένας αριθμός βιβλιοθηκών Python που μπορούν να σας βοηθήσουν να αναλύσετε HTML και να εξαγάγετε δεδομένα από σελίδες. Κάθε μία από αυτές τις βιβλιοθήκες έχει τα δυνατά και τα αδύνατα σημεία της και μπορείτε να επιλέξετε κάποια, με βάση τις ανάγκες σας.

Ως παράδειγμα, θα χρησιμοποιήσουμε τη βιβλιοθήκη *BeautifulSoup* και απλώς θα αναλύσουμε ορισμένες εισόδους HTML και θα εξαγάγουμε συνδέσμους. Η BeautifulSoup ανέχεται εξαιρετικά ελαττωματικό HTML και εξακολουθεί να σας επιτρέπει να εξαγάγετε εύκολα τα δεδομένα που χρειάζεστε. Μπορείτε να κατεβάσετε και να εγκαταστήσετε τον κώδικα BeautifulSoup από:

[1] Η μορφή XML περιγράφεται στο επόμενο κεφάλαιο.

https://pypi.python.org/pypi/beautifulsoup4

Πληροφορίες σχετικά με την εγκατάσταση του BeautifulSoup με το εργαλείο ευρετηρίου πακέτων pip της Python είναι διαθέσιμες στη διεύθυνση:

https://packaging.python.org/tutorials/installing-packages/

Θα χρησιμοποιήσουμε το urllib για να διαβάσουμε τη σελίδα και στη συνέχεια θα χρησιμοποιήσουμε τη BeautifulSoup για να εξαγάγουμε τα χαρακτηριστικά href από τις ετικέτες αγκύρωσης (a).

```
# Για να το εκτελέσετε, κάντε λήψη του αρχείου zip BeautifulSoup
# από  http://www.py4e.com/code3/bs4.zip
# και αποσυμπιέστε το στον ίδιο κατάλογο με αυτό το αρχείο

import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl

# Αγνόηση των σφαλμάτων πιστοποιητικού SSL
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Εισάγετε - ')
html = urllib.request.urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, 'html.parser')

# Ανάκτηση όλων των ετικετών αγκύρωσης
tags = soup('a')
for tag in tags:
    print(tag.get('href', None))
```

#Code: http://www.gr.py4e.com/code3/urllinks.py

Το πρόγραμμα ζητά μια διεύθυνση ιστού, στη συνέχεια ανοίγει την ιστοσελίδα, διαβάζει τα δεδομένα και μεταβιβάζει τα δεδομένα στον αναλυτή BeautifulSoup και, στη συνέχεια, ανακτά όλες τις ετικέτες αγκύρωσης και εκτυπώνει το χαρακτηριστικό href για κάθε ετικέτα.

Όταν το πρόγραμμα εκτελείται, παράγει την ακόλουθη έξοδο:

```
Εισάγετε - https://docs.python.org
genindex.html
```

```
py-modindex.html
https://www.python.org/
#
whatsnew/3.6.html
whatsnew/index.html
tutorial/index.html
library/index.html
reference/index.html
using/index.html
howto/index.html
installing/index.html
distributing/index.html
extending/index.html
c-api/index.html
faq/index.html
py-modindex.html
genindex.html
glossary.html
search.html
contents.html
bugs.html
about.html
license.html
copyright.html
download.html
https://docs.python.org/3.8/
https://docs.python.org/3.7/
https://docs.python.org/3.5/
https://docs.python.org/2.7/
https://www.python.org/doc/versions/
https://www.python.org/dev/peps/
https://wiki.python.org/moin/BeginnersGuide
https://wiki.python.org/moin/PythonBooks
https://www.python.org/doc/av/
genindex.html
py-modindex.html
https://www.python.org/
#
copyright.html
```

```
https://www.python.org/psf/donations/
bugs.html
http://sphinx.pocoo.org/
```

Αυτή η λίστα είναι πολύ μεγάλη επειδή ορισμένες ετικέτες αγκύρωσης HTML είναι σχετικές διαδρομές (π.χ. tutorial/index.html) ή αναφορές στη σελίδα (π.χ. '#'), που δεν περιλαμβάνουν "http://" ή "https:// », που αποτελούσε απαίτησή μας όταν χρησιμοποιήσαμε κανονική έκφραση.

Μπορείτε επίσης να χρησιμοποιήσετε το BeautifulSoup για να βγάλετε διάφορα μέρη κάθε ετικέτας:

```
# Για να το εκτελέσετε, κάντε λήψη του αρχείου zip BeautifulSoup
# από  http://www.py4e.com/code3/bs4.zip
# και αποσυμπιέστε το στον ίδιο κατάλογο με αυτό το αρχείο

from urllib.request import urlopen
from bs4 import BeautifulSoup
import ssl

# Αγνόηση των σφαλμάτων πιστοποιητικού SSL
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Εισάγετε - ')
html = urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, "html.parser")

# Ανάκτηση όλων των ετικετών αγκύρωσης
tags = soup('a')
for tag in tags:
    # Εξέταση των μερών μιας ετικέτας
    print('TAG:', tag)
    print('URL:', tag.get('href', None))
    print('Contents:', tag.contents[0])
    print('Attrs:', tag.attrs)
```

#Code: http://www.gr.py4e.com/code3/urllink2.py

```
python urllink2.py
Εισάγετε - http://www.dr-chuck.com/page1.htm
```

```
TAG: <a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>
URL: http://www.dr-chuck.com/page2.htm
Content: ['\nSecond Page']
Attrs: [('href', 'http://www.dr-chuck.com/page2.htm')]
```

Το `html.parser` είναι ο αναλυτής HTML που περιλαμβάνεται στην τυπική βιβλιοθήκη της Python 3. Πληροφορίες για άλλους αναλυτές HTML είναι διαθέσιμες στη διεύθυνση:

http://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser

Αυτά τα παραδείγματα μόνο ενδεικτικά μας μυούν στη δύναμη της BeautifulSoup για την ανάλυση HTML.

Μπόνους ενότητα για χρήστες Unix / Linux

Εάν διαθέτετε υπολογιστή Linux, Unix ή Macintosh, πιθανότατα διαθέτετε ενσωματωμένες εντολές, στο λειτουργικό σας σύστημα, που ανακτούν τόσο απλό κείμενο όσο και δυαδικά αρχεία, χρησιμοποιώντας τα πρωτόκολλα HTTP ή File Transfer (FTP). Μία από αυτές τις εντολές είναι το `curl`:

```
$ curl -O http://www.py4e.com/cover.jpg
```

Η εντολή `curl` είναι συντομογραφία για το "copy URL" και έτσι τα δύο παραδείγματα που παρατέθηκαν προηγουμένως, για την ανάκτηση δυαδικών αρχείων με χρήση της `urllib` ονομάζονται έξυπνα `curl1.py` και `curl2.py` στο www.gr.py4e.com/code3, καθώς υλοποιούν παρόμοια λειτουργικότητα με την εντολή `curl`. Υπάρχει επίσης ένα δείγμα προγράμματος `curl3.py`, που κάνει αυτήν την εργασία λίγο πιο αποτελεσματικά, σε περίπτωση που θέλετε πραγματικά να ενσωματώσετε αυτό το μοτίβο σε κάποιο πρόγραμμα που γράφετε.

Μια δεύτερη εντολή που λειτουργεί πολύ παρόμοια είναι η `wget`:

```
$ wget http://www.py4e.com/cover.jpg
```

Και οι δύο αυτές εντολές απλοποιούν την ανάκτηση ιστοσελίδων και απομακρυσμένων αρχείων.

Γλωσσάριο

BeautifulSoup : Μια βιβλιοθήκη Python για την ανάλυση εγγράφων HTML και την εξαγωγή δεδομένων από έγγραφα HTML που αντισταθμίζει τις περισσότερες από

τις ατέλειες στην HTML, που τα προγράμματα περιήγησης γενικά αγνοούν. Μπορείτε να κατεβάσετε τον κώδικα BeautifulSoup από www.crummy.com.

ανίχνευση - spider : Η πράξη μιας μηχανής αναζήτησης Ιστού, που ανακτά μια σελίδα και στη συνέχεια όλες τις σελίδες που συνδέονται με τη σελίδα αυτή και ούτω καθεξής μέχρι να έχουν σχεδόν όλες τις σελίδες στο Διαδίκτυο, τις οποίες χρησιμοποιούν για τη δημιουργία του ευρετηρίου αναζήτησής τους.

θύρα - port : Ένας αριθμός που γενικά υποδεικνύει με ποια εφαρμογή επικοινωνείτε όταν πραγματοποιείτε σύνδεση υποδοχής σε διακομιστή. Για παράδειγμα, η κυκλοφορία Ιστού χρησιμοποιεί συνήθως τη θύρα 80, ενώ η κυκλοφορία ηλεκτρονικού ταχυδρομείου χρησιμοποιεί τη θύρα 25.

ιστοσυγκομιδή - web scraping: Όταν ένα πρόγραμμα προσποιείται ότι είναι πρόγραμμα περιήγησης ιστού και ανακτά μια ιστοσελίδα και, στη συνέχεια, εξετάζει το περιεχόμενο της ιστοσελίδας. Συχνά τα προγράμματα αυτά ακολουθούν τους συνδέσμους σε μια σελίδα για να βρουν την επόμενη σελίδα, ώστε να μπορούν να διασχίσουν ένα δίκτυο σελίδων ή ένα κοινωνικό δίκτυο.

υποδοχή - socket : Μια σύνδεση δικτύου μεταξύ δύο εφαρμογών, όπου οι εφαρμογές μπορούν να στέλνουν και να λαμβάνουν δεδομένα προς οποιαδήποτε κατεύθυνση.

Ασκήσεις

Άσκηση 1: Αλλάξτε το πρόγραμμα υποδοχής `socket1.py` ώστε να ζητά από τον χρήστη τη διεύθυνση URL και να μπορεί να διαβάσει οποιαδήποτε ιστοσελίδα. Μπορείτε να χρησιμοποιήσετε το `split('/')` για να σπάσετε τη διεύθυνση URL στα συστατικά μέρη της, ώστε να μπορέσετε να εξαγάγετε το όνομα του κεντρικού υπολογιστή για την κλήση `connect` της υποδοχής. Προσθέστε τον έλεγχο σφαλμάτων χρησιμοποιώντας `try` και `except` για να χειριστείτε την κατάσταση όπου ο χρήστης εισάγει μια διεύθυνση URL με ακατάλληλη μορφή ή ανύπαρκτη.

Άσκηση 2: Αλλάξτε το πρόγραμμα υποδοχής σας έτσι ώστε να μετράει τον αριθμό των χαρακτήρων που έχει λάβει και να σταματά να εμφανίζει οποιοδήποτε κείμενο όταν εμφανίσει 3000 χαρακτήρες. Το πρόγραμμα θα πρέπει να ανακτήσει ολόκληρο το έγγραφο και να μετρήσει τον συνολικό αριθμό χαρακτήρων και να εμφανίσει τον αριθμό των χαρακτήρων στο τέλος του εγγράφου.

Άσκηση 3: Χρησιμοποιήστε το `urllib` για να αναπαραγάγετε την προηγούμενη άσκηση ώστε (1) να ανακτά το έγγραφο από μια διεύθυνση URL, (2) να εμφανίζει έως 3000 χαρακτήρες και (3) να μετρά τον συνολικό αριθμό χαρακτήρων στο έγγραφο. Μην ανησυχείτε για τις κεφαλίδες σε αυτήν την άσκηση, απλώς εμφανίστε τους πρώτους 3000 χαρακτήρες του περιεχομένου του εγγράφου.

Άσκηση 4: Αλλάξτε το πρόγραμμα `urllinks.py` για να εξαγάγετε και να μετρήσετε τις ετικέτες παραγράφου (p) από το ανακτηθέν έγγραφο HTML και να εμφανίσετε την καταμέτρηση των παραγράφων ως έξοδο του προγράμματός σας. Μην εμφανίσετε το κείμενο των παραγράφων, μόνο μετρήστε τις. Δοκιμάστε το πρόγραμμά σας σε πολλές μικρές ιστοσελίδες καθώς και σε ορισμένες μεγαλύτερες ιστοσελίδες.

Άσκηση 5: (Για προχωρημένους) Αλλάξτε το πρόγραμμα υποδοχής έτσι ώστε να εμφανίζει δεδομένα μόνο αφού ληφθούν οι κεφαλίδες και μια κενή γραμμή. Να θυμάστε ότι το `recv` λαμβάνει χαρακτήρες (νέες γραμμές και όλους), όχι γραμμές.

Κεφάλαιο 13

Χρήση Υπηρεσιών Ιστού

Από τη στιγμή που έγινε εύκολη η ανάκτηση και η ανάλυση εγγράφων μέσω HTTP, χρησιμοποιώντας προγράμματα, δεν χρειάστηκε πολύς χρόνος για να αναπτύξουμε μια προσέγγιση παραγωγής εγγράφων σχεδιασμένων ειδικά για κατανάλωση από άλλα προγράμματα (δηλαδή, όχι HTML για εμφάνιση σε πρόγραμμα περιήγησης).

Υπάρχουν δύο κοινές μορφές που χρησιμοποιούμε κατά την ανταλλαγή δεδομένων στον ιστό. Η eXtensible Markup Language (XML) ή Επεκτάσιμη Γλώσσα Σήμανσης χρησιμοποιείται εδώ και πολύ καιρό και είναι η καταλληλότερη για έγγραφα τυποποιημένα για ανταλλαγή δεδομένων. Όταν τα προγράμματα θέλουν απλώς να ανταλλάξουν λεξικά, λίστες ή άλλες εσωτερικές πληροφορίες μεταξύ τους, χρησιμοποιούν JavaScript Object Notation (JSON) ή Σημειογραφία Αντικειμένου JavaScript (δείτε www.json.org). Θα εξετάσουμε και τις δύο μορφές.

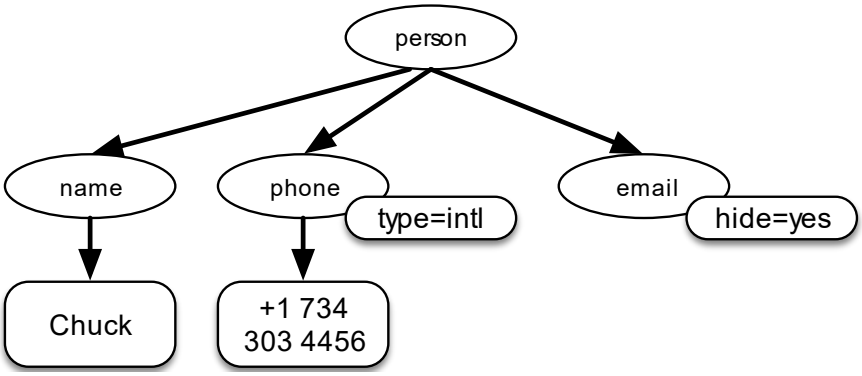
eXtensible Markup Language - XML

Η XML μοιάζει πολύ με την HTML, αλλά η XML είναι πιο δομημένη. Ακολουθεί ένα δείγμα εγγράφου XML:

```
<άτομο>
  <όνομα>Chuck</όνομα>
  <τηλέφωνο τύπος="εσωτερικό">
    +1 734 303 4456
  </τηλέφωνο>
  <email κρυφό="ναι" />
</άτομο>
```

Κάθε ζεύγος ετικετών ανοίγματος (π.χ. <άτομο>) και κλεισίματος (π.χ. </άτομο>) αντιπροσωπεύει ένα *στοιχείο* ή *κόμβο* με το ίδιο όνομα με αυτό της ετικέτας (π.χ. άτομο). Κάθε στοιχείο μπορεί να έχει κάποιο κείμενο, ορισμένα χαρακτηριστικά (π.χ. κρυφό) και άλλα εμφωλευμένα στοιχεία. Εάν ένα στοιχείο XML είναι κενό (δηλαδή, δεν έχει περιεχόμενο), μπορεί να απεικονίζεται από μια ετικέτα που κλείνει αυτόματα (π.χ. <email />).

Συχνά είναι χρήσιμο να σκεφτόμαστε ένα έγγραφο XML ως μια δομή δέντρου, όπου υπάρχει ένα κορυφαίο στοιχείο (εδώ το: άτομο) και οι άλλες ετικέτες (π.χ. τηλέφωνο) σχεδιάζονται ως *παιδιά* των στοιχείων *γονέα* τους.



Εικόνα 13.1: Μια αναπαράσταση δέντρου της XML

Ανάλυση XML

Εδώ είναι μια απλή εφαρμογή, που αναλύει ορισμένα δεδομένα XML και εξάγει κάποια στοιχεία δεδομένων από το XML:

```
import xml.etree.ElementTree as ET

data = '''
<άτομο>
  <όνομα>Chuck</όνομα>
  <τηλέφωνο τύπος="εσωτερικό">
    +1 734 303 4456
  </τηλέφωνο>
  <email κρυφό="ναι" />
</άτομο>'''

tree = ET.fromstring(data)
print('Όνομα:', tree.find('όνομα').text)
print('Χαρακτηριστικό:', tree.find('email').get('κρυφό'))
```

#Code: http://www.gr.py4e.com/code3/xml1.py

Το τριπλό μονό εισαγωγικό (`'''`), καθώς και το τριπλό διπλό εισαγωγικό (`"""`), επιτρέπουν τη δημιουργία συμβολοσειρών που εκτείνονται σε πολλές γραμμές.

Η κλήση `fromstring` μετατρέπει την αναπαράσταση συμβολοσειράς του XML σε ένα "δέντρο" στοιχείων XML. Όταν το XML αναπαρασταθεί με δέντρο, έχουμε μια σειρά από

μεθόδους που μπορούμε να καλέσουμε για να εξαγάγουμε τμήματα δεδομένων από τη συμβολοσειρά XML. Η συνάρτηση `find` πραγματοποιεί αναζήτηση στο δέντρο XML και ανακτά το στοιχείο που ταιριάζει με την καθορισμένη ετικέτα.

```
Όνομα: Chuck
Χαρακτηριστικό:: yes
```

Το XML σε αυτό το παράδειγμα είναι αρκετά απλό, αποδεικνύεται όμως ότι υπάρχουν πολλοί κανόνες σχετικά με έγκυρη XML. Η χρήση ενός αναλυτή XML, όπως το `ElementTree`, μας επιτρέπει να εξαγάγουμε δεδομένα από το XML χωρίς να ανησυχούμε για τους περίπλοκους, σε κάποιες περιπτώσεις, κανόνες σύνταξης XML.

Προσπέλαση των κόμβων με επανάληψη

Συχνά το XML έχει πολλούς κόμβους και πρέπει να γράψουμε έναν βρόχο για να επεξεργαστούμε όλους τους κόμβους. Στο παρακάτω πρόγραμμα, διατρέχουμε με βρόχο όλους τους κόμβους χρήστη:

```
import xml.etree.ElementTree as ET

input = '''
<stuff>
  <χρήστες>
    <χρήστης x="2">
      <id>001</id>
      <όνομα>Chuck</όνομα>
    </χρήστης>
    <χρήστης x="7">
      <id>009</id>
      <όνομα>Brent</όνομα>
    </χρήστης>
  </χρήστες>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('χρήστες/χρήστης')
print('Πλήθος χρηστών:', len(lst))

for item in lst:
```

```
    print('Όνομα', item.find('όνομα').text)
    print('Id', item.find('id').text)
    print('Χαρακτηριστικό', item.get('x'))
```

#Code: http://www.gr.py4e.com/code3/xml2.py

Η μέθοδος `findall` ανακτά μια λίστα Python με υποδέντρα που αντιπροσωπεύουν τις δομές χρήστης στο δέντρο XML. Στη συνέχεια, μπορούμε να γράψουμε ένα βρόχο `for`, που εξετάζει κάθε κόμβο χρήστη και εκτυπώνει τα στοιχεία κειμένου όνομα και `id` καθώς και το χαρακτηριστικό x του κόμβου (χρήστης).

```
Πλήθος χρηστών: 2
Όνομα Chuck
Id 001
Χαρακτηριστικό 2
Όνομα Brent
Id 009
Χαρακτηριστικό 7
```

Είναι σημαντικό να συμπεριληφθούν όλα τα στοιχεία γονικού επιπέδου στη δήλωση `findall`, εκτός από το στοιχείο ανώτατου επιπέδου (π.χ. χρήστες/χρήστης). Διαφορετικά, η Python δεν θα βρει κανέναν, επιθυμητό, κόμβο.

```
import xml.etree.ElementTree as ET

input = '''
<stuff>
  <χρήστες>
    <χρήστης x="2">
      <id>001</id>
      <όνομα>Chuck</όνομα>
    </χρήστης>
    <χρήστης x="7">
      <id>009</id>
      <όνομα>Brent</όνομα>
    </χρήστης>
  </χρήστες>
</stuff>'''

stuff = ET.fromstring(input)

lst = stuff.findall('χρήστες/χρήστης')
```

```
print('Πλήθος χρηστών:', len(lst))

lst2 = stuff.findall('χρήστης')
print('Πλήθος χρηστών:', len(lst2))
```

`lst` αποθηκεύει όλα τα στοιχεία `user` τα οποία είναι εμφωλευμένα στον γονέα `users`. `lst2` αναζητά τα στοιχεία `user` τα οποία δεν είναι εμφωλευμένα, μέσα στο στοιχείο ανώτατου επιπέδου `stuff`, τα οποία δεν υπάρχουν.

```
User count: 2
User count: 0
```

JavaScript Object Notation - JSON

Η μορφή JSON εμπνεύστηκε από τη μορφή αντικειμένου και πίνακα, που χρησιμοποιείται στη γλώσσα JavaScript. Αλλά δεδομένου ότι η Python επινοήθηκε πριν από την JavaScript, η σύνταξη της Python για λεξικά και λίστες επηρέασε τη σύνταξη του JSON. Έτσι, η μορφή του JSON είναι σχεδόν πανομοιότυπη με έναν συνδυασμό λιστών και λεξικών Python.

Εδώ παραθέτω μια κωδικοποίηση JSON, που είναι περίπου ισοδύναμη με την απλή XML παραπάνω:

```
{
  "όνομα" : "Chuck",
  "τηλέφωνο" : {
    "τύπος" : "εσωτερικό",
    "αριθμός" : "+1 734 303 4456"
   },
   "email" : {
     "κρυφό" : "ναι"
   }
}
```

Θα παρατηρήσετε κάποιες διαφορές. Αρχικά, στην XML, μπορούμε να προσθέσουμε χαρακτηριστικά, όπως το "εσωτερικό" στην ετικέτα "τηλέφωνο". Στο JSON, έχουμε απλώς ζεύγη κλειδιού-τιμής. Επίσης, η ετικέτα "άτομο" της XML έχει ακυρωθεί και αντικαθίσταται από ένα ζεύγος εξωτερικών αγκίστρων.

Γενικά, οι δομές JSON είναι απλούστερες από την XML επειδή το JSON έχει λιγότερες δυνατότητες από το XML. Αλλά το JSON έχει το πλεονέκτημα ότι αντιστοιχίζεται

απευθείας σε κάποιο συνδυασμό λεξικών και λιστών. Και, δεδομένου ότι σχεδόν όλες οι γλώσσες προγραμματισμού έχουν κάτι αντίστοιχο με τα λεξικά και τις λίστες της Python, το JSON είναι μια πολύ φυσική μορφή για να ανταλλάσσουν δεδομένα δύο συνεργαζόμενα προγράμματα.

Το JSON γίνεται ταχύτατα η μορφή που επιλέγεται, για σχεδόν το σύνολο των δεδομένων που ανταλλάσσονται μεταξύ εφαρμογών, λόγω της σχετικής απλότητάς του σε σύγκριση με την XML.

Ανάλυση JSON

Κατασκευάζουμε το JSON μας με εμφωλευμένα λεξικά και λίστες όπως απαιτείται. Σε αυτό το παράδειγμα, αναπαριστούμε μια λίστα χρηστών, όπου κάθε χρήστης είναι ένα σύνολο ζευγών κλειδιών-τιμών (δηλαδή, ένα λεξικό). Έχουμε λοιπόν μια λίστα με λεξικά.

Στο παρακάτω πρόγραμμα, χρησιμοποιούμε την ενσωματωμένη βιβλιοθήκη `json` για να αναλύσουμε το JSON και να διαβάσουμε τα δεδομένα. Συγκρίνετε προσεκτικά αυτό το πρόγραμμα και τα δεδομένα εισόδου του με τα ισοδύναμα δεδομένα και τον κώδικα XML παραπάνω. Το JSON έχει λιγότερες λεπτομέρειες, επομένως πρέπει να γνωρίζουμε εκ των προτέρων ότι λαμβάνουμε μια λίστα και ότι η λίστα είναι χρηστών και κάθε χρήστης είναι ένα σύνολο ζευγών κλειδιών-τιμών. Το JSON είναι πιο συνοπτικό (πλεονέκτημα) αλλά είναι επίσης λιγότερο αυτοπεριγραφικό (ένα μειονέκτημα).

```
import json

data = '''
[
  { "id" : "001",
    "x" : "2",
    "όνομα" : "Chuck"
  } ,
  { "id" : "009",
    "x" : "7",
    "όνομα" : "Brent"
  }
]'''

info = json.loads(data)
print('Πλήθος χρηστών:', len(info))
```

```
for item in info:
    print('Όνομα', item['όνομα'])
    print('Id', item['id'])
    print('Χαρακτηριστικό', item['x'])
```

#Code: http://www.gr.py4e.com/code3/json2.py

Εάν συγκρίνετε τον κώδικα για την εξαγωγή δεδομένων από το αναλυμένο JSON και το XML, θα δείτε ότι αυτό που λαμβάνουμε από το `json.loads()` είναι μια λίστα Python, την οποία διασχίζουμε με έναν βρόχο `for` και κάθε στοιχείο σε αυτήν τη λίστα είναι ένα λεξικό Python. Αφού αναλυθεί το JSON, μπορούμε να χρησιμοποιήσουμε τον τελεστή ευρετηρίου Python για να εξαγάγουμε τα διάφορα bit δεδομένων για κάθε χρήστη. Δεν χρειάζεται να χρησιμοποιήσουμε τη βιβλιοθήκη JSON για να διερευνήσουμε το αναλυμένο JSON, καθώς τα δεδομένα που επιστρέφονται είναι απλώς εγγενείς δομές Python.

Η έξοδος αυτού του προγράμματος είναι ακριβώς η ίδια με την παραπάνω έκδοση XML.

```
Πλήθος χρηστών: 2
Όνομα Chuck
Id 001
Χαρακτηριστικό 2
Όνομα Brent
Id 009
Χαρακτηριστικό 7
```

Σε γενικές γραμμές, υπάρχει μια τάση του κλάδου να απομακρύνεται από την XML και να υιοθετεί το JSON για τις υπηρεσίες Ιστού. Επειδή το JSON είναι απλούστερο και αντιστοιχίζεται πιο άμεσα σε εγγενείς δομές δεδομένων που έχουμε ήδη στις γλώσσες προγραμματισμού, ο κώδικας ανάλυσης και εξαγωγής δεδομένων είναι συνήθως απλούστερος και πιο άμεσος όταν χρησιμοποιείται JSON. Αλλά η XML είναι πιο αυτοπεριγραφική από την JSON και έτσι υπάρχουν ορισμένες εφαρμογές όπου η XML διατηρεί ένα πλεονέκτημα. Για παράδειγμα, οι περισσότεροι επεξεργαστές κειμένου αποθηκεύουν έγγραφα εσωτερικά χρησιμοποιώντας XML αντί JSON.

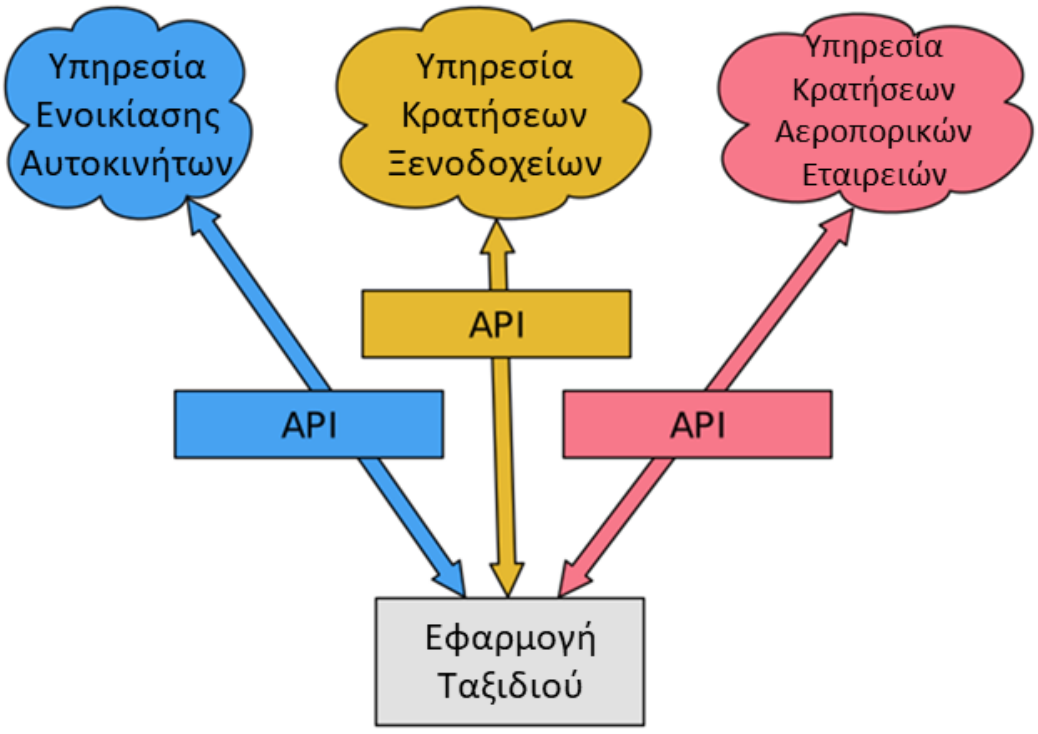
Διεπαφές προγραμματισμού εφαρμογών

Τώρα έχουμε τη δυνατότητα να ανταλλάσσουμε δεδομένα μεταξύ εφαρμογών, χρησιμοποιώντας το Πρωτόκολλο Μεταφοράς HyperText (HTTP) και έναν τρόπο να αναπαραστήσουμε σύνθετα δεδομένα, που ανταλλάσσουμε μεταξύ αυτών των

εφαρμογών χρησιμοποιώντας τη Γλώσσα Επεκτάσιμης Σήμανσης (XML) ή τη Σημειογραφία Αντικειμένων JavaScript (JSON).

Το επόμενο βήμα είναι να αρχίσουμε να ορίζουμε και να τεκμηριώνουμε "συμβάσεις" μεταξύ εφαρμογών χρησιμοποιώντας αυτές τις τεχνικές. Η γενική ονομασία για αυτές τις από εφαρμογής σε εφαρμογή συμβάσεις είναι *Διεπαφές Προγράμματος Εφαρμογής ή Application Program Interfaces* (APIs). Όταν χρησιμοποιούμε ένα API, γενικά ένα πρόγραμμα καθιστά διαθέσιμο ένα σύνολο *υπηρεσιών*, για χρήση από άλλες εφαρμογές και δημοσιεύει τα API (δηλαδή τους "κανόνες") που πρέπει να ακολουθούνται για την πρόσβαση στις υπηρεσίες που παρέχονται από αυτό.

Όταν αρχίζουμε να χτίζουμε τα προγράμματά μας, όπου η λειτουργικότητα του προγράμματός μας περιλαμβάνει πρόσβαση σε υπηρεσίες που παρέχονται από άλλα προγράμματα, ονομάζουμε την προσέγγιση *Αρχιτεκτονική προσανατολισμένη στις υπηρεσίες ή Service-oriented architecture* (SOA). Μια προσέγγιση SOA είναι μια προσέγγιση όπου η συνολική μας εφαρμογή κάνει χρήση των υπηρεσιών άλλων εφαρμογών. Μια προσέγγιση που δεν είναι SOA είναι αυτή όπου η εφαρμογή είναι μια ενιαία, αυτόνομη εφαρμογή, που περιέχει όλο τον απαραίτητο κώδικα για την υλοποίηση της εφαρμογής.



Εικόνα 13.2: Αρχιτεκτονική Προσανατολισμένη στις Υπηρεσίες (Service-oriented architecture)

Βλέπουμε πολλά παραδείγματα SOA όταν χρησιμοποιούμε τον ιστό. Μπορούμε να μεταβούμε σε έναν ιστότοπο και να κάνουμε κράτηση για αεροπορικά εισιτήρια, ξενοδοχεία και αυτοκίνητα, όλα από έναν μόνο ιστότοπο. Τα δεδομένα για τα ξενοδοχεία δεν αποθηκεύονται στους υπολογιστές της αεροπορικής εταιρείας. Αντίθετα, οι υπολογιστές της αεροπορικής εταιρείας επικοινωνούν με τις υπηρεσίες στους υπολογιστές του ξενοδοχείου, ανακτούν τα δεδομένα του ξενοδοχείου και τα παρουσιάζουν στον χρήστη. Όταν ο χρήστης συμφωνεί να κάνει μια κράτηση ξενοδοχείου, χρησιμοποιώντας τον ιστότοπο της αεροπορικής εταιρείας, ο ιστότοπος της αεροπορικής εταιρείας χρησιμοποιεί μια άλλη υπηρεσία web, στα συστήματα του ξενοδοχείου, για να υλοποιήσει πραγματικά την κράτηση. Και όταν έρθει η ώρα να χρεώσετε την πιστωτική σας κάρτα για ολόκληρη τη συναλλαγή, και πάλι άλλοι υπολογιστές εμπλέκονται στη διαδικασία.

Μια αρχιτεκτονική προσανατολισμένη στις υπηρεσίες έχει πολλά πλεονεκτήματα, όπως: (1) διατηρούμε πάντα μόνο ένα αντίγραφο δεδομένων (αυτό είναι ιδιαίτερα σημαντικό για πράγματα όπως κρατήσεις ξενοδοχείων, όπου δεν θέλουμε να κάνουμε υπερβολική δέσμευση) και (2) οι κάτοχοι των δεδομένα μπορούν να ορίσουν τους κανόνες σχετικά με τη χρήση των δεδομένων τους. Με αυτά τα πλεονεκτήματα, ένα σύστημα SOA πρέπει να είναι προσεκτικά σχεδιασμένο ώστε να έχει καλή απόδοση και να καλύπτει τις ανάγκες του χρήστη.

Όταν μια εφαρμογή διαθέτει ένα σύνολο υπηρεσιών στο API της μέσω του ιστού, την ονομάζουμε *υπηρεσίες Ιστού (web services)*.

Ασφάλεια και χρήση API

Είναι αρκετά συνηθισμένο να χρειάζεστε ένα κλειδί (key) API για να χρησιμοποιήσετε το API ενός προμηθευτή. Η γενική ιδέα είναι ότι θέλουν να γνωρίζουν ποιος χρησιμοποιεί τις υπηρεσίες τους και πόσο τις χρησιμοποιεί ο κάθε χρήστης. Ίσως έχουν δωρεάν και επί πληρωμή επίπεδα των υπηρεσιών τους ή έχουν μια πολιτική που περιορίζει τον αριθμό των αιτημάτων που μπορεί να υποβάλει ένα άτομο κατά τη διάρκεια μιας συγκεκριμένης χρονικής περιόδου.

Μερικές φορές, μόλις λάβετε το κλειδί API σας, απλώς συμπεριλάβετε το κλειδί ως μέρος των δεδομένων POST ή ίσως ως παράμετρο στη διεύθυνση URL κατά την κλήση του API.

Άλλες φορές, ο πωλητής θέλει αυξημένη διασφάλιση της προέλευσης των αιτημάτων και έτσι περιμένει από εσάς να στείλετε κρυπτογραφικά υπογεγραμμένα μηνύματα

χρησιμοποιώντας κοινόχρηστα κλειδιά και μυστικά. Μια πολύ κοινή τεχνολογία που χρησιμοποιείται για την υπογραφή αιτημάτων μέσω Διαδικτύου ονομάζεται *OAuth*. Μπορείτε να διαβάσετε περισσότερα για το πρωτόκολλο OAuth στο www.oauth.net.

Ευτυχώς, υπάρχουν πολλές βολικές και δωρεάν βιβλιοθήκες OAuth, ώστε να μπορείτε να αποφύγετε να γράψετε μια υλοποίηση OAuth, από την αρχή διαβάζοντας τις προδιαγραφές. Αυτές οι βιβλιοθήκες είναι ποικίλης πολυπλοκότητας και έχουν διαφορετικούς βαθμούς πλούτου. Ο ιστότοπος του OAuth έχει πληροφορίες σχετικά με διάφορες βιβλιοθήκες OAuth.

Γλωσσάριο

API : Application Program Interface (Διεπαφές Προγράμματος Εφαρμογή) - Ένα συμβόλαιο μεταξύ εφαρμογών που ορίζει τα πρότυπα αλληλεπίδρασης μεταξύ δύο στοιχείων εφαρμογής.

ElementTree : Μια ενσωματωμένη βιβλιοθήκη Python που χρησιμοποιείται για την ανάλυση δεδομένων XML.

JSON : JavaScript Object Notation (Σημειογραφία Αντικειμένου JavaScript) - Μια μορφή που επιτρέπει τη σήμανση δομημένων δεδομένων με βάση τη σύνταξη των αντικειμένων JavaScript.

SOA : Service-Oriented Architecture (Αρχιτεκτονική προσανατολισμένη στις υπηρεσίες) - Όταν μια εφαρμογή αποτελείται από στοιχεία συνδεδεμένα σε ένα δίκτυο.

XML : eXtensible Markup Language (Επεκτάσιμη Γλώσσα Σήμανσης) - Μια μορφή που επιτρέπει τη σήμανση δομημένων δεδομένων.

Εφαρμογή 1: Υπηρεσία ιστού γεωκωδικοποίησης (geocoding) Google

Η Google διαθέτει μια εξαιρετική υπηρεσία ιστού που μας επιτρέπει να χρησιμοποιούμε τη μεγάλη βάση δεδομένων γεωγραφικών πληροφοριών της. Μπορούμε να υποβάλουμε μια συμβολοσειρά γεωγραφικής αναζήτησης όπως "Ann Arbor, MI" στο API γεωκωδικοποίησης της και να ζητήσουμε από την Google να επιστρέψει την καλύτερη εικασία για το πού θα βρούμε τη συμβολοσειρά αναζήτησής μας σε έναν χάρτη και να μας ενημερώσει για τα γειτονικά ορόσημα.

Η υπηρεσία γεωκωδικοποίησης είναι δωρεάν αλλά περιορισμένης χρήση, επομένως δεν μπορείτε να κάνετε απεριόριστη χρήση του API σε μια εμπορική εφαρμογή. Ωστόσο, εάν

έχετε κάποια δεδομένα έρευνας, όπου ένας τελικός χρήστης έχει εισαγάγει μια τοποθεσία σε ένα πλαίσιο εισαγωγής ελεύθερης μορφής, μπορείτε να χρησιμοποιήσετε αυτό το API για να καθαρίσετε τα δεδομένα σας αρκετά καλά.

Όταν χρησιμοποιείτε ένα δωρεάν API, όπως το API γεωκωδικοποίησης της Google, θα πρέπει να δείχνετε σεβασμό στη χρήση αυτών των πόρων. Εάν πάρα πολλοί άνθρωποι κάνουν κατάχρηση της υπηρεσίας, η Google ενδέχεται να απορρίψει ή να περιορίσει σημαντικά τη δωρεάν υπηρεσία της.

Μπορείτε να διαβάσετε την ηλεκτρονική τεκμηρίωση για αυτήν την υπηρεσία, αλλά είναι αρκετά απλή και μπορείτε ακόμη και να τη δοκιμάσετε χρησιμοποιώντας ένα πρόγραμμα περιήγησης πληκτρολογώντας την ακόλουθη διεύθυνση URL στο πρόγραμμα περιήγησής σας:

http://maps.googleapis.com/maps/api/geocode/json?address=Ann+Arbor%2C+MI

Φροντίστε να αφαιρέσετε τυχόν κενά από τη διεύθυνση URL προτού την επικολλήσετε στο πρόγραμμα περιήγησής σας.

Η παρακάτω είναι μια απλή εφαρμογή, για να ζητήσει από τον χρήστη μια συμβολοσειρά αναζήτησης, να καλέσει το API γεωκωδικοποίησης της Google και να εξαγάγει πληροφορίες από το JSON που επιστράφηκε.

```
import urllib.request, urllib.parse, urllib.error
import json
import ssl

api_key = False
# Εάν διαθέτετε κλειδί API Google Places, πληκτρολογήστε το εδώ
# api_key = 'AIzaSy___IDByT70'
# https://developers.google.com/maps/documentation/geocoding/intro

if api_key is False:
    api_key = 42
    serviceurl = 'http://py4e-data.dr-chuck.net/json?'
else :
    serviceurl = 'https://maps.googleapis.com/maps/api/geocode/json?'

# Αγνόησε τα σφάλματα πιστοποιητικού SSL
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE
```

```
while True:
    address = input('Εισαγάγετε τοποθεσία: ')
    if len(address) < 1: break

    parms = dict()
    parms['address'] = address
    if api_key is not False: parms['key'] = api_key
    url = serviceurl + urllib.parse.urlencode(parms)

    print('Ανάκτηση', url)
    uh = urllib.request.urlopen(url, context=ctx)
    data = uh.read().decode()
    print('Ανακτήθηκαν', len(data), 'χαρακτήρες')

    try:
        js = json.loads(data)
    except:
        js = None

    if not js or 'status' not in js or js['status'] != 'OK':
        print('==== Αποτυχία ανάκτησης ====')
        print(data)
        continue

    print(json.dumps(js, indent=4))

    lat = js['results'][0]['geometry']['location']['lat']
    lng = js['results'][0]['geometry']['location']['lng']
    print('πλάτος', lat, 'μήκος', lng)
    location = js['results'][0]['formatted_address']
    print(location)
```

#Code: http://www.gr.py4e.com/code3/geojson.py

Το πρόγραμμα παίρνει τη συμβολοσειρά αναζήτησης και κατασκευάζει μια διεύθυνση URL με τη συμβολοσειρά αναζήτησης, ως σωστά κωδικοποιημένη παράμετρο, και στη συνέχεια χρησιμοποιεί το `urllib` για να ανακτήσει το κείμενο από το API γεωκωδικοποίησης της Google. Σε αντίθεση με μια σταθερή ιστοσελίδα, τα δεδομένα

που λαμβάνουμε εξαρτώνται από τις παραμέτρους που στέλνουμε και τα γεωγραφικά δεδομένα που είναι αποθηκευμένα στους διακομιστές της Google.

Μόλις ανακτήσουμε τα δεδομένα JSON, τα αναλύουμε με τη βιβλιοθήκη `json` και κάνουμε μερικούς ελέγχους για να βεβαιωθούμε ότι λάβαμε καλά δεδομένα και, στη συνέχεια, εξάγουμε τις πληροφορίες που αναζητούμε.

Η έξοδος του προγράμματος είναι η εξής (μερικά από τα JSON που επιστράφηκαν έχουν αφαιρεθεί):

```
$ python3 geojson.py
Εισαγάγετε τοποθεσία: Ann Arbor, MI
Ανάκτηση http://py4e-data.dr-chuck.net/json?address=Ann+Arbor%2C+MI&key=42
Ανακτήθηκαν 1736 χαρακτήρες
```

```
{
    "results": [
        {
            "address_components": [
                {
                    "long_name": "Ann Arbor",
                    "short_name": "Ann Arbor",
                    "types": [
                        "locality",
                        "political"
                    ]
                },
                {
                    "long_name": "Washtenaw County",
                    "short_name": "Washtenaw County",
                    "types": [
                        "administrative_area_level_2",
                        "political"
                    ]
                },
                {
                    "long_name": "Michigan",
                    "short_name": "MI",
                    "types": [
                        "administrative_area_level_1",
```

```
                "political"
            ]
        },
        {
            "long_name": "United States",
            "short_name": "US",
            "types": [
                "country",
                "political"
            ]
        }
    ],
    "formatted_address": "Ann Arbor, MI, USA",
    "geometry": {
        "bounds": {
            "northeast": {
                "lat": 42.3239728,
                "lng": -83.6758069
            },
            "southwest": {
                "lat": 42.222668,
                "lng": -83.799572
            }
        },
        "location": {
            "lat": 42.2808256,
            "lng": -83.7430378
        },
        "location_type": "APPROXIMATE",
        "viewport": {
            "northeast": {
                "lat": 42.3239728,
                "lng": -83.6758069
            },
            "southwest": {
                "lat": 42.222668,
                "lng": -83.799572
            }
        }
```

```
            },
            "place_id": "ChIJMx9D1A2wPIgR4rXIhkb5Cds",
            "types": [
                "locality",
                "political"
            ]
        }
    ],
    "status": "OK"
}
πλάτος 42.2808256 μήκος -83.7430378
Ann Arbor, MI, USA
```

```
Εισαγάγετε τοποθεσία:
```

Μπορείτε να κάνετε λήψη του www.gr.py4e.com/code3/geoxml.py για να εξερευνήσετε την παραλλαγή XML του API γεωκωδικοποίησης Google.

Άσκηση 1: Αλλάξτε είτε το geojson.py είτε το geoxml.py για να εκτυπώσετε τον κωδικό χώρας δύο χαρακτήρων από τα δεδομένα που ανακτήθηκαν. Προσθέστε έλεγχο σφαλμάτων, ώστε το πρόγραμμά σας να μην ανιχνεύει εάν ο κωδικός χώρας δεν υπάρχει. Μόλις το θέσετε σε λειτουργία, αναζητήστε "Atlantic Ocean" και βεβαιωθείτε ότι μπορεί να χειριστεί τοποθεσίες που δεν βρίσκονται σε καμία χώρα.

Εφαρμογή 2: Twitter

Καθώς το Twitter API γινόταν όλο και πιο πολύτιμο, το Twitter μετατράπηκε από ένα ανοιχτό και δημόσιο API σε ένα API που απαιτούσε τη χρήση υπογραφών OAuth σε κάθε αίτημα API.

Για αυτό το επόμενο δείγμα προγράμματος, πραγματοποιήστε λήψη των αρχείων *twurl.py*, *hidden.py*, *oauth.py* και *twitter1.py* από www.gr.py4e.com/code και αποθηκεύστε τα όλα σε ένα φάκελο στον υπολογιστή σας.

Για να χρησιμοποιήσετε αυτά τα προγράμματα θα χρειαστεί να έχετε λογαριασμό Twitter και να εξουσιοδοτήσετε τον κώδικα Python ως εφαρμογή, να ένα κλειδί (key), ένα μυστικό (secret), ένα διακριτικό (token) και ένα μυστικό διακριτικού (token secret).

Θα επεξεργαστείτε το αρχείο *hidden.py* και θα προσθέσετε αυτές τις τέσσερις συμβολοσειρές στις κατάλληλες μεταβλητές του αρχείου:

```
# Κρατήστε αυτό το αρχείο ξεχωριστά

# https://apps.twitter.com/
# Δημιουργήστε νέο App και πάρτε τις τέσσερις συμβολοσειρές

def oauth():
    return {"consumer_key": "h7Lu...Ng",
            "consumer_secret": "dNKenAC3New...mmn7Q",
            "token_key": "10185562-eibxCp9n2...P4GEQQOSGI",
            "token_secret": "H0ycCFemmC4wyf1...qoIpBo"}
```

#Code: http://www.gr.py4e.com/code3/hidden.py

Η πρόσβαση στην υπηρεσία ιστού Twitter γίνεται χρησιμοποιώντας μια διεύθυνση URL όπως αυτή:

https://api.twitter.com/1.1/statuses/user_timeline.json

Αλλά μόλις προστεθούν όλες οι πληροφορίες ασφαλείας, η διεύθυνση URL θα μοιάζει περισσότερο με:

```
https://api.twitter.com/1.1/statuses/user_timeline.json?count=2
&oauth_version=1.0&oauth_token=101...SGI&screen_name=drchuck
&oauth_nonce=09239679&oauth_timestamp=1380395644
&oauth_signature=rLK...BoD&oauth_consumer_key=h7Lu...GNg
&oauth_signature_method=HMAC-SHA1
```

Μπορείτε να διαβάσετε την προδιαγραφή OAuth, εάν θέλετε να μάθετε περισσότερα σχετικά με τη σημασία των διαφόρων παραμέτρων που προστίθενται για την κάλυψη των απαιτήσεων ασφαλείας του OAuth.

Για τα προγράμματα που εκτελούμε με το Twitter, αποκρύπτουμε όλη την πολυπλοκότητα στα αρχεία *oauth.py* και *twurl.py*. Απλώς ορίζουμε τα μυστικά στο *hidden.py* και μετά στέλνουμε το επιθυμητό URL στη συνάρτηση *twurl.augment()* και ο κώδικας της βιβλιοθήκης προσθέτει όλες τις απαραίτητες παραμέτρους στη διεύθυνση URL για εμάς.

Αυτό το πρόγραμμα ανακτά το χρονοδιάγραμμα (timeline) για έναν συγκεκριμένο χρήστη του Twitter και μας το επιστρέφει σε μορφή JSON σε μια συμβολοσειρά. Εκτυπώνουμε απλώς τους πρώτους 250 χαρακτήρες της συμβολοσειράς:

```
import urllib.request, urllib.parse, urllib.error
import twurl
import ssl

# https://apps.twitter.com/
# Create App and get the four strings, put them in hidden.py

TWITTER_URL = 'https://api.twitter.com/1.1/statuses/user_timeline.json'

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
    print('')
    acct = input('Enter Twitter Account:')
    if (len(acct) < 1): break
    url = twurl.augment(TWITTER_URL,
                        {'screen_name': acct, 'count': '2'})
    print('Retrieving', url)
    connection = urllib.request.urlopen(url, context=ctx)
    data = connection.read().decode()
    print(data[:250])
    headers = dict(connection.getheaders())
    # print headers
    print('Remaining', headers['x-rate-limit-remaining'])
```

#Code: http://www.gr.py4e.com/code3/twitter1.py

Όταν το πρόγραμμα εκτελείται, παράγει την ακόλουθη έξοδο:

```
Enter Twitter Account: drchuck
Retrieving https://api.twitter.com/1.1/ ...
[{"created_at":"Sat Sep 28 17:30:25 +0000 2013","
id":384007200990982144,"id_str":"384007200990982144",
"text":"RT @fixpert: See how the Dutch handle traffic
intersections: http:\/\/t.co\/tIiVWtEhj4\n#brilliant",
"source":"web","truncated":false,"in_rep
Remaining 178
```

```
Enter Twitter Account:fixpert
Retrieving https://api.twitter.com/1.1/ ...
[{"created_at":"Sat Sep 28 18:03:56 +0000 2013",
"id":384015634108919808,"id_str":"384015634108919808",
"text":"3 months after my freak bocce ball accident,
my wedding ring fits again! :)\n\nhttps:\/\/t.co\/2XmHPx7kgX",
"source":"web","truncated":false,
Remaining 177

Enter Twitter Account:
```

Μαζί με τα επιστρεφόμενα δεδομένα χρονοδιαγράμματος, το Twitter επιστρέφει επίσης μεταδεδομένα σχετικά με το αίτημα στις κεφαλίδες απόκρισης HTTP. Ειδικότερα, μια κεφαλίδα, `x-rate-limit-remaining`, μας ενημερώνει πόσα ακόμη αιτήματα μπορούμε να υποβάλουμε προτού αποκλειστούμε για ένα σύντομο χρονικό διάστημα. Μπορείτε να δείτε ότι οι ανακτήσεις που απομένουν μειώνονται κατά μία κάθε φορά που υποβάλλουμε ένα αίτημα στο API.

Στο παρακάτω παράδειγμα, ανακτούμε τους φίλους Twitter ενός χρήστη, αναλύουμε το JSON που επιστράφηκε και εξάγουμε ορισμένες από τις πληροφορίες σχετικά με τους φίλους. Επίσης, απορρίπτουμε το JSON μετά την ανάλυση και το "εκτυπώνουμε όμορφα" με μια εσοχή τεσσάρων χαρακτήρων για να μας επιτρέψει να διαπεράσουμε τα δεδομένα όταν θέλουμε να εξαγάγουμε περισσότερα πεδία.

```
import urllib.request, urllib.parse, urllib.error
import twurl
import json
import ssl

# https://apps.twitter.com/
# Create App and get the four strings, put them in hidden.py

TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
```

```
    print('')
    acct = input('Enter Twitter Account:')
    if (len(acct) < 1): break
    url = twurl.augment(TWITTER_URL,
                        {'screen_name': acct, 'count': '5'})
    print('Retrieving', url)
    connection = urllib.request.urlopen(url, context=ctx)
    data = connection.read().decode()

    js = json.loads(data)
    print(json.dumps(js, indent=2))

    headers = dict(connection.getheaders())
    print('Remaining', headers['x-rate-limit-remaining'])

    for u in js['users']:
        print(u['screen_name'])
        if 'status' not in u:
            print('   * No status found')
            continue
        s = u['status']['text']
        print('  ', s[:50])
```

#Code: http://www.gr.py4e.com/code3/twitter2.py

Εφόσον το JSON γίνεται ένα σύνολο από ένθετες λίστες και λεξικά Python, μπορούμε να χρησιμοποιήσουμε έναν συνδυασμό της λειτουργίας ευρετηρίου και των βρόχων for για να διατρέξουμε στις επιστρεφόμενες δομές δεδομένων με πολύ λίγο κώδικα Python.

Η έξοδος του προγράμματος έχει ως εξής (ορισμένα από τα στοιχεία δεδομένων συντομεύονται για να χωρούν στη σελίδα):

```
Enter Twitter Account: drchuck
Retrieving https://api.twitter.com/1.1/friends ...
Remaining 14
```


```
{
  "next_cursor": 1444171224491980205,
  "users": [
    {
      "id": 662433,
```

```
      "followers_count": 28725,
      "status": {
        "text": "@jazzychad I just bought one .__.",
        "created_at": "Fri Sep 20 08:36:34 +0000 2013",
        "retweeted": false,
      },
      "location": "San Francisco, California",
      "screen_name": "leahculver",
      "name": "Leah Culver",
    },
    {
      "id": 40426722,
      "followers_count": 2635,
      "status": {
        "text": "RT @WSJ: Big employers like Google ...",
        "created_at": "Sat Sep 28 19:36:37 +0000 2013",
      },
      "location": "Victoria Canada",
      "screen_name": "_valeriei",
      "name": "Valerie Irvine",
    }
  ],
 "next_cursor_str": "1444171224491980205"
}
```


```
leahculver
   @jazzychad I just bought one .__.
_valeriei
   RT @WSJ: Big employers like Google, AT&amp;amp;T are h
ericbollens
   RT @lukew: sneak peek: my LONG take on the good &a
halherzog
   Learning Objects is 10. We had a cake with the LO,
scweeker
   @DeviceLabDC love it! Now where so I get that "etc

Enter Twitter Account:
```

Το τελευταίο κομμάτι της εξόδου είναι αυτό όπου βλέπουμε τον βρόχο for να διαβάζει τους πέντε πιο πρόσφατους "φίλους" του λογαριασμού Twitter *@drchuck* και να εκτυπώνει το πιο πρόσφατο status του κάθε φίλου. Υπάρχουν πολλά περισσότερα διαθέσιμα δεδομένα στο JSON που επιστράφηκε. Αν κοιτάξετε στην έξοδο του προγράμματος, μπορείτε επίσης να δείτε ότι το "βρείτε τους φίλους (find the friends)" ενός συγκεκριμένου λογαριασμού έχει διαφορετικό όριο χρέωσης από τον αριθμό των ερωτημάτων χρονοδιαγράμματος, που επιτρέπεται να εκτελούμε ανά χρονική περίοδο.

Αυτά τα ασφαλή κλειδιά API επιτρέπουν στο Twitter να είναι σίγουρο ότι γνωρίζει ποιος χρησιμοποιεί το API και τα δεδομένα του και σε ποιο επίπεδο. Η προσέγγιση περιορισμού πρόσβασης μάς επιτρέπει να κάνουμε απλές, ανακτήσεις προσωπικών δεδομένων, αλλά δεν μας επιτρέπει να δημιουργήσουμε ένα προϊόν που αντλεί δεδομένα από το API τους, εκατομμύρια φορές την ημέρα.

Κεφάλαιο 14

Αντικειμενοστραφής προγραμματισμός – object oriented

Διαχείριση μεγαλύτερων προγραμμάτων

Στην αρχή αυτού του βιβλίου, καταλήξαμε σε τέσσερα βασικά μοτίβα προγραμματισμού τα οποία χρησιμοποιούμε για την κατασκευή προγραμμάτων:

- Δομή ακολουθίας
- Δομή επιλογής (εντολές if)
- Δομή επανάληψης (βρόχοι)
- Αποθήκευση και επαναχρησιμοποίηση (συναρτήσεις)

Σε επόμενα κεφάλαια, εξερευνήσαμε απλές μεταβλητές καθώς και δομές δεδομένων συλλογής όπως λίστες, πλειάδες και λεξικά.

Καθώς κατασκευάζουμε προγράμματα, σχεδιάζουμε δομές δεδομένων και γράφουμε κώδικα για να χειριστούμε αυτές τις δομές δεδομένων. Υπάρχουν πολλοί τρόποι για να γράψετε προγράμματα και μέχρι τώρα, πιθανότατα έχετε γράψει κάποια προγράμματα που "δεν ήταν τόσο κομψά" και άλλα προγράμματα που ήταν "πιο κομψά". Παρόλο που τα προγράμματά σας μπορεί να είναι μικρά, αρχίζετε να βλέπετε πως η σύνταξη κώδικα εμπεριέχει και λίγη τέχνη και αισθητική.

Καθώς τα προγράμματα φτάνουν το μήκος εκατομμυρίων γραμμών, γίνεται όλο και πιο σημαντικό να γράφουμε κώδικα που είναι εύκολο να κατανοηθεί. Εάν εργάζεστε σε ένα πρόγραμμα εκατομμυρίων γραμμών, δεν μπορείτε ποτέ να κρατήσετε ολόκληρο το πρόγραμμα στο μυαλό σας ταυτόχρονα. Χρειαζόμαστε τρόπους για να χωρίσουμε μεγάλα προγράμματα σε πολλά μικρότερα κομμάτια, ώστε να έχουμε λιγότερα να εξετάσουμε κατά την επίλυση ενός προβλήματος, τη διόρθωση ενός σφάλματος ή την προσθήκη μιας νέας δυνατότητας.

Κατά κάποιον τρόπο, ο αντικειμενοστραφής προγραμματισμός είναι ένας τρόπος για να τακτοποιήσετε τον κώδικά σας έτσι ώστε να μπορείτε να εστιάσετε σε 50 γραμμές του κώδικα και να τον κατανοήσετε, ενώ αγνοείτε τις άλλες 999.950 γραμμές, προς το παρόν.

Ξεκινώντας

Όπως και σε πολλές άλλες πτυχές του προγραμματισμού, είναι απαραίτητο να μάθετε τις έννοιες του αντικειμενοστρεφούς προγραμματισμού προτού μπορέσετε να τις χρησιμοποιήσετε αποτελεσματικά. Θα πρέπει να προσεγγίσετε αυτό το κεφάλαιο ως έναν τρόπο για να μάθετε ορισμένους όρους και έννοιες και να επεξεργαστείτε μερικά απλά παραδείγματα, για να θέσετε τα θεμέλια της μελλοντικής μάθησης.

Το βασικό που πρέπει να αποκομίσετε από αυτό το κεφαλαίο είναι να αποκτήσετε μια στοιχειώδη κατανόηση του πώς κατασκευάζονται τα αντικείμενα και πώς λειτουργούν και κυρίως πώς χρησιμοποιούμε τις δυνατότητες των αντικειμένων, που μας παρέχονται από την Python και τις βιβλιοθήκες της Python.

Χρήση αντικειμένων

Όπως αποδεικνύεται, χρησιμοποιούσαμε αντικείμενα σε αυτό το βιβλίο. Η Python μας παρέχει πολλά ενσωματωμένα αντικείμενα. Εδώ είναι ένας απλός κώδικας όπου οι πρώτες γραμμές θα πρέπει να σας φαίνονται πολύ απλές και φυσικές.

```
stuff = list()
stuff.append('python')
stuff.append('chuck')
stuff.sort()
print (stuff[0])
print (stuff.__getitem__(0))
print (list.__getitem__(stuff,0))
```

#Code: http://www.gr.py4e.com/code3/party1.py

Αντί να εστιάσουμε στο τι επιτυγχάνουν αυτές οι γραμμές, ας δούμε τι πραγματικά συμβαίνει, από την άποψη του αντικειμενοστρεφούς προγραμματισμού. Μην ανησυχείτε εάν οι παρακάτω παράγραφοι δεν έχουν νόημα την πρώτη φορά που τις διαβάζετε, επειδή δεν έχουμε ορίσει ακόμη όλους αυτούς τους όρους.

Η πρώτη γραμμή *κατασκευάζει* ένα αντικείμενο τύπου `list` (λίστα), η δεύτερη και η τρίτη γραμμή *καλούν* τη *μέθοδο* `append()`, η τέταρτη γραμμή καλεί τη μέθοδο `sort()` και η πέμπτη γραμμή *ανακτά* το στοιχείο στη θέση 0.

Η έκτη γραμμή καλεί τη μέθοδο `__getitem__()` στη λίστα `stuff` με παράμετρο μηδέν.

```
print (stuff.__getitem__(0))
```

Η έβδομη γραμμή είναι ένας ακόμη πιο αναλυτικός τρόπος ανάκτησης του 0ου στοιχείου στη λίστα.

```
print (list.__getitem__(stuff,0))
```

Σε αυτόν τον κώδικα, καλούμε τη μέθοδο `__getitem__` στην κλάση `list` και *περνάμε* τη λίστα και το στοιχείο, που θέλουμε να ανακτηθεί από τη λίστα, ως παραμέτρους.

Οι τρεις τελευταίες γραμμές του προγράμματος είναι ισοδύναμες, αλλά είναι πιο βολικό να χρησιμοποιήσετε απλώς τη σύνταξη της αγκύλης, για να ζητήσετε ένα στοιχείο σε μια συγκεκριμένη θέση μιας λίστας.

Μπορούμε να ρίξουμε μια ματιά στις δυνατότητες ενός αντικειμένου κοιτάζοντας την έξοδο της συνάρτησης `dir()`:

```
>>> stuff = list()
>>> dir(stuff)
['__add__', '__class__', '__contains__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__reversed__', '__rmul__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__',
'append', 'clear', 'copy', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

Το υπόλοιπο αυτού του κεφαλαίου θα ορίσει όλους τους παραπάνω όρους, επομένως φροντίστε να επιστρέψετε αφού ολοκληρώσετε το κεφάλαιο και να διαβάσετε ξανά τις παραπάνω παραγράφους για να ελέγξετε την κατανόησή σας.

Ξεκινώντας με προγράμματα

Ένα πρόγραμμα στην πιο βασική του μορφή παίρνει κάποια είσοδο, κάνει κάποια επεξεργασία και παράγει κάποια έξοδο. Το πρόγραμμα μετατροπής ανελκυστήρα αποτελεί ένα πολύ σύντομο, αλλά πλήρες πρόγραμμα, που δείχνει και τα τρία αυτά βήματα.

```
usf = input('Enter the US Floor Number: ')
wf = int(usf) - 1
print('Non-US Floor Number is',wf)
```

#Code: http://www.gr.py4e.com/code3/elev.py

Αν σκεφτούμε λίγο περισσότερο αυτό το πρόγραμμα, υπάρχει ο "έξω κόσμος" και το πρόγραμμα. Οι πτυχές εισόδου και εξόδου είναι εκεί όπου το πρόγραμμα αλληλεπιδρά με τον έξω κόσμο. Μέσα στο πρόγραμμα έχουμε κώδικα και δεδομένα για να ολοκληρώσουμε την εργασία, που έχει σχεδιαστεί για να λύσει το πρόγραμμα.

Εικόνα 14.1: Ένα Πρόγραμμα

Ένας τρόπος να αντιληφθούμε τον αντικειμενοστραφή προγραμματισμό είναι ότι διαχωρίζει το πρόγραμμά μας σε πολλαπλές "ζώνες". Κάθε ζώνη περιέχει κάποιο κώδικα και δεδομένα (όπως ένα πρόγραμμα) και έχει καλά καθορισμένες αλληλεπιδράσεις με τον έξω κόσμο και τις άλλες ζώνες εντός του προγράμματος.

Αν ξανά κοιτάξουμε την εφαρμογή εξαγωγής συνδέσμων στην οποία χρησιμοποιήσαμε τη βιβλιοθήκη BeautifulSoup, μπορούμε να δούμε ένα πρόγραμμα που κατασκευάζεται συνδέοντας διαφορετικά αντικείμενα μεταξύ τους για να ολοκληρώσει μια εργασία:

```
# Για να το εκτελέσετε, κάντε λήψη του αρχείου zip BeautifulSoup
# από  http://www.py4e.com/code3/bs4.zip
# και αποσυμπιέστε το στον ίδιο κατάλογο με αυτό το αρχείο

import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl

# Αγνόηση των σφαλμάτων πιστοποιητικού SSL
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Εισάγετε - ')
html = urllib.request.urlopen(url, context=ctx).read()
```

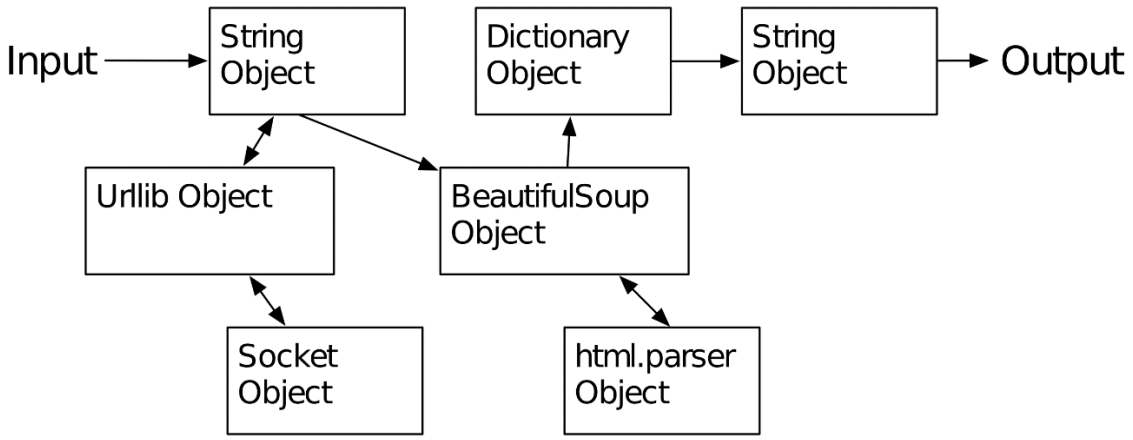
```
soup = BeautifulSoup(html, 'html.parser')

# Ανάκτηση όλων των ετικετών αγκύρωσης
tags = soup('a')
for tag in tags:
    print(tag.get('href', None))
```

#Code: http://www.gr.py4e.com/code3/urllinks.py

Διαβάζουμε τη διεύθυνση URL σε μια συμβολοσειρά και στη συνέχεια τη περνάμε στο "urllib" για να ανακτήσουμε τα δεδομένα από τον ιστό. Η βιβλιοθήκη `urllib` χρησιμοποιεί τη βιβλιοθήκη `socket` για να πραγματοποιήσει την σύνδεση δικτύου, για την ανάκτηση των δεδομένων. Παίρνουμε τη συμβολοσειρά που επιστρέφει το `urllib` και τη δίνουμε στη BeautifulSoup για ανάλυση. Η BeautifulSoup χρησιμοποιεί το αντικείμενο `html.parser`[1] και επιστρέφει ένα αντικείμενο. Καλούμε τη μέθοδο `tags()` στο επιστρεφόμενο αντικείμενο, που επιστρέφει ένα λεξικό αντικειμένων ετικετών. Με βρόχο διατρέχουμε τις ετικέτες και καλούμε τη μέθοδο `get()` για κάθε ετικέτα, για να εκτυπώσουμε το χαρακτηριστικό `href`.

Μπορούμε να σχεδιάσουμε μια εικόνα αυτού του προγράμματος και πώς συνεργάζονται τα αντικείμενα.



Εικόνα 14.2: Ένα πρόγραμμα ως Δίκτυο Αντικειμένων

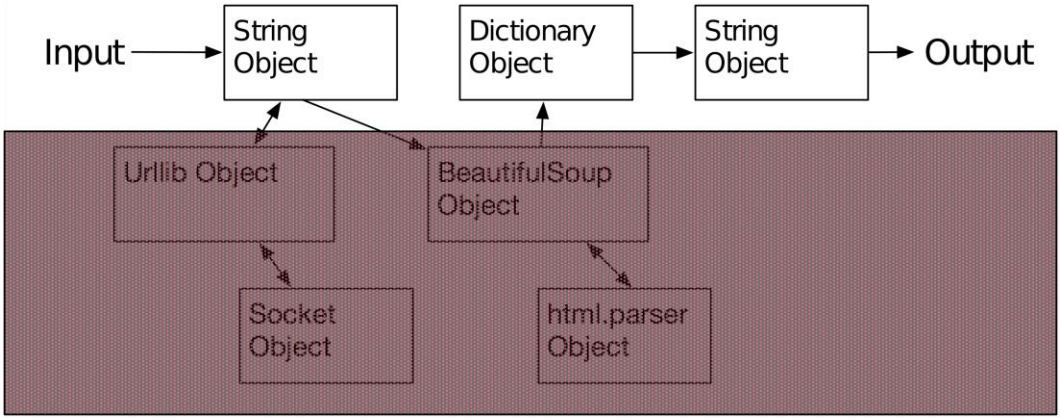
Το κλειδί εδώ δεν είναι να κατανοήσουμε τέλεια πώς λειτουργεί αυτό το πρόγραμμα, αλλά και να δούμε πώς χτίζουμε ένα δίκτυο αλληλεπιδρώντων αντικειμένων και ενορχηστρώνουμε την κίνηση των πληροφοριών μεταξύ των αντικειμένων για να δημιουργήσουμε ένα πρόγραμμα. Είναι επίσης σημαντικό να σημειωθεί ότι όταν

[1] https://docs.python.org/3/library/html.parser.html

κοιτάξατε αυτό το πρόγραμμα αρκετά κεφάλαια πίσω, μπορούσατε να καταλάβετε πλήρως τι συνέβαινε στο πρόγραμμα χωρίς καν να συνειδητοποιήσετε ότι το πρόγραμμα "ενορχηστρώνει την κίνηση των δεδομένων μεταξύ αντικειμένων". Ήταν απλώς γραμμές κώδικα που έκαναν τη δουλειά τους.

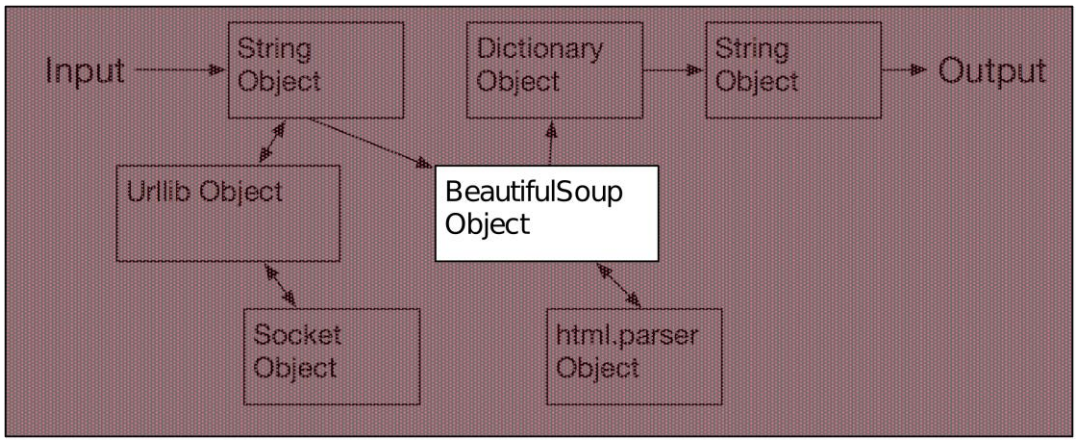
Υποδιαιρώντας ένα πρόβλημα

Ένα από τα πλεονεκτήματα της αντικειμενοστρεφούς προσέγγισης είναι ότι μπορεί να κρύψει την πολυπλοκότητα. Για παράδειγμα, ενώ πρέπει να γνωρίζουμε πώς να χρησιμοποιήσουμε τον κώδικα `urllib` και BeautifulSoup, δεν χρειάζεται να γνωρίζουμε πώς λειτουργούν αυτές οι βιβλιοθήκες εσωτερικά. Αυτό μας επιτρέπει να εστιάσουμε στο μέρος του προβλήματος που πρέπει να λύσουμε και να αγνοήσουμε τα άλλα μέρη του προγράμματος.



Εικόνα 14.3:Παράβλεψη Λεπτομερειών Κατά τη Χρήση Αντικειμένου

Αυτή η ικανότητα να εστιάζουμε αποκλειστικά στο μέρος ενός προγράμματος που μας ενδιαφέρει και να αγνοούμε τα υπόλοιπα είναι επίσης χρήσιμη στους προγραμματιστές των αντικειμένων που χρησιμοποιούμε. Για παράδειγμα, οι προγραμματιστές που ανέπτυξαν το BeautifulSoup δεν χρειαζόταν να γνωρίζουν ή να ενδιαφερθούν για το πώς ανακτούμε τη σελίδα HTML, ποια μέρη θέλουμε να διαβάσουμε ή τι σκοπεύουμε να κάνουμε με τα δεδομένα που εξάγουμε από την ιστοσελίδα.



Εικόνα 14.4: Αγνοώντας τις Λεπτομέρειες Κατά την Κατασκευή ενός Αντικειμένου

Το πρώτο μας αντικείμενο Python

Σε ένα στοιχειώδες επίπεδο, ένα αντικείμενο είναι απλώς κάποιος κώδικας συν κάποιες δομές δεδομένων, που είναι μικρότερες από ένα ολοκληρωμένο πρόγραμμα. Ο ορισμός μιας συνάρτησης μας επιτρέπει να αποθηκεύσουμε ένα κομμάτι κώδικα, να του δώσουμε ένα όνομα και στη συνέχεια να καλέσουμε αυτόν τον κωδικό, χρησιμοποιώντας το όνομα της συνάρτησης.

Ένα αντικείμενο μπορεί να περιέχει έναν αριθμό συναρτήσεων (τις οποίες ονομάζουμε *μεθόδους*) καθώς και δεδομένα, που χρησιμοποιούνται από αυτές τις συναρτήσεις. Καλούμε *χαρακτηριστικά* τα στοιχεία δεδομένων, που αποτελούν μέρος του αντικειμένου.

Χρησιμοποιούμε τη δεσμευμένη λέξη `class` για να ορίσουμε τα δεδομένα και τον κώδικα που θα αποτελέσουν κάθε ένα από τα αντικείμενα. Η δεσμευμένη λέξη class ακολουθείτε από το όνομα της κλάσης και οριοθετεί ένα μπλοκ κώδικα, με εσοχή, όπου συμπεριλαμβάνουμε τα χαρακτηριστικά (δεδομένα) και τις μεθόδους (κώδικας).

```
class PartyAnimal:
   x = 0

   def party(self) :
     self.x = self.x + 1
     print("So far",self.x)

an = PartyAnimal()
an.party()
```

```
an.party()
an.party()
PartyAnimal.party(an)
```

#Code: http://www.gr.py4e.com/code3/party2.py

Στο παραπάνω παράδειγμα, κάθε μέθοδος μοιάζει με μια συνάρτηση, που ξεκινά με τη δεσμευμένη λέξη `def` και αποτελείται από ένα μπλοκ κώδικα με εσοχή. Αυτό το αντικείμενο έχει ένα χαρακτηριστικό (`x`) και μία μέθοδο (`party`). Οι μέθοδοι έχουν μια ειδική πρώτη παράμετρο που ονομάζουμε κατά σύμβαση `self`.

Ακριβώς όπως η δεσμευμένη λέξη `def` δεν προκαλεί την εκτέλεση του κώδικα συνάρτησης, έτσι και η δεσμευμένη λέξη `class` δεν δημιουργεί κάποιο αντικείμενο. Αντίθετα, η δεσμευμένη λέξη `class` ορίζει ένα πρότυπο, που υποδεικνύει ποια δεδομένα και κώδικας θα περιέχονται σε κάθε αντικείμενο τύπου `PartyAnimal`. Η κλάση είναι σαν κόφτης (κουπάτ) μπισκότων και τα αντικείμενα που δημιουργούνται χρησιμοποιώντας την κλάση είναι τα μπισκότα[1]. Δεν βάζετε γλάσο στον κόφτη μπισκότων, βάζετε γλάσο στα μπισκότα και μπορείτε να βάλετε διαφορετικό γλάσο σε κάθε μπισκότο.

Εικόνα 14.5: Μια Κλάση και Δύο Αντικείμενα

Εάν διατρέξουμε αυτό το δείγμα προγράμματος, εντοπίζουμε την πρώτη εκτελέσιμη γραμμή κώδικα:

```
an = PartyAnimal()
```

[1] Cookie image copyright CC-BY
https://www.flickr.com/photos/dinnerseries/23570475099

Εδώ δίνουμε εντολή στην Python να κατασκευάσει (δηλαδή, να δημιουργήσει) ένα *αντικείμενο* ή *στιγμιότυπο* της κλάσης `PartyAnimal`. Μοιάζει με μια κλήση συνάρτησης, προς την ίδια την κλάση. Η Python κατασκευάζει το αντικείμενο με τα σωστά δεδομένα και μεθόδους και επιστρέφει το αντικείμενο το οποίο στη συνέχεια εκχωρείται στη μεταβλητή `an`. Κατά κάποιο τρόπο αυτό μοιάζει αρκετά με την ακόλουθη γραμμή που χρησιμοποιούσαμε όλο το προηγούμενο διάστημα:

```
πλήθη = dict()
```

Εδώ δίνουμε εντολή στην Python να κατασκευάσει ένα αντικείμενο, χρησιμοποιώντας το πρότυπο `dict` (που υπάρχει ήδη στην Python), να επιστρέψει το στιγμιότυπο του λεξικού και να το αναθέσει στη μεταβλητή πλήθη.

Όταν η κλάση `PartyAnimal` χρησιμοποιήθηκε για την κατασκευή ενός αντικειμένου, η μεταβλητή `an` χρησιμοποιήθηκε για να δείξει σε αυτό το αντικείμενο. Χρησιμοποιούμε το `an` για πρόσβαση στον κώδικα και τα δεδομένα του συγκεκριμένου στιγμιότυπου της κλάσης `PartyAnimal`.

Κάθε αντικείμενο/στιγμιότυπο Partyanimal περιέχει μέσα του μια μεταβλητή `x` και μια μέθοδο/συνάρτηση με το όνομα `party`. Καλούμε τη μέθοδο `party` σε αυτή τη γραμμή:

```
an.party()
```

Όταν καλείται η μέθοδος `party`, η πρώτη παράμετρος (την οποία ονομάζουμε κατά σύμβαση `self`) δείχνει τη συγκεκριμένη περίπτωση του αντικειμένου PartyAnimal για την οποία καλείται το `party`. Στη μέθοδο `party`, βλέπουμε τη γραμμή:

```
self.x = self.x + 1
```

Αυτή η σύνταξη, που χρησιμοποιεί τον τελεστή *dot* λέει 'το x μέσα στο self'. Κάθε φορά που καλείται η `party()`, η εσωτερική τιμή του `x` αυξάνεται κατά 1 και η τιμή εκτυπώνεται.

Η ακόλουθη γραμμή είναι ένας άλλος τρόπος για να καλέσετε τη μέθοδο `party` στο αντικείμενο `an`:

```
PartyAnimal.party(an)
```

Σε αυτήν την παραλλαγή, έχουμε πρόσβαση στον κώδικα μέσα από την κλάση και μεταβιβάζουμε ρητά τον δείκτη αντικειμένου `an` ως πρώτη παράμετρο (δηλαδή, `self` στη μέθοδο). Μπορείτε να σκεφτείτε το `an.party()` ως συντομογραφία για την παραπάνω γραμμή.

Όταν το πρόγραμμα εκτελείται, παράγει την ακόλουθη έξοδο:

```
So far 1
So far 2
So far 3
So far 4
```

Το αντικείμενο κατασκευάζεται και η μέθοδος party καλείται τέσσερις φορές, αυξάνοντας και εκτυπώνοντας την τιμή του x μέσα στο αντικείμενο an.

Οι κλάσεις ως τύποι

Όπως είδαμε, στην Python όλες οι μεταβλητές έχουν έναν τύπο. Μπορούμε να χρησιμοποιήσουμε την ενσωματωμένη συνάρτηση dir για να εξετάσουμε τις δυνατότητες μιας μεταβλητής. Μπορούμε επίσης να χρησιμοποιήσουμε τις type και dir με τις κλάσεις που δημιουργούμε.

```
class PartyAnimal:
   x = 0

   def party(self) :
     self.x = self.x + 1
     print("So far",self.x)

an = PartyAnimal()
print ("Type", type(an))
print ("Dir ", dir(an))
print ("Type", type(an.x))
print ("Type", type(an.party))
```

#Code: http://www.gr.py4e.com/code3/party3.py

Όταν εκτελείται αυτό το πρόγραμμα, παράγει την ακόλουθη έξοδο:

```
Type <class '__main__.PartyAnimal'>
Dir  ['__class__', '__delattr__', ...
'__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'party', 'x']
Type <class 'int'>
Type <class 'method'>
```

Μπορείτε να δείτε ότι χρησιμοποιώντας τη δεσμευμένη λέξη `class`, δημιουργήσαμε έναν νέο τύπο. Από την έξοδο της `dir`, μπορείτε να δείτε ότι και το χαρακτηριστικό ακεραίου `x` και η μέθοδος `party` είναι διαθέσιμα στο αντικείμενο.

Κύκλος ζωής αντικειμένου

Στα προηγούμενα παραδείγματα, ορίζουμε μια κλάση (πρότυπο), χρησιμοποιούμε αυτήν την κλάση για να δημιουργήσουμε ένα στιγμιότυπο αυτής της κλάσης (αντικείμενο) και, στη συνέχεια, χρησιμοποιούμε το στιγμιότυπο. Όταν τελειώσει το πρόγραμμα, όλες οι μεταβλητές καταστρέφονται. Συνήθως, δεν σκεφτόμαστε πολύ τη δημιουργία και την καταστροφή μεταβλητών, αλλά συχνά, καθώς τα αντικείμενά μας γίνονται πιο περίπλοκα, πρέπει να ορίσουμε κάποια ενέργεια μέσα στο αντικείμενο, για να ρυθμίσουμε την κατασκευή του αντικειμένου και πιθανώς να καθαρίσουμε τα πράγματα όταν το αντικείμενο καταστρέφεται.

Εάν θέλουμε το αντικείμενό μας "προετοιμασμένο" για αυτές τις στιγμές κατασκευής και καταστροφής, προσθέτουμε στο αντικείμενο μας ειδικές μεθόδους:

```
class PartyAnimal:
   x = 0

   def __init__(self):
     print('I am constructed')

   def party(self) :
     self.x = self.x + 1
     print('So far',self.x)

   def __del__(self):
     print('I am destructed', self.x)

an = PartyAnimal()
an.party()
an.party()
an = 42
print('an contains',an)
```

#Code: http://www.gr.py4e.com/code3/party4.py

Όταν εκτελείται αυτό το πρόγραμμα, παράγει την ακόλουθη έξοδο:

```
I am constructed
So far 1
So far 2
I am destructed 2
an contains 42
```

Καθώς η Python κατασκευάζει το αντικείμενό μας, καλεί τη μέθοδο `__init__` για να μας δώσει την ευκαιρία να ορίσουμε κάποιες προεπιλεγμένες ή αρχικές τιμές για το αντικείμενο. Όταν η Python συναντά τη γραμμή:

```
an = 42
```

Στην πραγματικότητα "πετάει το αντικείμενό μας", ώστε να μπορέσει να χρησιμοποιήσει ξανά τη μεταβλητή an για να αποθηκεύσει την τιμή 42. Ακριβώς τη στιγμή που το αντικείμενό μας an "καταστρέφεται" καλείται ο κωδικός του καταστροφέα μας (`__del__`). Δεν μπορούμε να σταματήσουμε την καταστροφή της μεταβλητής μας, αλλά μπορούμε να κάνουμε οποιονδήποτε απαραίτητο καθαρισμό πριν το αντικείμενό μας πάψει να υπάρχει πλέον.

Κατά την ανάπτυξη αντικειμένων, είναι αρκετά συνηθισμένο να προσθέτουμε έναν κατασκευαστή σε ένα αντικείμενο, για να ορίσουμε αρχικές τιμές για το αντικείμενο. Είναι σχετικά σπάνιο να χρειαστείτε καταστροφέα για ένα αντικείμενο.

Πολλαπλά στιγμιότυπα

Μέχρι στιγμής, ορίσαμε μια κλάση, κατασκευάσαμε ένα μεμονωμένο αντικείμενο, χρησιμοποιήσαμε αυτό το αντικείμενο και μετά το πετάξαμε. Ωστόσο, η πραγματική δύναμη του αντικειμενοστραφούς προγραμματισμού εκδηλώνεται όταν κατασκευάζουμε πολλαπλά στιγμιότυπο της κλάσης μας.

Όταν κατασκευάζουμε πολλά αντικείμενα από την κλάση μας, ίσως θελήσουμε να ορίσουμε διαφορετικές αρχικές τιμές, σε καθένα από τα αντικείμενα. Μπορούμε να περάσουμε δεδομένα στους κατασκευαστές για να δώσουμε σε κάθε αντικείμενο διαφορετική αρχική τιμή:

```
class PartyAnimal:
   x = 0
   name = ''
   def __init__(self, nam):
     self.name = nam
```

```
    print(self.name,'constructed')

  def party(self) :
    self.x = self.x + 1
    print(self.name,'party count',self.x)

s = PartyAnimal('Sally')
j = PartyAnimal('Jim')

s.party()
j.party()
s.party()
```

#Code: http://www.gr.py4e.com/code3/party5.py

Ο κατασκευαστής έχει και μια παράμετρο `self`, που δείχνει στο στιγμιότυπο του αντικειμένου, και πρόσθετες παραμέτρους, που μεταβιβάζονται στον κατασκευαστή καθώς κατασκευάζεται το αντικείμενο:

```
s = PartyAnimal('Sally')
```

Εντός του κατασκευαστή, η δεύτερη γραμμή αντιγράφει την παράμετρο (`nam`), που μεταβιβάζεται στο χαρακτηριστικό `name` στο στιγμιότυπο του αντικειμένου.

```
self.name = nam
```

Η έξοδος του προγράμματος δείχνει ότι καθένα από τα αντικείμενα (`s` και `j`) περιέχει τα δικά του ανεξάρτητα αντίγραφα των `x` και `nam`:

```
Sally constructed
Jim constructed
Sally party count 1
Jim party count 1
Sally party count 2
```

Κληρονομικότητα

Ένα άλλο ισχυρό χαρακτηριστικό του αντικειμενοστρεφούς προγραμματισμού είναι η δυνατότητα δημιουργίας μιας νέας κλάσης επεκτείνοντας μια υπάρχουσα κλάση. Όταν επεκτείνουμε μια κλάση, ονομάζουμε την αρχική κλάση *κλάση γονέας* και τη νέα κλάση *κλάση παιδί*.

Για αυτό το παράδειγμα, μετακινούμε την κλάση PartyAnimal στο δικό της αρχείο. Στη συνέχεια, μπορούμε να 'εισάγουμε - import' την κλάση PartyAnimal σε ένα νέο αρχείο και να την επεκτείνουμε, ως εξής:

```
from party import PartyAnimal

class CricketFan(PartyAnimal):
   points = 0
   def six(self):
      self.points = self.points + 6
      self.party()
      print(self.name,"points",self.points)

s = PartyAnimal("Sally")
s.party()
j = CricketFan("Jim")
j.party()
j.six()
print(dir(j))
```

#Code: http://www.gr.py4e.com/code3/party6.py

Όταν ορίζουμε την κλάση CricketFan, υποδεικνύουμε ότι επεκτείνουμε την κλάση PartyAnimal. Αυτό σημαίνει ότι όλες οι μεταβλητές (x) και οι μέθοδοι (party) της κλάσης PartyAnimal *κληρονομούνται* από την κλάση CricketFan. Για παράδειγμα, στη μέθοδο six, της κλάσης CricketFan, καλούμε τη μέθοδο party, από την κλάση PartyAnimal.

Καθώς εκτελείται το πρόγραμμα, δημιουργούμε τα s και j, ως ανεξάρτητα στιγμιότυπα των PartyAnimal και CricketFan. Το αντικείμενο j έχει πρόσθετες δυνατότητες πέρα από αυτές του αντικειμένου s.

```
Sally constructed
Sally party count 1
Jim constructed
Jim party count 1
Jim party count 2
Jim points 6
['__class__', '__delattr__', ... '__weakref__',
'name', 'party', 'points', 'six', 'x']
```

Στην έξοδο dir, για το αντικείμενο j (στιγμιότυπο της κλάσης CricketFan), βλέπουμε ότι έχει τα χαρακτηριστικά και τις μεθόδους της γονικής κλάσης, καθώς και τα χαρακτηριστικά και τις μεθόδους που προστέθηκαν όταν η κλάση επεκτάθηκε, για να δημιουργηθεί η κλάση CricketFan.

Περίληψη

Αυτή είναι μια πολύ γρήγορη εισαγωγή στον αντικειμενοστραφή προγραμματισμό, που εστιάζει κυρίως στην ορολογία, τον ορισμό και τη χρήσης αντικειμένων. Ας δούμε, γρήγορα, τον κώδικα που είδαμε στην αρχή του κεφαλαίου. Σε αυτό το σημείο θα πρέπει να καταλάβετε πλήρως τι συμβαίνει.

```
stuff = list()
stuff.append('python')
stuff.append('chuck')
stuff.sort()
print (stuff[0])
print (stuff.__getitem__(0))
print (list.__getitem__(stuff,0))
```

#Code: http://www.gr.py4e.com/code3/party1.py

Η πρώτη γραμμή δημιουργεί ένα *αντικείμενο* list (λίστα). Όταν η Python δημιουργεί το αντικείμενο list, καλεί τη μέθοδο *κατασκευαστή* (με το όνομα __init__), για να ρυθμίσει τα εσωτερικά χαρακτηριστικά δεδομένων, που θα χρησιμοποιηθούν για την αποθήκευση των δεδομένων της λίστας. Δεν έχουμε περάσει καμία παράμετρο στον *κατασκευαστή*. Όταν ο κατασκευαστής επιστρέφει, χρησιμοποιούμε τη μεταβλητή stuff, για να δείξουμε το επιστρεφόμενο στιγμιότυπο της κλάσης list.

Η δεύτερη και η τρίτη γραμμή καλούν τη μέθοδο append με μία παράμετρο, για να προσθέσουν ένα νέο στοιχείο στο τέλος της λίστας, ενημερώνοντας τα χαρακτηριστικά μέσα στο stuff. Στη συνέχεια, στην τέταρτη γραμμή, καλούμε τη μέθοδο sort χωρίς παραμέτρους, για να ταξινομήσουμε τα δεδομένα μέσα στο αντικείμενο stuff.

Στη συνέχεια, εκτυπώνουμε το πρώτο στοιχείο στη λίστα, χρησιμοποιώντας τις αγκύλες, που αποτελούν συντόμευση για την κλήση της μεθόδου __getitem__ μέσα στο stuff. Αυτό ισοδυναμεί με την κλήση της μεθόδου __getitem__ στη *κλάση* list και τη διαβίβαση του αντικειμένου stuff ως πρώτη παράμετρο και τη θέση που αναζητούμε ως δεύτερη παράμετρο.

Στο τέλος του προγράμματος, το αντικείμενο `stuff` απορρίπτεται, αλλά όχι πριν καλέσετε τον *καταστροφέα* (με το όνομα `__del__`), έτσι ώστε το αντικείμενο να μπορεί να τακτοποιήσει τυχόν εκκρεμή ζητήματα όπως απαιτείται.

Αυτά είναι τα βασικά του αντικειμενοστρεφούς προγραμματισμού. Υπάρχουν πολλές πρόσθετες λεπτομέρειες σχετικά με τον καλύτερο τρόπο χρήσης αντικειμενοστρεφών προσεγγίσεων κατά την ανάπτυξη μεγάλων εφαρμογών και βιβλιοθηκών, που δεν εμπίπτουν στο πεδίο αυτού του κεφαλαίου.[1]

Γλωσσάριο

attribute - χαρακτηριστικό ή ιδιότητα : Μια μεταβλητή που είναι μέρος μιας κλάσης.

αντικείμενο - object: Ένα κατασκευασμένο στιγμιότυπο μιας κλάσης. Ένα αντικείμενο περιέχει όλα τα χαρακτηριστικά και τις μεθόδους, που ορίστηκαν από την κλάση. Κάποια αντικειμενοστραφή τεκμηρίωση χρησιμοποιεί τον όρο 'στιγμιότυπο' εναλλακτικά του 'αντικείμενο'.

κατασκευστής - constructor: Μια προαιρετική μέθοδος με ειδική ονομασία (`__init__`), που καλείται τη στιγμή που μια κλάση χρησιμοποιείται για την κατασκευή ενός αντικειμένου. Συνήθως χρησιμοποιείται για τη ρύθμιση αρχικών τιμών του αντικειμένου.

καταστροφέας – destructor : Μια προαιρετική μέθοδος με ειδική ονομασία (`__del__`), που καλείται τη στιγμή ακριβώς πριν από την καταστροφή ενός αντικειμένου. Οι καταστροφείς χρησιμοποιούνται σπάνια.

κλάση : Ένα πρότυπο, που μπορεί να χρησιμοποιηθεί για την κατασκευή ενός αντικειμένου. Καθορίζει τα χαρακτηριστικά και τις μεθόδους που θα αποτελέσουν το αντικείμενο.

---

[1] Εάν είστε περίεργοι για το πού ορίζεται η κλάση `list`, ρίξτε μια ματιά στο (ελπίζουμε ότι η διεύθυνση URL δεν θα αλλάξει) https://github.com/python/cpython/blob/master/Objects/listobject.c η κλάση λίστας είναι γραμμένη σε μια γλώσσα που ονομάζεται "C". Αν ρίξετε μια ματιά σε αυτόν τον πηγαίο κώδικα και τον βρείτε περίεργο, ίσως θέλετε να εξερευνήσετε μερικά μαθήματα Επιστήμης Υπολογιστών.

κλάση γονέας - parent class : Η κλάση που επεκτείνεται για τη δημιουργία μιας νέας θυγατρικής κλάσης. Η γονεϊκή κλάση συνεισφέρει όλες τις μεθόδους και τα χαρακτηριστικά της στη νέα θυγατρική κλάση.

κλάση παιδί - child class : Μια νέα κλάση που δημιουργείται όταν επεκτείνεται μια γονεϊκή κλάση. Η κλάση παιδί κληρονομεί όλα τα χαρακτηριστικά και τις μεθόδους της γονεϊκής κλάσης.

κληρονομικότητα inheritance : Όταν δημιουργούμε μια νέα κλάση (παιδί) επεκτείνοντας μια υπάρχουσα κλάση (γονέας). Η θυγατρική κλάση έχει όλα τα χαρακτηριστικά και τις μεθόδους της γονεϊκής κλάσης συν επιπλέον χαρακτηριστικά και μεθόδους που ορίζονται από τη θυγατρική κλάση.

μέθοδος : Μια συνάρτηση που περιέχεται σε μια κλάση και στα αντικείμενα που κατασκευάζονται από αυτή την κλάση. Ορισμένα αντικειμενοστραφή μοτίβα χρησιμοποιούν τον όρο 'μήνυμα' αντί για 'μέθοδο' για να περιγράψουν αυτήν την έννοια.

Κεφάλαιο 15

Χρήση Βάσεων Δεδομένων και SQL

Τι είναι μια βάση δεδομένων;

Μια *βάση δεδομένων* είναι ένα αρχείο που είναι οργανωμένο για την αποθήκευση δεδομένων. Οι περισσότερες βάσεις δεδομένων είναι οργανωμένες σαν λεξικό, με την έννοια ότι υπάρχει αντιστοίχιση ανάμεσα σε κλειδιά και τιμές. Η μεγαλύτερη διαφορά είναι ότι η βάση δεδομένων βρίσκεται στο δίσκο (ή σε άλλη μονάδα μόνιμης αποθήκευσης), επομένως παραμένει και μετά το τέλος του προγράμματος. Επειδή μια βάση δεδομένων αποθηκεύεται σε μονάδα μόνιμης αποθήκευσης, μπορεί να αποθηκεύσει πολύ περισσότερα δεδομένα από ένα λεξικό, το οποίο περιορίζεται στο μέγεθος της μνήμης στον υπολογιστή.

Όπως ένα λεξικό, το λογισμικό βάσης δεδομένων έχει σχεδιαστεί για να παρέχει πολύ γρήγορη εισαγωγή και πρόσβαση στα δεδομένα, ακόμη και για μεγάλες ποσότητες δεδομένων. Το λογισμικό βάσης δεδομένων διατηρεί την απόδοσή του δημιουργώντας *ευρετήρια*, καθώς τα δεδομένα προστίθενται στη βάση δεδομένων, για να επιτρέψουν στον υπολογιστή την γρήγορη προσπέλαση μιας συγκεκριμένης καταχώρησης.

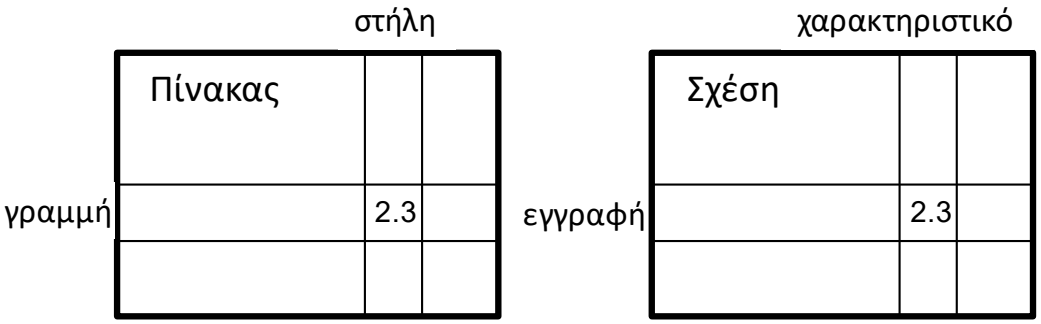
Υπάρχουν πολλά διαφορετικά συστήματα βάσεων δεδομένων, που χρησιμοποιούνται για μια μεγάλη ποικιλία σκοπών, όπως: Oracle, MySQL, Microsoft SQL Server, PostgreSQL και SQLite. Εστιάζουμε στο SQLite, σε αυτό το βιβλίο, επειδή είναι μια πολύ κοινή βάση δεδομένων και είναι ήδη ενσωματωμένη στην Python. Το SQLite έχει σχεδιαστεί για να *ενσωματώνεται* σε άλλες εφαρμογές και να παρέχει υποστήριξη βάσης δεδομένων εντός της εφαρμογής. Για παράδειγμα, το πρόγραμμα περιήγησης Firefox χρησιμοποιεί, κι αυτό, τη βάση δεδομένων SQLite, εσωτερικά, όπως και πολλά άλλα προϊόντα.

http://sqlite.org/

Το SQLite είναι κατάλληλο για ορισμένα από τα προβλήματα χειρισμού δεδομένων, που βλέπουμε στην Πληροφορική, όπως η εφαρμογή ιστοσυγκομιδής του Twitter, που περιγράφουμε σε αυτό το κεφάλαιο.

Έννοιες βάσης δεδομένων

Όταν κοιτάζετε για πρώτη φορά μια βάση δεδομένων, μοιάζει με υπολογιστικό φύλλο με πολλά φύλλα. Οι κύριες δομές δεδομένων σε μια βάση δεδομένων είναι: *πίνακες*, *γραμμές* και *στήλες*.



Εικόνα 15.1: Σχεσιακές βάσεις δεδομένων

Στις τεχνικές περιγραφές των σχεσιακών βάσεων δεδομένων, οι έννοιες του πίνακα, της γραμμής και της στήλης αναφέρονται και ως *σχέση*, *εγγραφή* και *χαρακτηριστικό*, αντίστοιχα. Θα χρησιμοποιήσουμε τους λιγότερο επίσημους όρους σε αυτό το κεφάλαιο.

Πρόγραμμα περιήγησης βάσεων δεδομένων για SQLite

Ενώ αυτό το κεφάλαιο θα επικεντρωθεί στη χρήση της Python για εργασία με δεδομένα σε αρχεία βάσης δεδομένων SQLite, πολλές λειτουργίες μπορούν να γίνουν πιο βολικά χρησιμοποιώντας το λογισμικό που ονομάζεται *Database Browser for SQLite (Πρόγραμμα περιήγησης βάσης δεδομένων για SQLite)* το οποίο διατίθεται δωρεάν από:

http://sqlitebrowser.org/

Χρησιμοποιώντας το πρόγραμμα περιήγησης μπορείτε εύκολα να δημιουργήσετε πίνακες, να εισαγάγετε δεδομένα, να επεξεργαστείτε δεδομένα ή να εκτελέσετε απλά ερωτήματα SQL στα δεδομένα της βάσης δεδομένων.

Κατά μία έννοια, το πρόγραμμα περιήγησης της βάσης δεδομένων είναι παρόμοιο με ένα πρόγραμμα επεξεργασίας κειμένου, όταν εργάζεστε με αρχεία κειμένου. Όταν θέλετε να κάνετε μία ή πολύ λίγες επεξεργασίες σε ένα αρχείο κειμένου, μπορείτε απλώς να το ανοίξετε σε ένα πρόγραμμα επεξεργασίας κειμένου και να κάνετε τις αλλαγές που θέλετε. Όταν έχετε πολλές αλλαγές που πρέπει να κάνετε σε ένα αρχείο κειμένου, συχνά γράφετε ένα απλό πρόγραμμα Python. Θα εντοπίσετε την ίδια λογική, όταν εργάζεστε με βάσεις δεδομένων. Θα υλοποιείται απλές λειτουργίες στον διαχειριστή βάσης δεδομένων και πιο σύνθετες λειτουργίες θα γίνονται πιο βολικά με την Python.

Δημιουργία πίνακα βάσης δεδομένων

Οι βάσεις δεδομένων απαιτούν πιο καθορισμένη δομή από τις λίστες ή τα λεξικά Python[1].

Όταν δημιουργούμε έναν *πίνακα* βάσης δεδομένων, πρέπει να πούμε εκ των προτέρων στη βάση δεδομένων τα ονόματα καθεμιάς από τις *στήλες* στον πίνακα και τον τύπο των δεδομένων που σκοπεύουμε να αποθηκεύσουμε σε κάθε *στήλη*. Όταν το λογισμικό της βάσης δεδομένων γνωρίζει τον τύπο των δεδομένων σε κάθε στήλη, μπορεί να επιλέξει τον πιο αποτελεσματικό τρόπο αποθήκευσης και αναζήτησης των δεδομένων, με βάση τον τύπο των δεδομένων.

Μπορείτε να δείτε τους διάφορους τύπους δεδομένων που υποστηρίζονται από το SQLite στην ακόλουθη διεύθυνση url:

http://www.sqlite.org/datatypes.html

Ο καθορισμός της δομής για τα δεδομένα σας εκ των προτέρων μπορεί να φαίνεται άβολος στην αρχή, αλλά το αποτέλεσμα είναι η γρήγορη πρόσβαση στα δεδομένα σας, ακόμα και όταν η βάση δεδομένων περιέχει μεγάλο όγκο δεδομένων.

Ο κώδικας για τη δημιουργία ενός αρχείου βάσης δεδομένων και ενός πίνακα με το όνομα `Tracks` με δύο στήλες στη βάση δεδομένων είναι ο εξής:

```
import sqlite3

conn = sqlite3.connect('music.sqlite')
cur = conn.cursor()

cur.execute('DROP TABLE IF EXISTS Tracks')
cur.execute('CREATE TABLE Tracks (title TEXT, plays INTEGER)')

conn.close()
```

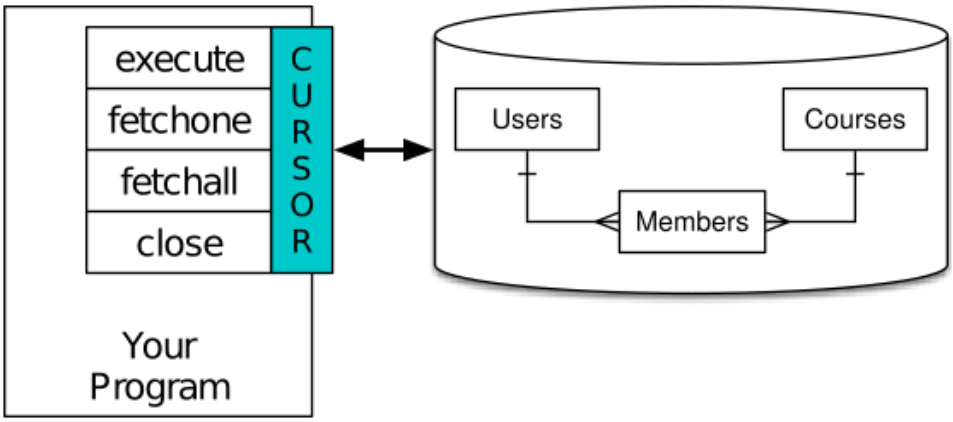
#Code: http://www.gr.py4e.com/code3/db1.py

Η λειτουργία `connect` πραγματοποιεί μια "σύνδεση" με τη βάση δεδομένων, που είναι αποθηκευμένη στο αρχείο `music.sqlite`, στον τρέχοντα κατάλογο. Εάν το αρχείο δεν

[1] Το SQLite στην πραγματικότητα επιτρέπει κάποια ευελιξία στον τύπο των δεδομένων που αποθηκεύονται σε μια στήλη, αλλά θα διατηρήσουμε τους τύπους δεδομένων μας αυστηρούς σε αυτό το κεφάλαιο, ώστε οι έννοιες να ισχύουν εξίσου και σε άλλα συστήματα βάσεων δεδομένων, όπως π.χ. MySQL.

υπάρχει, θα δημιουργηθεί. Ο λόγος που αυτό ονομάζεται "σύνδεση" είναι ότι μερικές φορές η βάση δεδομένων αποθηκεύεται σε έναν ξεχωριστό "διακομιστή βάσης δεδομένων", από τον διακομιστή στον οποίο εκτελούμε την εφαρμογή μας. Στα απλά παραδείγματά μας η βάση δεδομένων θα είναι απλώς ένα τοπικό αρχείο στον ίδιο κατάλογο με τον κώδικα Python που εκτελούμε.

Ένας *κέρσορας (cursor)* είναι σαν ένας περιγραφέας αρχείου, που μπορούμε να χρησιμοποιήσουμε για να εκτελέσουμε λειτουργίες επί των δεδομένων, που είναι αποθηκευμένα στη βάση δεδομένων. Η κλήση του `cursor()` μοιάζει πολύ εννοιολογικά με την κλήση του `open()`, όταν πρόκειται για αρχεία κειμένου.



Εικόνα 15.2: Ένας Κέρσορας Βάσης Δεδομένων

Μόλις έχουμε τον κέρσορα, μπορούμε να αρχίσουμε να εκτελούμε εντολές προς τα περιεχόμενα της βάσης δεδομένων χρησιμοποιώντας τη μέθοδο `execute()`.

Οι εντολές βάσης δεδομένων εκφράζονται σε μια ειδική γλώσσα, που έχει τυποποιηθεί σε πολλούς διαφορετικούς προμηθευτές βάσεων δεδομένων για να μας δώσουν τη δυνατότητα να μάθουμε μια ενιαία γλώσσα βάσης δεδομένων. Η γλώσσα των βάσεων δεδομένων ονομάζεται *Γλώσσα Δομημένων Ερωταπαντήσεων - Structured Query Language* ή *SQL* για συντομία.

http://en.wikipedia.org/wiki/SQL

Στο παράδειγμά μας, εκτελούμε δύο εντολές SQL στη βάση δεδομένων μας. Κατά σύμβαση, θα γράφουμε τις δεσμευμένες λέξεις της SQL με κεφαλαία γράμματα και τα υπόλοιπα μέρη της εντολής, που προσθέτουμε (όπως τα ονόματα του πίνακα και των στηλών) θα εμφανίζονται με πεζά.

Η πρώτη εντολή SQL διαγράφει τον πίνακα `Tracks` από τη βάση δεδομένων, εάν υπάρχει. Αυτό το μοτίβο χρησιμοποιείται απλώς για να μας επιτρέψει να εκτελέσουμε το ίδιο πρόγραμμα και να δημιουργήσουμε ξανά και ξανά τον πίνακα `Tracks` χωρίς να

προκληθεί σφάλμα. Σημειώστε ότι η εντολή DROP TABLE διαγράφει τον πίνακα και όλα τα περιεχόμενά του από τη βάση δεδομένων (δηλαδή, δεν υπάρχει "αναίρεση").

```
cur.execute('DROP TABLE IF EXISTS Tracks ')
```

Η δεύτερη εντολή δημιουργεί έναν πίνακα με όνομα Tracks, με μια στήλη κειμένου, που ονομάζεται title και μια στήλη ακεραίων, με το όνομα plays.

```
cur.execute('CREATE TABLE Tracks (title TEXT, plays INTEGER)')
```

Τώρα που δημιουργήσαμε έναν πίνακα με το όνομα Tracks, μπορούμε να εισάγουμε κάποια δεδομένα σε αυτόν τον πίνακα, χρησιμοποιώντας την εντολή SQL INSERT. Και πάλι, ξεκινάμε κάνοντας μια σύνδεση με τη βάση δεδομένων και αποκτώντας τον cursor. Στη συνέχεια, μπορούμε να εκτελέσουμε εντολές SQL, χρησιμοποιώντας τον κέρσορα.

Η εντολή SQL INSERT δηλώνει ποιον πίνακα θα χρησιμοποιήσουμε και στη συνέχεια ορίζει μια νέα σειρά, παραθέτοντας τα πεδία που θέλουμε να συμπεριλάβουμε (title, plays), ακολουθούμενα από το VALUES και τις τιμές, που θέλουμε να τοποθετηθούν στη νέα σειρά. Καθορίζουμε τις τιμές ως ερωτηματικά (?, ?), για να υποδείξουμε ότι οι πραγματικές τιμές μεταβιβάζονται ως η πλειάδα ('My Way', 15) και μάλιστα, ως η δεύτερη παράμετρος στην κλήση της execute().

```
import sqlite3
conn = sqlite3.connect('music.sqlite')
cur = conn.cursor()

cur.execute('INSERT INTO Tracks (title, plays) VALUES (?, ?)',
    ('Thunderstruck', 20))
cur.execute('INSERT INTO Tracks (title, plays) VALUES (?, ?)',
    ('My Way', 15))
conn.commit()

print('Tracks:')
cur.execute('SELECT title, plays FROM Tracks')
for row in cur:
     print(row)

cur.execute('DELETE FROM Tracks WHERE plays < 100')
conn.commit()
cur.close()
```

#Code: http://www.gr.py4e.com/code3/db2.py

Πρώτα, με το INSERT εισαγάγουμε δύο γραμμές στον πίνακά μας και χρησιμοποιούμε την commit() για να αναγκάσουμε τα δεδομένα να εγγραφούν στο αρχείο της βάσης δεδομένων.

Tracks

title	plays
Thunderstruck	20
My Way	15

Εικόνα 15.3: Γραμμές σε έναν Πίνακα

Στη συνέχεια, χρησιμοποιούμε την εντολή SELECT, για να ανακτήσουμε τις γραμμές που μόλις εισαγάγαμε από τον πίνακα. Στην εντολή SELECT, δηλώνουμε ποιες στήλες θέλουμε (title, plays) και δηλώνουμε επίσης από ποιον πίνακα θέλουμε να ανακτήσουμε τα δεδομένα. Αφού εκτελέσουμε την πρόταση SELECT, ο κέρσορας μπορεί να χρησιμοποιηθεί στο βρόχο for για να διατρέξουμε τις γραμμές που ανέκτησε η SELECT. Για λόγους αποτελεσματικότητας, ο κέρσορας δεν διαβάζει όλα τα δεδομένα από τη βάση δεδομένων όταν εκτελούμε την εντολή SELECT. Αντίθετα, τα δεδομένα διαβάζονται κατ' απαίτηση καθώς κάνουμε τις απαιτούμε μία μία, με το βρόχοfor.

Η έξοδος του προγράμματος είναι η εξής:

```
Tracks:
('Thunderstruck', 20)
('My Way', 15)
```

Ο βρόχος for βρίσκει δύο γραμμές και κάθε γραμμή είναι μια πλειάδα Python με την πρώτη τιμή ως title και τη δεύτερη τιμή ως τον αριθμό των plays.

*Σημείωση: Μπορεί να δείτε συμβολοσειρές που ξεκινούν με u' σε άλλα βιβλία ή στο Διαδίκτυο. Αυτό ήταν μια ένδειξη στην Python 2 ότι οι συγκεκριμένες συμβολοσειρές είναι συμβολοσειρές **Unicode**, που μπορούν να αποθηκεύουν σύνολα μη Λατινικών χαρακτήρων. Στην Python 3, όλες οι συμβολοσειρές είναι από προεπιλογή συμβολοσειρές unicode.*

Στο τέλος του προγράμματος, εκτελούμε μια εντολή SQL DELETE για διαγραφή των γραμμών που μόλις δημιουργήσαμε, ώστε να μπορούμε να τρέχουμε ξανά και ξανά το ίδιο πρόγραμμα. Η εντολή DELETE χρησιμοποιεί τον όρο WHERE, που μας επιτρέπει να εφαρμόσουμε ένα κριτήριο επιλογής, ώστε να μπορούμε να ζητήσουμε από τη βάση δεδομένων να εφαρμόσει την εντολή μόνο στις γραμμές που ταιριάζουν με το κριτήριο. Σε αυτό το παράδειγμα το κριτήριο συμβαίνει να ισχύει για όλες τις γραμμές, έτσι

αδειάζουμε τον πίνακα, ώστε να μπορούμε να τρέξουμε το πρόγραμμα επανειλημμένα. Αφού εκτελεστεί η DELETE, καλούμε επίσης την commit(), για να κατοχυρώσουμε τις αλλαγές (διαγραφές) στη βάση δεδομένων.

Σύνοψη γλώσσας δομημένων ερωταπαντήσεων

Μέχρι στιγμής, χρησιμοποιούσαμε τη γλώσσα δομημένων ερωταπαντήσεων μέσα στα παραδείγματα Python και έχουμε καλύψει πολλά από τα βασικά των εντολών SQL. Σε αυτήν την ενότητα, εξετάζουμε συγκεκριμένα τη γλώσσα SQL και δίνουμε μια επισκόπηση της σύνταξης SQL.

Δεδομένου ότι υπάρχουν τόσοι πολλοί διαφορετικοί προμηθευτές βάσεων δεδομένων, η γλώσσα δομημένων ερωταπαντήσεων (SQL) τυποποιήθηκε, ώστε να μπορούμε να επικοινωνούμε, με φορητό τρόπο, με συστήματα βάσεων δεδομένων από διαφορετικούς προμηθευτές.

Μια σχεσιακή βάση δεδομένων αποτελείται από πίνακες, γραμμές και στήλες. Οι στήλες έχουν γενικά έναν τύπο, όπως δεδομένα κειμένου, αριθμών ή ημερομηνίας. Όταν δημιουργούμε έναν πίνακα, δηλώνουμε τα ονόματα και τους τύπους των στηλών:

```
CREATE TABLE Tracks (title TEXT, plays INTEGER)
```

Για να εισάγουμε μια γραμμή σε έναν πίνακα, χρησιμοποιούμε την εντολή SQL INSERT:

```
INSERT INTO Tracks (title, plays) VALUES ('My Way', 15)
```

Η εντολή INSERT καθορίζει το όνομα του πίνακα, στη συνέχεια μια λίστα με τα πεδία/στήλες που θέλετε να καταχωρήσετε στη νέα γραμμή και, στη συνέχεια, τη δεσμευμένη λέξη VALUES και μια λίστα με τις αντίστοιχες τιμές, για κάθε ένα από τα πεδία αυτά.

Η εντολή SQL SELECT χρησιμοποιείται για την ανάκτηση γραμμών και στηλών από μια βάση δεδομένων. Η εντολή SELECT σάς επιτρέπει να καθορίσετε ποιες στήλες θέλετε να ανακτήσετε όπως επίσης, η δήλωση WHERE σάς επιτρέπει να επιλέξετε ποιες γραμμές θέλετε να επιστραφούν. Μπορεί επίσης να περιέχει έναν προαιρετικό όρο ORDER BY για την ταξινόμηση των επιστρεφόμενων γραμμών.

```
SELECT * FROM Tracks WHERE title = 'My Way'
```

Η χρήση του * δηλώνει ότι θέλετε η βάση δεδομένων να επιστρέψει όλες τις στήλες, για κάθε γραμμή που ταιριάζει με την πρόταση WHERE.

Σημειώστε, σε αντίθεση με την Python, σε μια πρόταση SQL WHERE χρησιμοποιούμε ένα μόνο σύμβολο ίσου για να υποδείξουμε μια σύγκριση ισότητας, αντί για διπλό σύμβολο ίσου. Άλλοι συγκριτικοί και λογικοί τελεστές, που επιτρέπονται, σε μια πρόταση WHERE περιλαμβάνουν τα <, >, <=, >=, !=, καθώς και τα AND και OR και παρενθέσεις για τη δημιουργία των λογικών σας εκφράσεων.

Μπορείτε να ζητήσετε να ταξινομηθούν οι επιστρεφόμενες γραμμές, κατά ένα από τα πεδία, ως εξής:

```
SELECT title,plays FROM Tracks ORDER BY title
```

Για να διαγράψετε μια γραμμή, χρειάζεστε μια πρόταση WHERE σε συνδυασμό με μια εντολή SQL DELETE. Η πρόταση WHERE καθορίζει ποιες σειρές θα διαγραφούν:

```
DELETE FROM Tracks WHERE title = 'My Way'
```

Είναι δυνατό να "ενημερώσετε" μια ή περισσότερες στήλες σε μία ή περισσότερες γραμμές ενός πίνακα, χρησιμοποιώντας την εντολή SQL UPDATE ως εξής:

```
UPDATE Tracks SET plays = 16 WHERE title = 'My Way'
```

Η εντολή UPDATE καθορίζει έναν πίνακα και, στη συνέχεια, μια λίστα πεδίων και τιμών που θα αλλάξουν, μετά τη δεσμευμένη λέξη SET και, στη συνέχεια, μια προαιρετική πρόταση WHERE, για να επιλέξετε τις γραμμές που πρόκειται να ενημερωθούν. Μια μεμονωμένη δήλωση UPDATE ενημερώνει όλες τις γραμμές που ταιριάζουν με την πρόταση WHERE. Εάν δεν έχει καθοριστεί η πρόταση WHERE, εκτελείται "ενημέρωση" όλων των γραμμών του πίνακα.

Αυτές οι τέσσερις βασικές εντολές SQL (INSERT, SELECT, UPDATE και DELETE) υλοποιούν τις τέσσερις βασικές λειτουργίες, που απαιτούνται για τη δημιουργία και τη συντήρηση των δεδομένων.

Ανίχνευση του Twitter με χρήση βάσης δεδομένων

Σε αυτή την ενότητα, θα δημιουργήσουμε ένα απλό πρόγραμμα ανίχνευσης (spidering), που θα διατρέξει λογαριασμούς του Twitter και θα δημιουργήσει μια βάση δεδομένων με αυτούς. *Σημείωση: Να είστε πολύ προσεκτικοί όταν εκτελείτε αυτό το πρόγραμμα. Δεν θέλετε να τραβήξετε πάρα πολλά δεδομένα ή να εκτελέσετε το πρόγραμμα για πολύ μεγάλο χρονικό διάστημα και να καταλήξετε να τερματίσετε την πρόσβασή σας στο Twitter.*

Ένα από τα προβλήματα κάθε είδους προγράμματος spidering είναι ότι πρέπει να μπορεί να διακοπεί και να επανεκκινηθεί πολλές φορές και δεν θέλετε να χάσετε τα δεδομένα που έχετε ανακτήσει μέχρι τώρα. Δεν θέλετε να επανεκκινείτε και η ανάκτηση των δεδομένων να αρχίζει στην αρχή, επομένως θέλουμε να αποθηκεύουμε τα δεδομένα καθώς τα ανακτούμε, ώστε το πρόγραμμά μας να μπορεί να δημιουργεί αντίγραφα ασφαλείας και να συνεχίζει από εκεί που σταμάτησε.

Θα ξεκινήσουμε με την ανάκτηση των φίλων ενός ατόμου στο Twitter και της κατάστασής τους, διατρέχοντας επαναληπτικά τη λίστα των φίλων και προσθέτοντας κάθε έναν από τους φίλους σε μια βάση δεδομένων, την οποία θα προσπελάσουμε αργότερα. Αφού επεξεργαστούμε τους φίλους ενός ατόμου στο Twitter, ελέγχουμε τη βάση δεδομένων μας και ανακτούμε έναν από τους φίλους του φίλου. Το κάνουμε αυτό ξανά και ξανά, επιλέγοντας ένα άτομο που δεν έχει "επισκεφτεί", ανακτώντας τη λίστα φίλων του και προσθέτοντας φίλους, που δεν έχουμε δει στη λίστα μας για μελλοντική επίσκεψη.

Παρακολουθούμε επίσης πόσες φορές έχουμε δει έναν συγκεκριμένο φίλο στη βάση δεδομένων για να σχηματίσουμε μια εικόνα της «δημοτικότητάς» του.

Αποθηκεύοντας τη λίστα των γνωστών λογαριασμών μας και εάν έχουμε ανακτήσει τον λογαριασμό ή όχι και πόσο δημοφιλής είναι ο λογαριασμός σε μια βάση δεδομένων, στο δίσκο του υπολογιστή, μπορούμε να σταματήσουμε και να επανεκκινήσουμε το πρόγραμμά μας όσες φορές θέλουμε.

Αυτό το πρόγραμμα είναι λίγο περίπλοκο. Βασίζεται στον κώδικα από την άσκηση νωρίτερα στο βιβλίο, που χρησιμοποιεί το Twitter API.

Εδώ είναι ο πηγαίος κώδικας για την εφαρμογή ανίχνευσης του Twitter:

```
from urllib.request import urlopen
import urllib.error
import twurl
import json
import sqlite3
import ssl

TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'

conn = sqlite3.connect('spider.sqlite')
cur = conn.cursor()
```

```
cur.execute('''
            CREATE TABLE IF NOT EXISTS Twitter
            (name TEXT, retrieved INTEGER, friends INTEGER)''')

# Αγνοήστε τα σφάλματα πιστοποιητικού SSL
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
    acct = input('Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση
με quit: ')
    if (acct == 'quit'): break
    if (len(acct) < 1):
        cur.execute('SELECT name FROM Twitter WHERE retrieved = 0 LIMIT 1')
        try:
            acct = cur.fetchone()[0]
        except:
            print('Δεν βρέθηκαν λογαριασμοί Twitter που δεν έχουν ήδη
ανακτηθεί')
            continue

    url = twurl.augment(TWITTER_URL, {'screen_name': acct, 'count': '20'})
    print('Ανάκτηση του', url)
    connection = urlopen(url, context=ctx)
    data = connection.read().decode()
    headers = dict(connection.getheaders())

    print('Απομένουν', headers['x-rate-limit-remaining'])
    js = json.loads(data)
    # Εκσφαλμάτωση
    # print json.dumps(js, indent=4)

    cur.execute('UPDATE Twitter SET retrieved=1 WHERE name = ?', (acct, ))

    countnew = 0
    countold = 0
    for u in js['users']:
        friend = u['screen_name']
```

```
        print(friend)
        cur.execute('SELECT friends FROM Twitter WHERE name = ? LIMIT 1',
                    (friend, ))
        try:
            count = cur.fetchone()[0]
            cur.execute('UPDATE Twitter SET friends = ? WHERE name = ?',
                        (count+1, friend))
            countold = countold + 1
        except:
            cur.execute('''INSERT INTO Twitter (name, retrieved, friends)
                        VALUES (?, 0, 1)''', (friend, ))
            countnew = countnew + 1
    print('Νέοι λογαριασμοί =', countnew, ' Επισκέφθηκαν εκ νέου =', countold)
    conn.commit()

cur.close()
```

#Code: http://www.gr.py4e.com/code3/twspider.py

Η βάση δεδομένων μας είναι αποθηκευμένη στο αρχείο `spider.sqlite` και έχει έναν πίνακα με όνομα `Twitter`. Κάθε γραμμή του πίνακα `Twitter` έχει μια στήλη για το όνομα του λογαριασμού, μία για το εάν έχουμε ανακτήσει τους φίλους αυτού του λογαριασμού και μια τρίτη με το πόσες φορές αυτός ο λογαριασμός έχει γίνει "φίλος".

Στον κύριο βρόχο του προγράμματος, ζητάμε από τον χρήστη ένα όνομα λογαριασμού στο Twitter ή "quit" για έξοδο από το πρόγραμμα. Εάν ο χρήστης εισαγάγει έναν λογαριασμό Twitter, ανακτούμε τη λίστα φίλων και καταστάσεων (status) για αυτόν τον χρήστη και προσθέτουμε κάθε φίλο του στη βάση δεδομένων, εάν δεν βρίσκεται ήδη στη βάση δεδομένων. Εάν ο φίλος βρίσκεται ήδη στη λίστα, προσθέτουμε 1 στο πεδίο `friends` της γραμμής του, στη βάση δεδομένων.

Εάν ο χρήστης πατήσει enter, αναζητούμε στη βάση δεδομένων τον επόμενο λογαριασμό Twitter που δεν έχουμε ανακτήσει ακόμη, ανακτούμε τους φίλους και τα status για αυτόν τον λογαριασμό, τα προσθέτουμε στη βάση δεδομένων ή τους ενημερώνουμε και αυξάνουμε τον αριθμό των `friends` τους, εάν είναι ήδη καταχωρημένοι στη βάση δεδομένων.

Μόλις ανακτήσουμε τη λίστα των φίλων και των καταστάσεων, πραγματοποιούμε αναζήτηση σε όλα τα στοιχεία `user` στο επιστρεφόμενο JSON και ανακτούμε το `screen_name` για κάθε χρήστη. Στη συνέχεια, χρησιμοποιούμε τη δήλωση `SELECT` για να

δούμε αν έχουμε ήδη αποθηκεύσει το συγκεκριμένο screen_name στη βάση δεδομένων και ανακτούμε τον αριθμό φίλων (friends) εάν υπάρχει η εγγραφή.

```
countnew = 0
countold = 0
for u in js['users'] :
    friend = u['screen_name']
    print(friend)
    cur.execute('SELECT friends FROM Twitter WHERE name = ? LIMIT 1',
        (friend, ) )
    try:
        count = cur.fetchone()[0]
        cur.execute('UPDATE Twitter SET friends = ? WHERE name = ?',
            (count+1, friend) )
        countold = countold + 1
    except:
        cur.execute('''INSERT INTO Twitter (name, retrieved, friends)
            VALUES ( ?, 0, 1 )''', ( friend, ) )
        countnew = countnew + 1
print('Νέοι λογαριασμοί=',countnew,' Επισκέφθηκαν εκ νέου=',countold)
conn.commit()
```

Μόλις ο κέρσορας εκτελέσει την πρόταση SELECT, πρέπει να ανακτήσουμε τις γραμμές. Θα μπορούσαμε να το κάνουμε αυτό με μια δήλωση for, αλλά επειδή ανακτούμε μόνο μία γραμμή (LIMIT 1), μπορούμε να χρησιμοποιήσουμε τη μέθοδο fetchone() για να ανακτήσουμε την πρώτη (και μοναδική) σειρά που είναι το αποτέλεσμα τη λειτουργία SELECT. Επειδή το fetchone() επιστρέφει τη γραμμή ως *πλειάδα* (ακόμα και αν υπάρχει μόνο ένα πεδίο), χρησιμοποιούμε την πρώτη τιμή της πλειάδας για να εκχωρήσουμε τον τρέχοντα αριθμό φίλων στη μεταβλητή count.

Εάν αυτή η ανάκτηση είναι επιτυχής, χρησιμοποιούμε την εντολή UPDATE της SQL με μια δήλωση WHERE, για να προσθέσουμε 1 στη στήλη friends της γραμμής που ταιριάζει με τον λογαριασμό του φίλου. Σημειώστε ότι υπάρχουν δύο σύμβολα κράτησης θέσης (δηλαδή, ερωτηματικά) στον κώδικα SQL και ότι η δεύτερη παράμετρος της execute() είναι μια πλειάδα δύο στοιχείων, που περιέχει τις τιμές, που πρέπει να αντικαταστήσουν στον κώδικα SQL, τα ερωτηματικά.

Εάν ο κώδικας στο μπλοκ try αποτύχει, αυτό είναι πιθανό να συμβεί αν καμία εγγραφή δεν ταιριάζει με τον όρο WHERE name = ? στη δήλωση SELECT. Έτσι, στο μπλοκ except, χρησιμοποιούμε την εντολή SQL INSERT, για να προσθέσουμε το screen_name του φίλου

στον πίνακα με ένδειξη ότι δεν έχουμε ακόμη ανακτήσει το screen_name και ορίζουμε τον αριθμό φίλων (friends) σε ένα.

Έτσι την πρώτη φορά που τρέχει το πρόγραμμα και μπαίνουμε σε λογαριασμό Twitter, το πρόγραμμα εκτελείται ως εξής:

```
Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση με quit:
drchuck
Ανάκτηση του http://api.twitter.com/1.1/friends ...
Νέοι λογαριασμοί = 20  Επισκέφθηκαν εκ νέου = 0
Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση με quit: quit
```

Επειδή είναι η πρώτη φορά που τρέχουμε το πρόγραμμα, η βάση δεδομένων δεν υπάρχει, δημιουργούμε τη βάση στο αρχείο spider.sqlite και προσθέτουμε έναν πίνακα με το όνομα Twitter στη βάση δεδομένων. Στη συνέχεια ανακτούμε μερικούς φίλους και τους προσθέτουμε όλους στη βάση δεδομένων, αφού η βάση είναι άδεια.

Σε αυτό το σημείο, μπορεί να θελήσουμε να γράψουμε ένα απλό πρόγραμμα για να ρίξουμε μια ματιά στο περιεχόμενο του αρχείου spider.sqlite:

```
import sqlite3

conn = sqlite3.connect('spider.sqlite')
cur = conn.cursor()
cur.execute('SELECT * FROM Twitter')
count = 0
for row in cur:
    print(row)
    count = count + 1
print(count, 'γραμμές.')
cur.close()
```

#Code: http://www.gr.py4e.com/code3/twdump.py

Αυτό το πρόγραμμα απλώς ανοίγει τη βάση δεδομένων και επιλέγει όλες τις στήλες, όλων των γραμμών του πίνακα Twitter, στη συνέχεια διατρέχει τις γραμμές και εκτυπώνει κάθε μία από αυτές.

Εάν εκτελέσουμε αυτό το πρόγραμμα μετά την πρώτη εκτέλεση του Twitter spider παραπάνω, η έξοδος του θα είναι η εξής:

```
('opencontent', 0, 1)
('lhawthorn', 0, 1)
('steve_coppin', 0, 1)
```

```
('davidkocher', 0, 1)
('hrheingold', 0, 1)
...
20 γραμμές.
```

Βλέπουμε μια σειρά για κάθε screen_name, ότι δεν έχουμε ανακτήσει τα δεδομένα για αυτό το screen_name (η δεύτερη τιμή της πλειάδας που αντιστοιχεί στη στήλη retrieved είναι 0) και όλοι στη βάση δεδομένων έχουν έναν φίλο.

Τώρα η βάση δεδομένων μας αντικατοπτρίζει την ανάκτηση των φίλων του πρώτου μας λογαριασμού Twitter (*drchuck*). Μπορούμε να εκτελέσουμε ξανά το πρόγραμμα και να του πούμε να ανακτήσει τους φίλους του επόμενου "μη επεξεργασμένου" λογαριασμού πατώντας απλώς enter αντί για λογαριασμό Twitter ως εξής:

```
Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση με quit:
Ανάκτηση του http://api.twitter.com/1.1/friends ...
Νέοι λογαριασμοί = 18  Επισκέφθηκαν εκ νέου = 2
Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση με quit:
Ανάκτηση του http://api.twitter.com/1.1/friends ...
Νέοι λογαριασμοί = 17  Επισκέφθηκαν εκ νέου = 3
Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση με quit: quit
```

Αφού πατήσαμε enter (δηλαδή, δεν καθορίσαμε λογαριασμό Twitter), εκτελείται ο ακόλουθος κώδικας:

```
if ( len(acct) < 1 ) :
    cur.execute('SELECT name FROM Twitter WHERE retrieved = 0 LIMIT 1')
    try:
        acct = cur.fetchone()[0]
    except:
        print('Δεν βρέθηκαν λογαριασμοί Twitter που δεν έχουν ήδη
ανακτηθεί')
        continue
```

Χρησιμοποιούμε την εντολή SQL SELECT για να ανακτήσουμε το όνομα του πρώτου (LIMIT 1) χρήστη, που εξακολουθεί να έχει μηδενική τιμή στο "έχουμε ανακτήσει αυτόν τον χρήστη". Χρησιμοποιούμε επίσης το μοτίβο fetchone()[0], σε ένα μπλοκ try/except, για να εξαγάγουμε ένα screen_name από τα δεδομένα που ανακτήθηκαν ή για να εμφανίσουμε ένα μήνυμα σφάλματος και να τερματίσουμε την τρέχουσα εκτέλεση του βρόχου.

Εάν ανακτήσαμε με επιτυχία ένα μη επεξεργασμένο screen_name, ανακτούμε τα δεδομένα του ως εξής:

```
url=twurl.augment(TWITTER_URL,{'screen_name': acct,'count': '20'})
print('Ανάκτηση του', url)
connection = urllib.urlopen(url)
data = connection.read()
js = json.loads(data)

cur.execute('UPDATE Twitter SET retrieved=1 WHERE name = ?',(acct, ))
```

Μόλις ανακτήσουμε τα δεδομένα επιτυχώς, χρησιμοποιούμε την εντολή UPDATE για να θέσουμε στη στήλη retrieved το 1, για να υποδείξουμε ότι έχουμε ολοκληρώσει την ανάκτηση των φίλων αυτού του λογαριασμού. Αυτό μας εμποδίζει από το να ανακτούμε τα ίδια δεδομένα ξανά και ξανά και μας επιτρέπει να προχωράμε στο δίκτυο φίλων Twitter.

Εάν τρέξουμε το πρόγραμμα φίλου και πατήσουμε το enter δύο φορές για να ανακτήσουμε τους επόμενους φίλους, που δεν έχουν επισκεφτεί και μετά εκτελέσουμε το πρόγραμμα εμφάνισης (twdump.py), θα μας δώσει την ακόλουθη έξοδο:

```
('opencontent', 1, 1)
('lhawthorn', 1, 1)
('steve_coppin', 0, 1)
('davidkocher', 0, 1)
('hrheingold', 0, 1)
...
('cnxorg', 0, 2)
('knoop', 0, 1)
('kthanos', 0, 2)
('LectureTools', 0, 1)
...
55 γραμμές.
```

Μπορούμε να δούμε ότι έχουμε καταγράψει σωστά ότι έχουμε επισκεφτεί το lhawthorn και το opencontent. Επίσης οι λογαριασμοί cnxorg και kthanos έχουν ήδη δύο ακόλουθους. Εφόσον τώρα έχουμε ανακτήσει τους φίλους τριών ατόμων (drchuck, opencontent και lhawthorn), ο πίνακας μας έχει 52 (=55-3) γραμμές φίλων για ανάκτηση.

Κάθε φορά που εκτελούμε το πρόγραμμα και πατάμε enter, θα επιλέγει τον επόμενο λογαριασμό, που δεν έχει επισκεφτεί (π.χ., ο επόμενος λογαριασμός θα είναι ο `steve_coppin`), θα ανακτά τους φίλους του, θα τον επισημαίνει ως ανακτημένο και κάθε έναν από τους φίλους του `steve_coppin` είτε τον προσθέτει στο τέλος της βάσης δεδομένων είτε ενημερώνει τον αριθμό των φίλων του, εάν βρίσκονται ήδη στη βάση δεδομένων.

Μιας και τα δεδομένα του προγράμματος αποθηκεύονται όλα στο δίσκο, σε μια βάση δεδομένων, η δραστηριότητα spidering μπορεί να ανασταλεί και να συνεχιστεί όσες φορές θέλετε χωρίς απώλεια δεδομένων.

Στοιχειώδης μοντελοποίηση δεδομένων

Η πραγματική ισχύς μιας σχεσιακής βάσης δεδομένων φαίνεται όταν δημιουργούμε πολλούς πίνακες και τους συνδέουμε. Το να αποφασίσετε πώς να χωρίσετε τα δεδομένα της εφαρμογής σας σε πολλούς πίνακες και η δημιουργία των σχέσεων μεταξύ των πινάκων ονομάζεται *μοντελοποίηση δεδομένων*. Το έγγραφο με το σχεδιάγραμμα, που δείχνει τους πίνακες και τις σχέσεις μεταξύ τους, ονομάζεται *μοντέλο δεδομένων*.

Η μοντελοποίηση δεδομένων είναι μια σχετικά περίπλοκη διαδικασία και θα αναφερθούμε μόνο στις πιο βασικές έννοιες της μοντελοποίησης σχεσιακών δεδομένων σε αυτήν την ενότητα. Για περισσότερες λεπτομέρειες σχετικά με τη μοντελοποίηση δεδομένων, μπορείτε να ξεκινήσετε από:

http://en.wikipedia.org/wiki/Relational_model

Ας πούμε ότι στην εφαρμογή ανίχνευσης του Twitter, αντί να μετράμε απλώς τους φίλους ενός ατόμου, θέλαμε να κρατήσουμε μια λίστα με όλες τις εισερχόμενες σχέσεις, ώστε να μπορούμε να δημιουργήσουμε μια λίστα με όλους όσους ακολουθούν έναν συγκεκριμένο λογαριασμό.

Δεδομένου ότι όλοι θα έχουν, πιθανώς, πολλούς λογαριασμούς που τους ακολουθούν, δεν μπορούμε απλώς να προσθέσουμε μία στήλη στον πίνακα `Twitter`. Δημιουργούμε λοιπόν έναν νέο πίνακα που καταγράφει τα ζεύγη φίλων. Ο παρακάτω είναι ένας απλός τρόπος για να φτιάξετε έναν τέτοιο πίνακα:

```
CREATE TABLE Pals (from_friend TEXT, to_friend TEXT)
```

Κάθε φορά που συναντάμε ένα άτομο που ακολουθεί ο `drchuck`, εισάγουμε μια γραμμή της μορφής:

```
INSERT INTO Pals (from_friend,to_friend) VALUES ('drchuck', 'lhawthorn')
```

Καθώς επεξεργαζόμαστε τους 20 φίλους από τη ροή Twitter του drchuck, θα εισαγάγουμε 20 εγγραφές με το "drchuck" ως πρώτη παράμετρο, οπότε θα καταλήξουμε να αντιγράφουμε την ίδια συμβολοσειρά, πολλές φορές, στη βάση δεδομένων.

Αυτή η αντιγραφή δεδομένων συμβολοσειράς παραβιάζει μία από τις βασικές πρακτικές της *κανονικοποίησης βάσεων δεδομένων*, η οποία ουσιαστικά δηλώνει ότι δεν πρέπει, ποτέ, να αποθηκεύουμε τα ίδια δεδομένα συμβολοσειράς στη βάση δεδομένων, περισσότερες από μία φορές. Εάν χρειαζόμαστε τα δεδομένα περισσότερες από μία φορές, δημιουργούμε ένα αριθμητικό *κλειδί* για τα δεδομένα και αναφερόμαστε στα πραγματικά δεδομένα χρησιμοποιώντας αυτό το κλειδί.

Πρακτικά, μια συμβολοσειρά καταλαμβάνει πολύ περισσότερο χώρο από έναν ακέραιο στο δίσκο και στη μνήμη του υπολογιστή μας και απαιτείται περισσότερος χρόνος από τον επεξεργαστή για σύγκριση και ταξινόμηση. Εάν έχουμε μόνο μερικές εκατοντάδες καταχωρήσεις, ο χρόνος αποθήκευσης και επεξεργασίας δεν έχουν σημασία. Ωστόσο, εάν έχουμε ένα εκατομμύριο άτομα στη βάση δεδομένων μας και την πιθανότητα 100 εκατομμυρίων συνδέσμων φίλων, είναι σημαντικό να μπορούμε να σαρώνουμε δεδομένα όσο το δυνατόν γρηγορότερα.

Θα αποθηκεύσουμε τους λογαριασμούς μας στο Twitter σε έναν πίνακα με το όνομα People, αντί του πίνακα Twitter, που χρησιμοποιήθηκε στο προηγούμενο παράδειγμα. Ο πίνακας People έχει μια πρόσθετη στήλη, για την αποθήκευση του αριθμητικού κλειδιού που σχετίζεται με τη γραμμή αυτού του χρήστη του Twitter. Το SQLite έχει μια δυνατότητα που προσθέτει αυτόματα την τιμή κλειδιού σε κάθε γραμμή που εισάγουμε σε έναν πίνακα, χρησιμοποιώντας έναν ειδικό τύπο στήλης δεδομένων (INTEGER PRIMARY KEY).

Μπορούμε να δημιουργήσουμε τον πίνακα People με αυτήν την πρόσθετη στήλη id ως εξής:

```
CREATE TABLE People
    (id INTEGER PRIMARY KEY, name TEXT UNIQUE, retrieved INTEGER)
```

Παρατηρήστε ότι δεν διατηρούμε πλέον τον αριθμό φίλων, σε κάθε γραμμή του πίνακα People. Όταν επιλέγουμε το INTEGER PRIMARY KEY ως τον τύπο της στήλης id, υποδεικνύουμε ότι θα θέλαμε η SQLite να διαχειρίζεται αυτή τη στήλη και να εκχωρεί αυτόματα, σε κάθε γραμμή που εισάγουμε, ένα μοναδικό αριθμητικό κλειδί.

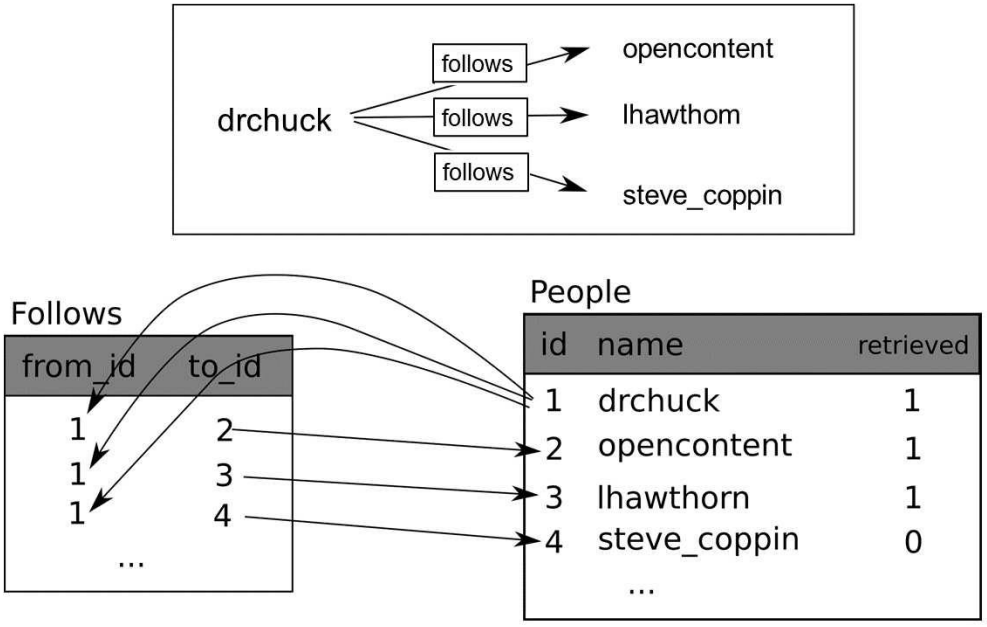
Προσθέτουμε επίσης τη δεσμευμένη λέξη UNIQUE για να επιβάλλουμε έναν περιορισμό, να υποδείξουμε ότι δεν θα επιτρέψουμε στην SQLite να εισάγει δύο γραμμές με την ίδια τιμή για το name.

Τώρα αντί να δημιουργήσουμε τον παραπάνω πίνακα Pals, δημιουργούμε έναν πίνακα που ονομάζεται Follows, με δύο στήλες ακεραίων, from_id και to_id και έναν περιορισμό στον πίνακα ότι ο *συνδυασμός* των from_id και to_id πρέπει να να είναι μοναδικός σε αυτόν τον πίνακα (δηλαδή, δεν μπορούμε να εισαγάγουμε διπλότυπες εγγραφές) στη βάση δεδομένων μας.

```
CREATE TABLE Follows
    (from_id INTEGER, to_id INTEGER, UNIQUE(from_id, to_id) )
```

Όταν προσθέτουμε περιορισμό UNIQUE στους πίνακές μας, επικοινωνούμε ένα σύνολο κανόνων που ζητάμε από τη βάση δεδομένων να τους επιβάλει, όταν προσπαθούμε να εισαγάγουμε εγγραφές. Δημιουργούμε αυτούς τους περιορισμούς στα προγράμματά μας για ευκολία, όπως θα δούμε σε λίγο. Οι περιορισμοί μας εμποδίζουν από το να κάνουμε λάθη και μας διευκολύνουν να γράψουμε μέρος του κώδικά μας.

Στην ουσία, δημιουργώντας αυτόν τον πίνακα Follows, διαμορφώνουμε μια "σχέση", όπου ένα άτομο "ακολουθεί" κάποιον άλλο και την αναπαριστάνουμε με ένα ζεύγος αριθμών που υποδεικνύουν ότι (α) τα άτομα είναι συνδεδεμένα και (β) την κατεύθυνση της σχέσης.



Εικόνα 15.4: Σχέσεις Μεταξύ Πινάκων

Χρήση πολλών πινάκων

Τώρα θα επαναλάβουμε το πρόγραμμα ανίχνευσης Twitter χρησιμοποιώντας δύο πίνακες, τα κύρια κλειδιά και τις αναφορές κλειδιών όπως περιγράφηκαν παραπάνω. Εδώ είναι ο κώδικας για τη νέα έκδοση του προγράμματος:

```
import urllib.request, urllib.parse, urllib.error
import twurl
import json
import sqlite3
import ssl

TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'

conn = sqlite3.connect('friends.sqlite')
cur = conn.cursor()

cur.execute('''CREATE TABLE IF NOT EXISTS People
            (id INTEGER PRIMARY KEY, name TEXT UNIQUE, retrieved INTEGER)''')
cur.execute('''CREATE TABLE IF NOT EXISTS Follows
            (from_id INTEGER, to_id INTEGER, UNIQUE(from_id, to_id))''')

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
    acct = input('Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση
με quit: ')
    if (acct == 'quit'): break
    if (len(acct) < 1):
        cur.execute('SELECT id, name FROM People WHERE retrieved=0 LIMIT 1')
        try:
            (id, acct) = cur.fetchone()
        except:
            print('Δεν βρέθηκαν λογαριασμοί Twitter που δεν έχουν ήδη
ανακτηθεί')
            continue
    else:
```

```
        cur.execute('SELECT id FROM People WHERE name = ? LIMIT 1',
                    (acct, ))
        try:
            id = cur.fetchone()[0]
        except:
            cur.execute('''INSERT OR IGNORE INTO People
                        (name, retrieved) VALUES (?, 0)''', (acct, ))
            conn.commit()
            if cur.rowcount != 1:
                print('Σφάλμα κατά την εισαγωγή του λογαριασμού:', acct)
                continue
            id = cur.lastrowid

    url = twurl.augment(TWITTER_URL, {'screen_name': acct, 'count': '100'})
    print('Ανάκτηση του', acct)
    try:
        connection = urllib.request.urlopen(url, context=ctx)
    except Exception as err:
        print('Η ανάκτηση απέτυχε ', err)
        break

    data = connection.read().decode()
    headers = dict(connection.getheaders())

    print('Απομένουν', headers['x-rate-limit-remaining'])

    try:
        js = json.loads(data)
    except:
        print('Δεν είναι δυνατή η ανάλυση του json')
        print(data)
        break

    # Εκσφαλμάτωση
    # print(json.dumps(js, indent=4))

    if 'users' not in js:
        print('Λήφθηκε λάθος JSON')
        print(json.dumps(js, indent=4))
```

```
        continue

    cur.execute('UPDATE People SET retrieved=1 WHERE name = ?', (acct, ))

    countnew = 0
    countold = 0
    for u in js['users']:
        friend = u['screen_name']
        print(friend)
        cur.execute('SELECT id FROM People WHERE name = ? LIMIT 1',
                    (friend, ))
        try:
            friend_id = cur.fetchone()[0]
            countold = countold + 1
        except:
            cur.execute('''INSERT OR IGNORE INTO People (name, retrieved)
                        VALUES (?, 0)''', (friend, ))
            conn.commit()
            if cur.rowcount != 1:
                print('Σφάλμα κατά την εισαγωγή λογαριασμού', friend)
                continue
            friend_id = cur.lastrowid
            countnew = countnew + 1
        cur.execute('''INSERT OR IGNORE INTO Follows (from_id, to_id)
                    VALUES (?, ?)''', (id, friend_id))
    print('Νέοι λογαριασμοί =', countnew, ' Επισκέφθηκαν εκ νέου =', countold)
    print('Απομένουν', headers['x-rate-limit-remaining'])
    conn.commit()
cur.close()
```

#Code: http://www.gr.py4e.com/code3/twfriends.py

Αυτό το πρόγραμμα αρχίζει να γίνεται λίγο περίπλοκο, αλλά δείχνει τα μοτίβα που πρέπει να ακολουθούμε όταν χρησιμοποιούμε ακέραια κλειδιά για σύνδεση πινάκων. Τα βασικά σημεία είναι:

1. Δημιουργήστε πίνακες με πρωτεύοντα κλειδιά και περιορισμούς.

2. Όταν έχουμε ένα λογικό κλειδί για ένα άτομο (π.χ. όνομα λογαριασμού) και χρειαζόμαστε την τιμή `id` για το άτομο, ανάλογα με το αν το άτομο βρίσκεται ήδη στον πίνακα `People` ή όχι, χρειάζεται: (1) να αναζητήστε το άτομο στον πίνακα

People και να ανακτήστε την τιμή id για το άτομο ή (2) να προσθέστε το άτομο στον πίνακα People και να κρατήσετε την τιμή id της νέας σειράς που προστέθηκε.

3. Εισαγάγετε τη γραμμή που καταγράφει τη σχέση Follows ("ακολουθεί").

Θα καλύψουμε καθένα από αυτά με τη σειρά του.

Περιορισμοί σε πίνακες βάσης δεδομένων

Καθώς σχεδιάζουμε τις δομές των πινάκων μας, μπορούμε να πούμε στο σύστημα βάσης δεδομένων ότι θα θέλαμε να επιβάλει μερικούς περιορισμούς. Αυτοί οι περιορισμοί μας βοηθούν να αποφύγουμε λάθη και εισαγωγή εσφαλμένων δεδομένων στους πίνακές μας, όταν δημιουργούμε τους πίνακές μας:

```
cur.execute('''CREATE TABLE IF NOT EXISTS People
    (id INTEGER PRIMARY KEY, name TEXT UNIQUE, retrieved INTEGER)''')
cur.execute('''CREATE TABLE IF NOT EXISTS Follows
    (from_id INTEGER, to_id INTEGER, UNIQUE(from_id, to_id))''')
```

Υποδεικνύουμε ότι η στήλη name στον πίνακα People πρέπει να είναι UNIQUE (ΜΟΝΑΔΙΚΗ). Υποδεικνύουμε επίσης ότι ο συνδυασμός των δύο αριθμών (from_id και to_id), σε κάθε γραμμή του πίνακα Follows, πρέπει να είναι μοναδικός. Αυτοί οι περιορισμοί μας εμποδίζουν να κάνουμε λάθη, όπως η προσθήκη της ίδιας σχέσης, περισσότερες από μία φορές.

Μπορούμε να εκμεταλλευτούμε αυτούς τους περιορισμούς στον ακόλουθο κώδικα:

```
cur.execute('''INSERT OR IGNORE INTO People (name, retrieved)
    VALUES ( ?, 0)''', ( friend, ) )
```

Προσθέτουμε τον όρο OR IGNORE στην εντολή INSERT για να υποδείξουμε ότι εάν αυτό το συγκεκριμένο INSERT προκαλεί παραβίαση του περιορισμού "Το name πρέπει να είναι μοναδικό", το σύστημα βάσης δεδομένων επιτρέπεται να αγνοήσει το INSERT. Χρησιμοποιούμε τον περιορισμό της βάσης δεδομένων ως δίχτυ ασφαλείας, για να βεβαιωθούμε ότι δεν θα κάνουμε άθελά μας κάτι λάθος.

Ομοίως, ο ακόλουθος κώδικας διασφαλίζει ότι δεν θα προσθέσουμε την ίδια ακριβώς σχέση Follows δύο φορές.

```
cur.execute('''INSERT OR IGNORE INTO Follows
    (from_id, to_id) VALUES (?, ?)''', (id, friend_id) )
```

Και πάλι, απλώς λέμε στη βάση δεδομένων να αγνοήσει το INSERT (ΕΙΣΑΓΩΓΗ), εάν παραβιάζει τον περιορισμό μοναδικότητας, που καθορίσαμε για τις γραμμές του Follows.

Ανάκτηση και/ή εισαγωγή εγγραφής

Όταν ζητάμε από τον χρήστη έναν λογαριασμό Twitter, εάν ο λογαριασμός υπάρχει, πρέπει να αναζητήσουμε την τιμή id του. Εάν ο λογαριασμός δεν υπάρχει ακόμα στον πίνακα People, πρέπει να εισαγάγουμε την εγγραφή και να λάβουμε την τιμή id της εισαγόμενης γραμμής.

Αυτό είναι ένα πολύ κοινό μοτίβο και γίνεται δύο φορές στο παραπάνω πρόγραμμα. Αυτός ο κώδικας δείχνει πώς αναζητούμε το id, για τον λογαριασμό ενός φίλου, όταν έχουμε εξαγάγει ένα screen_name από έναν κόμβο user στο ανακτηθέν Twitter JSON.

Δεδομένου ότι μετά από αρκετές εκτελέσεις, του παραπάνω κώδικα, θα είναι όλο και πιο πιθανό ο λογαριασμός να βρίσκεται ήδη στη βάση δεδομένων, πρώτα ελέγχουμε αν υπάρχει η εγγραφή στον People, χρησιμοποιώντας μια εντολή SELECT.

Αν όλα πάνε καλά[1] μέσα στην ενότητα try, ανακτούμε την εγγραφή χρησιμοποιώντας το fetchone() και στη συνέχεια ανακτάμε το πρώτο (και μοναδικό) στοιχείο της πλειάδας που επιστράφηκε και το αποθηκεύουμε στο friend_id.

Εάν το SELECT αποτύχει, ο κώδικας fetchone()[0] θα αποτύχει επίσης και ο έλεγχος θα μεταφερθεί στην ενότητα except.

```
    friend = u['screen_name']
    cur.execute('SELECT id FROM People WHERE name = ? LIMIT 1',
        (friend, ) )
    try:
        friend_id = cur.fetchone()[0]
        countold = countold + 1
    except:
        cur.execute('''INSERT OR IGNORE INTO People (name, retrieved)
            VALUES ( ?, 0)''', ( friend, ) )
```

[1] Γενικά, όταν μια πρόταση ξεκινά με "αν όλα πάνε καλά" θα διαπιστώσετε ότι ο κώδικας πρέπει να χρησιμοποιεί το try/except.

```
        conn.commit()
        if cur.rowcount != 1 :
            print('Error inserting account:',friend)
            continue
        friend_id = cur.lastrowid
        countnew = countnew + 1
```

Αν καταλήξουμε στον κώδικα του `except`, σημαίνει απλώς ότι η σειρά δεν βρέθηκε, επομένως πρέπει να εισαγάγουμε τη σειρά. Χρησιμοποιούμε `INSERT OR IGNORE` απλώς για να αποφύγουμε σφάλματα και στη συνέχεια καλούμε το `commit()`, για να αναγκάσουμε τη βάση δεδομένων να ενημερωθεί. Αφού ολοκληρωθεί η εγγραφή, μπορούμε να ελέγξουμε το `cur.rowcount` για να δούμε πόσες σειρές επηρεάστηκαν. Εφόσον προσπαθούμε να εισαγάγουμε μία μόνο σειρά, εάν ο αριθμός των επηρεαζόμενων σειρών είναι κάτι διαφορετικό από 1, είναι σφάλμα.

Εάν το `INSERT` είναι επιτυχές, μπορούμε να κοιτάξουμε το `cur.lastrowid` για να ελέγξουμε ποια τιμή έχει εκχωρήσει η βάση δεδομένων στη στήλη `id`, στη νεοδημιουργηθήσα γραμμή.

Αποθήκευση της σχέσης φίλου

Από τη στιγμή που ξέρετε την τιμή κλειδιού, τόσο για τον χρήστη του Twitter όσο και για τον φίλο, στο JSON, είναι απλό να εισαγάγετε τους δύο αριθμούς στον πίνακα `Follows με τον ακόλουθο κώδικα:

```
cur.execute('INSERT OR IGNORE INTO Follows (from_id, to_id) VALUES (?,
?)',
    (id, friend_id) )
```

Σημειώστε ότι αφήνουμε τη βάση δεδομένων να φροντίσει για την αποφυγή "διπλής εισαγωγής" μιας σχέσης, δημιουργώντας τον πίνακα με έναν περιορισμό μοναδικότητας και, στη συνέχεια, προσθέτοντας `OR IGNORE` στην εντολή `INSERT`.

Ακολουθεί ένα δείγμα εκτέλεσης αυτού του προγράμματος:

```
Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση με quit:
Δεν βρέθηκαν λογαριασμοί Twitter που δεν έχουν ήδη ανακτηθεί
Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση με quit:
drchuck
```

```
Ανάκτηση του http://api.twitter.com/1.1/friends ...
Νέοι λογαριασμοί = 20  Επισκέφθηκαν εκ νέου = 0
Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση με quit:
Ανάκτηση του http://api.twitter.com/1.1/friends ...
Νέοι λογαριασμοί = 17  Επισκέφθηκαν εκ νέου = 3
Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση με quit:
Ανάκτηση του http://api.twitter.com/1.1/friends ...
Νέοι λογαριασμοί = 17  Επισκέφθηκαν εκ νέου = 3
Εισαγάγετε έναν λογαριασμό Twitter ή τερματίστε την εκτέλεση με quit: quit
```

Ξεκινήσαμε με τον λογαριασμό drchuck και μετά αφήσαμε το πρόγραμμα να επιλέξει αυτόματα τους επόμενους δύο λογαριασμούς για ανάκτηση και προσθήκη στη βάση δεδομένων μας.

Ακολουθούν οι πρώτες λίγες σειρές στους πίνακες People και Follows, μετά την ολοκλήρωση αυτής της εκτέλεσης:

```
People:
(1, 'drchuck', 1)
(2, 'opencontent', 1)
(3, 'lhawthorn', 1)
(4, 'steve_coppin', 0)
(5, 'davidkocher', 0)
55 γραμμές.
Follows:
(1, 2)
(1, 3)
(1, 4)
(1, 5)
(1, 6)
60 γραμμές.
```

Μπορείτε να δείτε τα πεδία id, name και visited στον πίνακα People και βλέπετε τους αριθμούς και των δύο άκρων των σχέσεων στον πίνακα Follows. Στον πίνακα People, μπορούμε να δούμε ότι έχουμε επισκεφθεί τα τρία πρώτα άτομα και τα δεδομένα τους έχουν ανακτηθεί. Τα δεδομένα στον πίνακα Follows υποδεικνύουν ότι ο drchuck (χρήστης 1) είναι φίλος με όλα τα άτομα που εμφανίζονται στις πρώτες πέντε σειρές. Αυτό είναι λογικό, γιατί τα πρώτα δεδομένα που ανακτήσαμε και αποθηκεύσαμε ήταν

οι φίλοι Twitter του drchuck. Εάν επρόκειτο να εκτυπώσετε περισσότερες σειρές από τον πίνακα Follows, θα βλέπατε και τους φίλους των χρηστών 2 και 3.

Τριών ειδών κλειδιά

Τώρα που αρχίσαμε να χτίζουμε ένα μοντέλο δεδομένων, τοποθετώντας τα δεδομένα μας σε πολλούς συνδεδεμένους πίνακες και συνδέοντας τις σειρές αυτών των πινάκων χρησιμοποιώντας *κλειδιά*, πρέπει να δούμε κάποια ορολογία γύρω από τα κλειδιά. Υπάρχουν γενικά τρία είδη κλειδιών που χρησιμοποιούνται σε ένα μοντέλο βάσης δεδομένων.

- Ένα *λογικό κλειδί (logical key)* είναι ένα κλειδί που μπορεί να χρησιμοποιηθεί στον "πραγματικό κόσμο" για την αναζήτηση μιας σειράς. Στο μοντέλο δεδομένων του παραδείγματός μας, το πεδίο name είναι ένα λογικό κλειδί. Είναι το όνομα με το οποίο εμφανίζεται ο χρήστης (screen name) και όντως αναζητούμε τη σειρά ενός χρήστη αρκετές φορές στο πρόγραμμα χρησιμοποιώντας το πεδίο name. Συχνά θα διαπιστώσετε ότι είναι λογικό να προσθέσετε έναν περιορισμό UNIQUE σε ένα λογικό κλειδί. Δεδομένου ότι το λογικό κλειδί είναι ο τρόπος με τον απoίο αναζητούμε μια σειρά στην πραγματική ζωή, δεν έχει νόημα να επιτρέπουμε πολλές σειρές με την ίδια τιμή στον πίνακα.
- Ένα *πρωτεύον κλειδί (primary key)* είναι συνήθως ένας αριθμός που εκχωρείται αυτόματα από τη βάση δεδομένων. Γενικά δεν έχει νόημα εκτός προγράμματος και χρησιμοποιείται μόνο για τη σύνδεση σειρών από διαφορετικούς πίνακες, μεταξύ τους. Όταν θέλουμε να αναζητήσουμε μια σειρά σε έναν πίνακα, συνήθως η αναζήτηση της σειράς, χρησιμοποιώντας το πρωτεύον κλειδί είναι ο πιο γρήγορος τρόπος για να βρείτε τη σειρά. Δεδομένου ότι τα πρωτεύοντα κλειδιά είναι ακέραιοι αριθμοί, καταλαμβάνουν πολύ λίγο χώρο αποθήκευσης και μπορούν να συγκριθούν ή να ταξινομηθούν πολύ γρήγορα. Στο μοντέλο δεδομένων μας, το πεδίο id είναι ένα παράδειγμα πρωτεύοντος κλειδιού.
- Ένα *ξένο κλειδί (foreign key)* είναι συνήθως ένας αριθμός που δείχνει το πρωτεύον κλειδί μιας συσχετισμένης σειράς σε διαφορετικό πίνακα. Ένα παράδειγμα ξένου κλειδιού στο μοντέλο δεδομένων μας είναι το from_id.

Χρησιμοποιούμε μια σύμβαση ονομασίας που πάντα το όνομα πεδίου του πρωτεύοντος κλειδιού id και προσαρτάμε το επίθημα _id σε οποιοδήποτε όνομα πεδίου που είναι ξένο κλειδί.

Χρήση JOIN για ανάκτηση δεδομένων

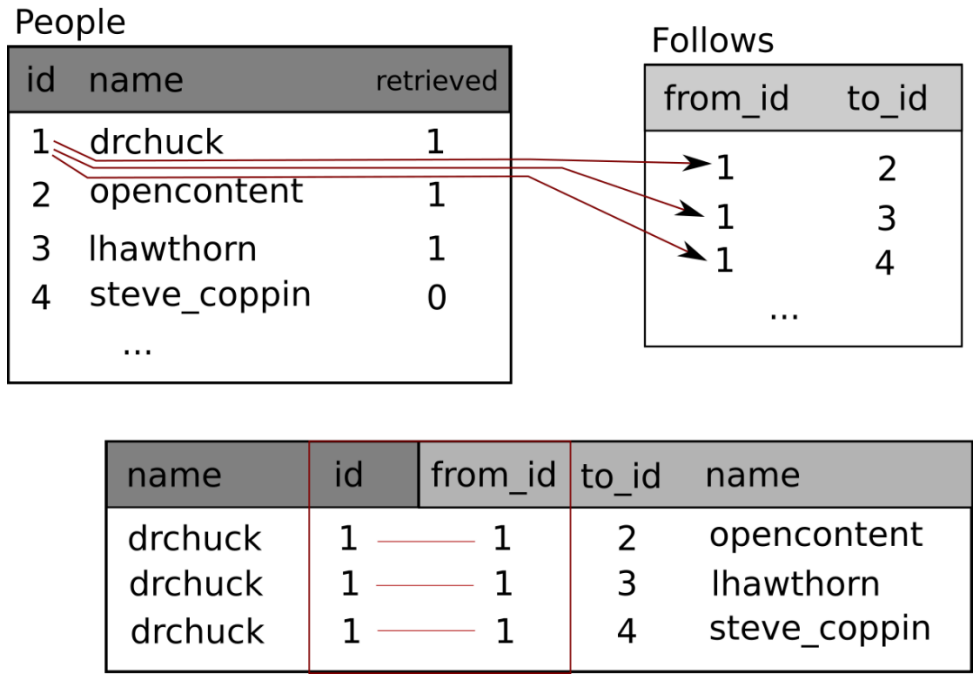
Τώρα που έχουμε ακολουθήσει τους κανόνες κανονικοποίησης της βάσης δεδομένων και τα δεδομένα χωρίζονται σε δύο πίνακες, συνδεδεμένα μεταξύ τους χρησιμοποιώντας πρωτεύοντα και ξένα κλειδιά, πρέπει να είμαστε σε θέση να δημιουργήσουμε ένα `SELECT` που επανασυναρμολογεί τα δεδομένα στους πίνακες.

Η SQL χρησιμοποιεί τον όρο `JOIN` για να επανασυνδέσει αυτούς τους πίνακες. Στον όρο `JOIN` καθορίζετε τα πεδία που χρησιμοποιούνται για την επανασύνδεση των σειρών των πινάκων.

Το παρακάτω είναι ένα παράδειγμα `SELECT` με ρήτρα `JOIN`:

```
SELECT * FROM Follows JOIN People
    ON Follows.from_id = People.id WHERE People.id = 1
```

Ο όρος `JOIN` υποδηλώνει ότι τα πεδία που επιλέγουμε διασχίζουν και τους πίνακες `Follows` και `People`. Η ρήτρα `ON` υποδεικνύει πώς θα ενωθούν οι δύο πίνακες: Πάρε τις σειρές από το `Follows` και προσθέστε τη σειρά του `People`, όπου το πεδίο `from_id` στο `Follows` είναι το ίδιο με την τιμή `id` του πίνακα `People`.



Εικόνα 15.5: Σύνδεση πινάκων με χρήση JOIN

Το αποτέλεσμα του JOIN είναι να δημιουργηθούν πολύ μεγάλες "meta-γραμμές" οι οποίες έχουν και τα δύο πεδία από τον `People` και τα αντίστοιχα πεδία από τον `Follows`. Όπου υπάρχουν περισσότερες από μία αντιστοιχίσεις μεταξύ του πεδίου `id` από τον `People` και του `from_id` από τον `Follows`, τότε το JOIN δημιουργεί μια meta-

γραμμή για *κάθε ένα* από τα αντίστοιχα ζεύγη σειρών, αντιγράφοντας δεδομένα όπως απαιτείται.

Ο παρακάτω κώδικας δείχνει τα δεδομένα που θα έχουμε στη βάση δεδομένων μετά την εκτέλεση του προγράμματος πολλαπλών πινάκων Twitter spider (παραπάνω) πολλές φορές.

```
import sqlite3

conn = sqlite3.connect('friends.sqlite')
cur = conn.cursor()

cur.execute('SELECT * FROM People')
count = 0
print('People:')
for row in cur:
    if count < 5: print(row)
    count = count + 1
print(count, 'γραμμές.')

cur.execute('SELECT * FROM Follows')
count = 0
print('Follows:')
for row in cur:
    if count < 5: print(row)
    count = count + 1
print(count, 'γραμμές.')

cur.execute('''SELECT * FROM Follows JOIN People
            ON Follows.to_id = People.id
            WHERE Follows.from_id = 2''')
count = 0
print('Συνδέσεις για το id=2:')
for row in cur:
    if count < 5: print(row)
    count = count + 1
print(count, 'γραμμές.')

cur.close()
```

#Code: http://www.gr.py4e.com/code3/twjoin.py

Σε αυτό το πρόγραμμα, καταργούμε πρώτα τους πίνακες People και Follows και, στη συνέχεια, απορρίπτουμε ένα υποσύνολο των δεδομένων στους πίνακες που είναι ενωμένοι μεταξύ τους.

Εδώ είναι η έξοδος του προγράμματος:

```
python twjoin.py
People:
(1, 'drchuck', 1)
(2, 'opencontent', 1)
(3, 'lhawthorn', 1)
(4, 'steve_coppin', 0)
(5, 'davidkocher', 0)
55 γραμμές.
Follows:
(1, 2)
(1, 3)
(1, 4)
(1, 5)
(1, 6)
60 γραμμές.
Συνδέσεις για το id=2:
(2, 1, 1, 'drchuck', 1)
(2, 28, 28, 'cnxorg', 0)
(2, 30, 30, 'kthanos', 0)
(2, 102, 102, 'SomethingGirl', 0)
(2, 103, 103, 'ja_Pac', 0)
20 γραμμές.
```

Βλέπετε τις στήλες από τους πίνακες People και Follows και το τελευταίο σύνολο σειρών είναι το αποτέλεσμα του SELECT με τον όρο JOIN.

Στην τελευταία επιλογή, αναζητούμε λογαριασμούς που είναι φίλοι του "opencontent" (δηλαδή, People.id=2).

Σε κάθε μία από τις "meta-γραμμές" στην τελευταία επιλογή, οι δύο πρώτες στήλες είναι από τον πίνακα Follows ακολουθούμενες από τις στήλες τρία έως πέντε του πίνακα People. Μπορείτε επίσης να δείτε ότι η δεύτερη στήλη (Follows.to_id) ταιριάζει με την τρίτη στήλη (People.id) σε κάθε μία από τις ενωμένες "meta-γραμμές".

Περίληψη

Αυτό το κεφάλαιο έχει καλύψει πολλά, για να σας δώσει μια επισκόπηση των βασικών στοιχείων της χρήσης μιας βάσης δεδομένων στην Python. Είναι πιο περίπλοκο να γράψετε τον κώδικα για να χρησιμοποιήσετε μια βάση δεδομένων για την αποθήκευση δεδομένων, από τα λεξικά Python ή τα επίπεδα αρχεία, επομένως δεν υπάρχει λόγος να χρησιμοποιήσετε μια βάση δεδομένων, εκτός εάν η εφαρμογή σας χρειάζεται πραγματικά τις δυνατότητες μιας βάσης δεδομένων. Οι περιπτώσεις όπου μια βάση δεδομένων μπορεί να είναι αρκετά χρήσιμη είναι: (1) όταν η εφαρμογή σας χρειάζεται να κάνει πολλές μικρές τυχαίες ενημερώσεις μέσα σε ένα μεγάλο σύνολο δεδομένων, (2) όταν τα δεδομένα σας είναι τόσο μεγάλα που δεν χωρούν σε ένα λεξικό και πρέπει να αναζητήσετε πληροφορίες επαναληπτικά ή (3) όταν έχετε μια χρονοβόρα διαδικασία, που θέλετε να μπορείτε να σταματήσετε και να επανεκκινήσετε, και να διατηρήσετε τα δεδομένα από τη μια εκτέλεση στην άλλη.

Μπορείτε να δημιουργήσετε μια απλή βάση δεδομένων με έναν μόνο πίνακα για να ταιριάζει σε πολλές ανάγκες εφαρμογών, αλλά τα περισσότερα προβλήματα απαιτούν πολλούς πίνακες και συνδέσμους/σχέσεις μεταξύ σειρών σε διαφορετικούς πίνακες. Όταν ξεκινάτε να δημιουργείτε σχέσεις μεταξύ πινάκων, είναι σημαντικό να τις σχεδιάσετε προσεκτικά και να ακολουθήσετε τους κανόνες κανονικοποίησης της βάσης δεδομένων, για να αξιοποιήσετε με τον καλύτερο τρόπο τις δυνατότητες της βάσης δεδομένων. Δεδομένου ότι το κύριο κίνητρο για τη χρήση μιας βάσης δεδομένων είναι ότι έχετε μεγάλο όγκο δεδομένων για επεξεργασία, είναι σημαντικό να μοντελοποιήσετε τα δεδομένα σας αποτελεσματικά, ώστε τα προγράμματά σας να εκτελούνται όσο το δυνατόν γρηγορότερα.

Εκσφαλμάτωση

Ένα κοινό μοτίβο, όταν αναπτύσσετε ένα πρόγραμμα Python για να συνδεθείτε σε μια βάση δεδομένων SQLite, θα είναι η εκτέλεση ενός προγράμματος Python και ο έλεγχος των αποτελεσμάτων χρησιμοποιώντας το πρόγραμμα περιήγησης βάσης δεδομένων για SQLite (Database Browser for SQLite). Το πρόγραμμα περιήγησης σάς επιτρέπει να ελέγχετε γρήγορα εάν το πρόγραμμά σας λειτουργεί σωστά.

Πρέπει να είστε προσεκτικοί γιατί το SQLite φροντίζει να εμποδίζει την αλλαγή των ίδιων δεδομένων από δύο προγράμματα ταυτόχρονα. Για παράδειγμα, εάν ανοίξετε μια βάση δεδομένων στο πρόγραμμα περιήγησης και κάνετε μια αλλαγή στη βάση δεδομένων και δεν έχετε πατήσει ακόμη το κουμπί "αποθήκευση" στο πρόγραμμα

περιήγησης, το πρόγραμμα περιήγησης "κλειδώνει" το αρχείο βάσης δεδομένων και εμποδίζει οποιοδήποτε άλλο πρόγραμμα να έχει πρόσβαση στο αρχείο. Συγκεκριμένα, το πρόγραμμα Python σας δεν θα μπορεί να έχει πρόσβαση στο αρχείο εάν είναι κλειδωμένο.

Επομένως, μια λύση είναι να βεβαιωθείτε ότι είτε κλείσατε το πρόγραμμα περιήγησης της βάσης δεδομένων είτε χρησιμοποιήσετε το μενού *Αρχείο* για να κλείσετε τη βάση δεδομένων στο πρόγραμμα περιήγησης πριν επιχειρήσετε να αποκτήσετε πρόσβαση στη βάση δεδομένων από την Python, για να αποφύγετε το πρόβλημα της αποτυχίας του κώδικα Python, επειδή η βάση δεδομένων είναι κλειδωμένη.

Γλωσσάριο

ευρετήριο : Πρόσθετα δεδομένα, που το λογισμικό της βάσης δεδομένων διατηρεί ως γραμμές και τα εισάγει σε έναν πίνακα για να πραγματοποιεί τις αναζητήσεις πιο γρήγορα.

κανονικοποίηση : Σχεδιασμός ενός μοντέλου δεδομένων έτσι ώστε να μην γίνεται αναπαραγωγή δεδομένων. Αποθηκεύουμε κάθε στοιχείο δεδομένων σε ένα σημείο της βάσης δεδομένων και το αναφέρουμε αλλού χρησιμοποιώντας ένα ξένο κλειδί.

κέρσορας : Ένας κέρσορας σάς επιτρέπει να εκτελείτε εντολές SQL σε μια βάση δεδομένων και να ανακτάτε δεδομένα από τη βάση δεδομένων. Ο κέρσορας είναι παρόμοιος με μια υποδοχή ή έναν περιγραφέα αρχείου για συνδέσεις δικτύου και αρχεία, αντίστοιχα.

λογικό κλειδί : Ένα κλειδί που χρησιμοποιεί ο "έξω κόσμος" για να αναζητήσει μια συγκεκριμένη σειρά. Για παράδειγμα, σε έναν πίνακα λογαριασμών χρηστών, η διεύθυνση email ενός ατόμου μπορεί να είναι καλός υποψήφιος ως λογικό κλειδί για τα δεδομένα του χρήστη.

ξένο κλειδί - foreign key : Ένα αριθμητικό κλειδί που δείχνει το πρωτεύον κλειδί μιας γραμμής ενός άλλου πίνακα. Τα ξένα κλειδιά δημιουργούν σχέσεις μεταξύ γραμμών, που είναι αποθηκευμένες σε διαφορετικούς πίνακες.

περιορισμός : Όταν λέμε στη βάση δεδομένων να επιβάλει έναν περιορισμός σε ένα πεδίο ή μια γραμμή ενός πίνακα. Ένας κοινός περιορισμός είναι να επιμείνουμε ότι δεν μπορούν να υπάρχουν διπλότυπες τιμές σε ένα συγκεκριμένο πεδίο (δηλαδή, όλες οι τιμές πρέπει να είναι μοναδικές).

πλειάδα : Μια μεμονωμένη καταχώρηση σε έναν πίνακα βάσης δεδομένων, που είναι ένα σύνολο χαρακτηριστικών. Συνήθως αναφέρεται ως "γραμμή".

πρόγραμμα περιήγησης βάσης δεδομένων : Ένα κομμάτι λογισμικού που σας επιτρέπει να συνδεθείτε απευθείας σε μια βάση δεδομένων και να χειριστείτε τη βάση δεδομένων απευθείας χωρίς να γράψετε πρόγραμμα.

πρωτεύων κλειδί : Ένα αριθμητικό κλειδί, που εκχωρείται σε κάθε γραμμή και χρησιμοποιείται για να αναφερθούμε σε μια γραμμή ενός πίνακα από έναν άλλο πίνακα. Συχνά η βάση δεδομένων ρυθμίζεται ώστε να εκχωρεί αυτόματα πρωτεύοντα κλειδιά, καθώς εισάγονται γραμμές.

σχέση : Μια περιοχή μέσα σε μια βάση δεδομένων που περιέχει πλειάδες και χαρακτηριστικά. Συνήθως ονομάζεται "πίνακας".

χαρακτηριστικό : Μία από τις τιμές μέσα σε μια πλειάδα. Συνήθως ονομάζεται "στήλη" ή "πεδίο".

Κεφάλαιο 16

Οπτικοποίηση δεδομένων

Μέχρι στιγμής μαθαίνουμε τη γλώσσα Python και πώς να χρησιμοποιούμε την Python, το δίκτυο και τις βάσεις δεδομένων, για να χειριζόμαστε δεδομένα.

Σε αυτό το κεφάλαιο, ρίχνουμε μια ματιά σε τρεις ολοκληρωμένες εφαρμογές που συγκεντρώνουν όλα όσα μάθαμε, για τη διαχείριση και την οπτικοποίηση δεδομένων. Μπορείτε να χρησιμοποιήσετε αυτές τις εφαρμογές ως δείγμα κώδικα για να ξεκινήσετε την επίλυση ενός πραγματικού προβλήματος.

Κάθε μία από τις εφαρμογές είναι ένα αρχείο ZIP, που μπορείτε να κατεβάσετε και να εξαγάγετε στον υπολογιστή σας καθώς και να το εκτελέσετε.

Δημιουργία ενός OpenStreetMap από γεωκωδικοποιημένα δεδομένα

Σε αυτό το πρότζεκτ, χρησιμοποιούμε το API γεωκωδικοποίησης OpenStreetMap για να καθαρίσουμε ορισμένες γεωγραφικές τοποθεσίες, που έχουν εισαχθεί από τους χρήστες με ονόματα πανεπιστημίων και, στη συνέχεια, να τοποθετήσουμε τα δεδομένα σε ένα πραγματικό OpenStreetMap.

Εικόνα 16.1: Ένα OpenStreetMap

Για να ξεκινήσετε, κατεβάστε την εφαρμογή από:

www.gr.py4e.com/code3/opengeo.zip

Το πρώτο πρόβλημα που πρέπει να λύσουμε είναι ότι αυτά τα API γεωκωδικοποίησης είναι περιορισμένου ρυθμού, με έναν ορισμένο αριθμό αιτημάτων ανά ημέρα. Εάν έχετε πολλά δεδομένα, ίσως χρειαστεί να διακόψετε και να επανεκκινήσετε τη διαδικασία αναζήτησης αρκετές φορές. Έτσι χωρίζουμε το πρόβλημα σε δύο φάσεις.

Σε πρώτη φάση παίρνουμε τα δεδομένα εισόδου τη "έρευνάς" μας από το αρχείο *where.data*, τα διαβάζουμε μία γραμμή τη φορά, ανακτούμε τις γεωκωδικοποιημένες πληροφορίες από την Google και τις αποθηκεύουμε σε μια βάση δεδομένων, τη *geodata.sqlite*. Προτού χρησιμοποιήσουμε το API γεωκωδικοποίησης, για κάθε τοποθεσία που έχει εισαγάγει ο χρήστης, απλώς ελέγχουμε να δούμε αν έχουμε ήδη αποθηκεύσει τα δεδομένα για τη συγκεκριμένη γραμμή εισαγωγής. Η βάση δεδομένων λειτουργεί ως τοπική "κρυφή μνήμη" των δεδομένων γεωκωδικοποίησής μας, για να διασφαλίσουμε ότι δεν θα ζητήσουμε ποτέ από την Google τα ίδια δεδομένα δύο φορές.

Μπορείτε να επανεκκινήσετε τη διαδικασία ανά πάσα στιγμή, αφαιρώντας το αρχείο *geodata.sqlite*.

Εκτελέστε το πρόγραμμα *geoload.py*. Αυτό το πρόγραμμα θα διαβάσει τις γραμμές εισόδου στο *where.data* και για κάθε γραμμή θα ελέγξει αν βρίσκεται ήδη στη βάση δεδομένων. Εάν δεν έχουμε τα δεδομένα για την συγκεκριμένη τοποθεσία, θα καλέσει το API γεωκωδικοποίησης για να ανακτήσει τα δεδομένα και να τα αποθηκεύσει στη βάση δεδομένων.

Ακολουθεί ένα δείγμα εκτέλεσης, ενώ έχουν ήδη εισαχθει κάποια δεδομένα στη βάση δεδομένων:

```
Βρέθηκε στη βάση δεδομένων AGH University of Science and Technology

Βρέθηκε στη βάση δεδομένων Academy of Fine Arts Warsaw Poland

Βρέθηκε στη βάση δεδομένων American University in Cairo

Βρέθηκε στη βάση δεδομένων Arizona State University

Βρέθηκε στη βάση δεδομένων Athens Information Technology

Ανάκτηση της https://py4e-data.dr-chuck.net/
   opengeo?q=BITS+Pilani
```

```
Ανακτήθηκαν 794 χαρακτήρες {"type":"FeatureColl

Ανάκτηση της https://py4e-data.dr-chuck.net/
   opengeo?q=Babcock+University
Ανακτήθηκαν 760 χαρακτήρες {"type":"FeatureColl

Ανάκτηση της https://py4e-data.dr-chuck.net/
   opengeo?q=Banaras+Hindu+University
Ανακτήθηκαν 866 χαρακτήρες {"type":"FeatureColl
...
```

Οι πρώτες πέντε τοποθεσίες βρίσκονται ήδη στη βάση δεδομένων και έτσι παραλείπονται. Το πρόγραμμα σαρώνει μέχρι το σημείο όπου βρίσκει νέες τοποθεσίες και ξεκινά την ανάκτησή τους.

Το πρόγραμμα *geoload.py* μπορεί να διακοπεί ανά πάσα στιγμή και υπάρχει ένας μετρητής που μπορείτε να χρησιμοποιήσετε για να περιορίσετε τον αριθμό των κλήσεων στο API γεωκωδικοποίησης για κάθε εκτέλεση. Δεδομένου ότι το *where.data* έχει μόνο μερικές εκατοντάδες στοιχεία δεδομένων, δεν θα πρέπει να φτάσετε το ημερήσιο όριο ρυθμού, αλλά εάν είχατε περισσότερα δεδομένα, ενδέχεται να χρειαστούν αρκετές εκτελέσεις, σε αρκετές ημέρες, για να φτάσει η βάση δεδομένων σας να έχει όλα τα γεωκωδικοποιημένα δεδομένα για τα δεδομένα εισαγωγής σας.

Αφού φορτώσετε ορισμένα δεδομένα στο *geodata.sqlite*, μπορείτε να οπτικοποιήσετε τα δεδομένα χρησιμοποιώντας το πρόγραμμα *geodump.py*. Αυτό το πρόγραμμα διαβάζει τη βάση δεδομένων και δημιουργεί το αρχείο *where.js* με τη θέση, το γεωγραφικό πλάτος και μήκος, με τη μορφή εκτελέσιμου κώδικα JavaScript.

Μια εκτέλεση του προγράμματος *geodump.py* είναι η εξής:

```
AGH University of Science and Technology, Czarnowiejska,
Czarna Wieś, Krowodrza, Kraków, Lesser Poland
Voivodeship, 31-126, Poland 50.0657 19.91895

Academy of Fine Arts, Krakowskie Przedmieście,
Northern Śródmieście, Śródmieście, Warsaw, Masovian
Voivodeship, 00-046, Poland 52.239 21.0155
...
Υπήρχαν 260 εγραφές στο where.js
```

```
Ανοίξτε το where.html για να προβάλετε τα δεδομένα σε ένα πρόγραμμα
περιήγησης
```

Το αρχείο *where.html* αποτελείται από HTML και JavaScript, για την οπτικοποίηση ενός χάρτη Google. Διαβάζει τα πιο πρόσφατα δεδομένα στο *where.js* για να οπτικοποιήσει τα δεδομένα. Ακολουθεί η μορφή του αρχείου *where.js*:

```
myData = [
[50.0657,19.91895,
'AGH University of Science and Technology, Czarnowiejska,
Czarna Wieś, Krowodrza, Kraków, Lesser Poland
Voivodeship, 31-126, Poland '],
[52.239,21.0155,
'Academy of Fine Arts, Krakowskie Przedmieściee,
Śródmieście Północne, Śródmieście, Warsaw,
Masovian Voivodeship, 00-046, Poland'],
   ...
];
```

Αυτή είναι μια μεταβλητή JavaScript που περιέχει μια λίστα λιστών. Η σύνταξη για τις σταθερές λίστας JavaScript είναι πολύ παρόμοια με την Python, επομένως η σύνταξη θα πρέπει να σας είναι οικεία.

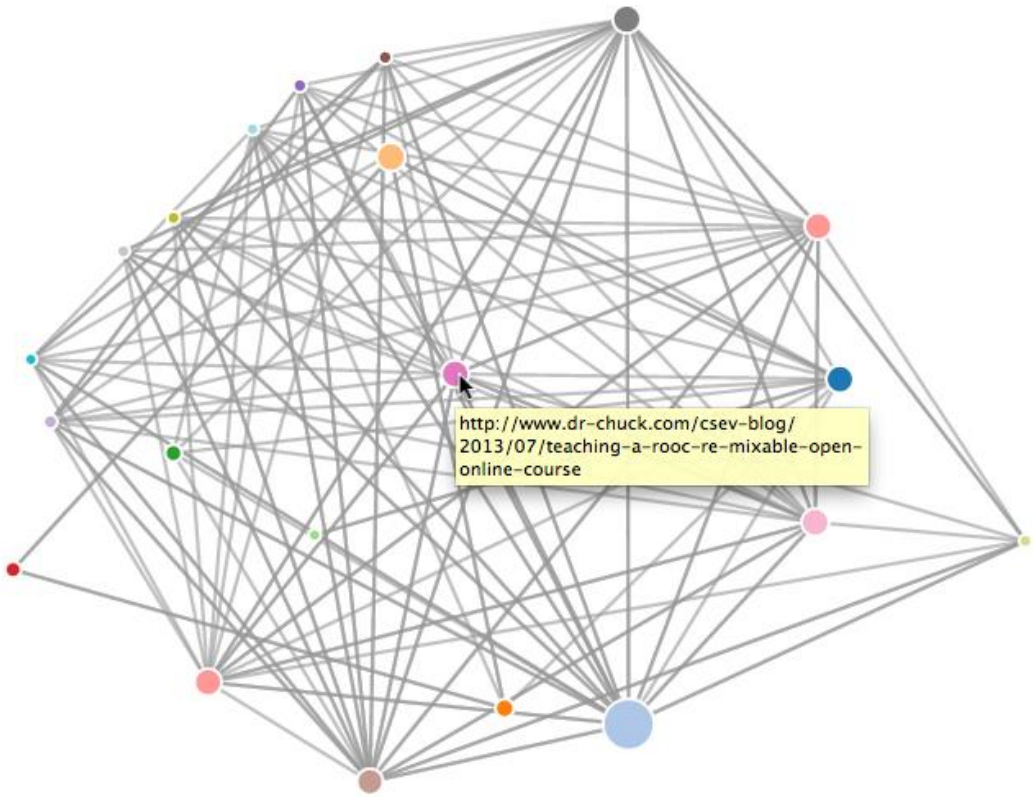
Απλώς ανοίξτε το *where.html*, σε ένα πρόγραμμα περιήγησης, για να δείτε τις τοποθεσίες. Μπορείτε να τοποθετήσετε τον δείκτη του ποντικιού πάνω από κάθε καρφίτσα του χάρτη, για να βρείτε την τοποθεσία που επέστρεψε το API γεωκωδικοποίησης για την είσοδο που εισήγαγε ο χρήστης. Εάν δεν μπορείτε να δείτε τα δεδομένα όταν ανοίγετε το αρχείο *where.html*, ίσως χρειαστεί να ελέγξετε το JavaScript ή την κονσόλα προγραμματιστή, για το πρόγραμμα περιήγησής σας.

Οπτικοποίηση δικτύων και διασυνδέσεων

Σε αυτήν την εφαρμογή, θα εκτελέσουμε ορισμένες από τις λειτουργίες μιας μηχανής αναζήτησης. Αρχικά θα δημιουργήσουμε ένα μικρό υποσύνολο του ιστού και θα εκτελέσουμε μια απλοποιημένη έκδοση του αλγόριθμου κατάταξης σελίδων της Google, για να προσδιορίσουμε ποιες σελίδες είναι πιο συνδεδεμένες και, στη συνέχεια, θα οπτικοποιήσουμε την κατάταξη σελίδας και τη συνδεσιμότητα της μικρής μας γωνιάς του ιστού. Θα χρησιμοποιήσουμε τη βιβλιοθήκη οπτικοποίησης JavaScript D3 http://d3js.org/, για να παράγουμε την έξοδο οπτικοποίησης.

Μπορείτε να κατεβάσετε και να εξαγάγετε αυτήν την εφαρμογή από:

www.gr.py4e.com/code3/pagerank.zip



Εικόνα 16.2: Κατάταξη σελίδας

Το πρώτο πρόγραμμα προγράμματος (*spider.py*) ανιχνεύει έναν ιστότοπο και καταχωρεί μια σειρά σελίδων στη βάση δεδομένων (*spider.sqlite*), καταγράφοντας τους συνδέσμους μεταξύ των σελίδων. Μπορείτε να επανεκκινήσετε τη διαδικασία ανά πάσα στιγμή, αφαιρώντας το αρχείο *spider.sqlite* και εκτελώντας ξανά το *spider.py*.

```
Εισαγάγετε τη διεύθυνση url ιστού ή enter: http://www.dr-chuck.com/
['http://www.dr-chuck.com']
Πόσες σελίδες:2
1 http://www.dr-chuck.com/ 12
2 http://www.dr-chuck.com/csev-blog/ 57
Πόσες σελίδες:
```

Σε αυτό το δείγμα εκτέλεσης, ζητήσαμε να ανιχνεύσει έναν ιστότοπο και να ανακτήσει δύο σελίδες. Εάν κάνετε επανεκκίνηση του προγράμματος και του πείτε να ανιχνεύσει περισσότερες σελίδες, δεν θα ανιχνεύσει ξανά καμία σελίδα που υπάρχει ήδη στη βάση δεδομένων. Με την επανεκκίνηση πηγαίνει σε μια τυχαία σελίδα που δεν ανιχνεύθηκε και ξεκινά από εκεί. Έτσι, κάθε διαδοχική εκτέλεση του *spider.py* λειτουργεί προσθετικά.

```
Εισαγάγετε τη διεύθυνση url ιστού ή enter: http://www.dr-chuck.com/
['http://www.dr-chuck.com']
Πόσες σελίδες:3
3 http://www.dr-chuck.com/csev-blog 57
4 http://www.dr-chuck.com/dr-chuck/resume/speaking.htm 1
5 http://www.dr-chuck.com/dr-chuck/resume/index.htm 13
Πόσες σελίδες:
```

Μπορείτε να έχετε πολλαπλά σημεία εκκίνησης στην ίδια βάση δεδομένων — εντός του προγράμματος, αυτά ονομάζονται "ιστοί". Το spider επιλέγει τυχαία μεταξύ όλων των συνδέσμων που δεν έχουν επισκεφτεί, σε όλους τους ιστούς, για να καθορίσει την επόμενη σελίδα προς ανίχνευση.

Εάν θέλετε να καταργήσετε τα περιεχόμενα του αρχείου *spider.sqlite*, μπορείτε να εκτελέσετε το *spdump.py* ως εξής:

```
(5, None, 1.0, 3, 'http://www.dr-chuck.com/csev-blog')
(3, None, 1.0, 4, 'http://www.dr-chuck.com/dr-chuck/resume/speaking.htm')
(1, None, 1.0, 2, 'http://www.dr-chuck.com/csev-blog/')
(1, None, 1.0, 5, 'http://www.dr-chuck.com/dr-chuck/resume/index.htm')
4 γραμμές.
```

Αυτό δείχνει τον αριθμό των εισερχόμενων συνδέσμων, την παλιά κατάταξη της σελίδας, τη νέα κατάταξη της σελίδας, το αναγνωριστικό της σελίδας και τη διεύθυνση url της σελίδας. Το πρόγραμμα *spdump.py* εμφανίζει μόνο σελίδες που έχουν τουλάχιστον έναν εισερχόμενο σύνδεσμο προς αυτές.

Αφού έχετε μερικές σελίδες στη βάση δεδομένων, μπορείτε να εκτελέσετε το page rank στις σελίδες αυτές, χρησιμοποιώντας το πρόγραμμα *sprank.py*. Απλώς του λέτε πόσες επαναλήψεις κατάταξης σελίδων θα εκτελεστούν.

```
Πόσες επαναλήψεις:2
1 0.546848992536
2 0.226714939664
[(1, 0.559), (2, 0.659), (3, 0.985), (4, 2.135), (5, 0.659)]
```

Μπορείτε να διαγράψετε ξανά τη βάση δεδομένων, για να δείτε ότι η κατάταξη σελίδας ενημερώνεται:

```
(5, 1.0, 0.985, 3, 'http://www.dr-chuck.com/csev-blog')
(3, 1.0, 2.135, 4, 'http://www.dr-chuck.com/dr-chuck/resume/speaking.htm')
(1, 1.0, 0.659, 2, 'http://www.dr-chuck.com/csev-blog/')
(1, 1.0, 0.659, 5, 'http://www.dr-chuck.com/dr-chuck/resume/index.htm')
```

```
4 γραμμές.
```

Μπορείτε να εκτελέσετε το *sprank.py* όσες φορές θέλετε και απλά αυτό θα, βελτιώνει την κατάταξη της σελίδας κάθε φορά που το εκτελείτε. Μπορείτε ακόμη και να εκτελέσετε το *sprank.py* μερικές φορές και, στη συνέχεια, να κάνετε spider μερικές ακόμα σελίδες με το *spider.py* και, στη συνέχεια, να εκτελέσετε το *sprank.py* για να επανασυγκλίνετε οι τιμές κατάταξης σελίδας. Μια μηχανή αναζήτησης συνήθως εκτελεί και τα προγράμματα ανίχνευσης και κατάταξης συνεχώς.

Εάν θέλετε να επανεκκινήσετε τους υπολογισμούς κατάταξης σελίδας χωρίς να επαναφέρετε τις ιστοσελίδες, μπορείτε να χρησιμοποιήσετε το *spreset.py* και, στη συνέχεια, να επανεκκινήσετε το *sprank.py*.

```
Πόσες επαναλήψεις:50
1 0.546848992536
2 0.226714939664
3 0.0659516187242
4 0.0244199333
5 0.0102096489546
6 0.00610244329379
...
42 0.000109076928206
43 9.91987599002e-05
44 9.02151706798e-05
45 8.20451504471e-05
46 7.46150183837e-05
47 6.7857770908e-05
48 6.17124694224e-05
49 5.61236959327e-05
50 5.10410499467e-05
[(512, 0.0296), (1, 12.79), (2, 28.93), (3, 6.808), (4, 13.46)]
```

Για κάθε επανάληψη του αλγορίθμου κατάταξης σελίδας εκτυπώνει τη μέση αλλαγή στην κατάταξη σελίδας ανά σελίδα. Το δίκτυο αρχικά είναι αρκετά ασταθές και έτσι οι μεμονωμένες τιμές κατάταξης σελίδας αλλάζουν δραματικά, μεταξύ των επαναλήψεων. Αλλά μετά από μερικές επαναλήψεις, η κατάταξη σελίδας συγκλίνει. Θα πρέπει να εκτελέσετε το *sprank.py* αρκετές φορές, ώστε οι τιμές της κατάταξης σελίδας να συγκλίνουν.

Εάν θέλετε να απεικονίσετε τις τρέχουσες κορυφαίες σελίδες, ως προς την κατάταξη σελίδων, εκτελέστε το *spjson.py* για να διαβάσει τη βάση δεδομένων και να εξάγει τα

δεδομένα, για τις πιο συνδεδεμένες σελίδες, σε μορφή JSON για προβολή σε ένα πρόγραμμα περιήγησης ιστού.

```
Δημιουργία εξόδου JSON στο spider.js...
Πόσους κόμβους? 30
Ανοίξτε το force.html σε ένα πρόγραμμα περιήγησης για να προβάλετε την
οπτικοποίηση
```

Μπορείτε να δείτε αυτά τα δεδομένα ανοίγοντας το αρχείο *force.html* στο πρόγραμμα περιήγησής σας. Αυτό παρουσιάζει μια αυτόματη διάταξη των κόμβων και των συνδέσμων. Μπορείτε να κάνετε κλικ και να σύρετε οποιονδήποτε κόμβο και μπορείτε επίσης να κάνετε διπλό κλικ σε έναν κόμβο για να εντοπίσετε τη διεύθυνση URL που αντιπροσωπεύεται από τον κόμβο.

Εάν εκτελέσετε ξανά τα άλλα βοηθητικά προγράμματα, τότε εκτελέστε ξανά το *spjson.py* και πατήστε το refresh στο πρόγραμμα περιήγησης για να λάβετε τα νέα δεδομένα από το *spider.json*.

Οπτικοποίηση δεδομένων αλληλογραφίας

Μέχρι αυτό το σημείο του βιβλίου, έχετε εξοικειωθεί αρκετά με τα αρχεία δεδομένων *mbox-short.txt* και *mbox.txt*. Τώρα είναι καιρός να προχωρήσουμε την ανάλυσή μας για τα δεδομένα email στο επόμενο επίπεδο.

Στον πραγματικό κόσμο, μερικές φορές πρέπει να αφαιρέσετε κάποια δεδομένα αλληλογραφίας από διακομιστές. Αυτό μπορεί να πάρει αρκετό χρόνο και τα δεδομένα μπορεί να είναι ασυνεπή, γεμάτα σφάλματα και να χρειάζονται αρκετό καθάρισμα ή προσαρμογή. Σε αυτήν την ενότητα, εργαζόμαστε με μια εφαρμογή που είναι η πιο περίπλοκη μέχρι στιγμής, αφαιρούμε σχεδόν ένα gigabyte δεδομένων και τα οπτικοποιούμε.

Μπορείτε να κατεβάσετε αυτήν την εφαρμογή από:

https://www.gr.py4e.com/code3/gmane.zip

Θα χρησιμοποιήσουμε δεδομένα από μια δωρεάν υπηρεσία αρχειοθέτησης λιστών email, που ονομάζεται http://www.gmane.org. Αυτή η υπηρεσία είναι πολύ δημοφιλής σε έργα ανοιχτού κώδικα, επειδή παρέχει ένα ωραίο αρχείο με δυνατότητα αναζήτησης της δραστηριότητας ηλεκτρονικού ταχυδρομείου τους. Έχουν επίσης μια πολύ φιλελεύθερη πολιτική σχετικά με την πρόσβαση στα δεδομένα τους, μέσω του API τους. Δεν έχουν όρια τιμών, αλλά θα σας παρακαλούσα να μην υπερφορτώνετε την υπηρεσία

τους και να λαμβάνετε μόνο τα δεδομένα που χρειάζεστε. Μπορείτε να διαβάσετε τους όρους και τις προϋποθέσεις του gmane σε αυτή τη σελίδα:

http://www.gmane.org/export.php

Εικόνα 16.3: Ένα Σύννεφο Λέξεων από το Sakai Developer List

Είναι πολύ σημαντικό να χρησιμοποιείτε υπεύθυνα τα δεδομένα του gmane.org, προσθέτοντας καθυστερήσεις στην πρόσβασή σας στις υπηρεσίες του και κατανέμοντας μακροχρόνιες εργασίας σε μεγαλύτερα χρονικά διαστήματα. Μην καταχραστείτε αυτήν τη δωρεάν υπηρεσία και μην την καταστρέψετε, για εμάς τους υπόλοιπους.

Όταν τα δεδομένα email του Sakai κυκλοφόρησαν χρησιμοποιώντας αυτό το λογισμικό, παρήγαγε σχεδόν ένα Gigabyte δεδομένων και χρειάστηκαν μια σειρά εκτελέσεων, σε αρκετές ημέρες. Το αρχείο *README.txt* στο παραπάνω ZIP ενδέχεται να έχει οδηγίες σχετικά με το πώς μπορείτε να κάνετε λήψη ενός προκαθορισμένου αντιγράφου του αρχείου *content.sqlite*, για το μεγαλύτερο μέρος του σώματος ηλεκτρονικού ταχυδρομείου του Sakai, ώστε να μην χρειαστεί να κάνετε λήψη, επί πέντε ημέρες, μόνο για την εκτέλεση των προγραμμάτων. Εάν κάνετε λήψη του προκαταρκτικού

περιεχομένου, θα πρέπει να εκτελέσετε τη διαδικασία spidering για να το ενημερώσετε με τα πιο πρόσφατα μηνύματα.

Το πρώτο βήμα είναι να κάνετε ανίχνευση του αποθετηρίου του gmane. Η βασική διεύθυνση URL είναι κωδικοποιημένη στο *gmane.py* και είναι κωδικοποιημένη στη Sakai developer list. Μπορείτε να δημιουργήσετε ένα άλλο αποθετήριο, αλλάζοντας αυτό το βασικό url. Φροντίστε να διαγράψετε το αρχείο *content.sqlite* εάν αλλάξετε τη βασική διεύθυνση url.

Το αρχείο *gmane.py* λειτουργεί ως υπεύθυνος ανιχνευτής ιστού, προσωρινής αποθήκευσης, καθώς εκτελείται αργά και ανακτά ένα μήνυμα αλληλογραφίας ανά δευτερόλεπτο, ώστε να αποφευχθεί η υπερφόρτωση του gmane. Αποθηκεύει όλα τα δεδομένα του σε μια βάση δεδομένων και μπορεί να διακόπτεται και να επανεκκινείται όσο συχνά χρειάζεται. Ενδέχεται να χρειαστούν πολλές ώρες για να ανακτηθούν όλα τα δεδομένα. Επομένως, ίσως χρειαστεί να κάνετε επανεκκίνηση αρκετές φορές.

Ακολουθεί μια εκτέλεση από *gmane.py* που ανακτά τα τελευταία πέντε μηνύματα της λίστας προγραμματιστών Sakai:

```
Πόσα μηνύματα:10
http://download.gmane.org/gmane.comp.cms.sakai.devel/51410/51411 9460
    nealcaidin@sakaifoundation.org 2013-04-05 re: [building ...
http://download.gmane.org/gmane.comp.cms.sakai.devel/51411/51412 3379
    samuelgutierrezjimenez@gmail.com 2013-04-06 re: [building ...
http://download.gmane.org/gmane.comp.cms.sakai.devel/51412/51413 9903
    da1@vt.edu 2013-04-05 [building sakai] melete 2.9 oracle ...
http://download.gmane.org/gmane.comp.cms.sakai.devel/51413/51414 349265
    m.shedid@elraed-it.com 2013-04-07 [building sakai] ...
http://download.gmane.org/gmane.comp.cms.sakai.devel/51414/51415 3481
    samuelgutierrezjimenez@gmail.com 2013-04-07 re: ...
http://download.gmane.org/gmane.comp.cms.sakai.devel/51415/51416 0

Δεν ξεκινά με From
```

Το πρόγραμμα σαρώνει το *content.sqlite* από το id 0 μέχρι τον πρώτο αριθμό μηνύματος που δεν έχει ήδη ελεγχθεί και αρχίζει να σαρώνει το μήνυμα αυτό. Συνεχίζει να εκτελείται, έως ότου συμπληρώσει τον επιθυμητό αριθμό μηνυμάτων ή μέχρι να φτάσει σε μια σελίδα που δεν φαίνεται να αποτελεί ένα σωστά διαμορφωμένο μήνυμα.

Μερικές φορές στο **gmane.org** μπορεί να λείπει κάποιο μήνυμα. Ίσως οι διαχειριστές να μπορούν να διαγράψουν μηνύματα ή ίσως να έχουν χαθεί. Εάν η ανίχνευσή σας

σταματήσει και φαίνεται να συνάντησε κάποιο μήνυμα που λείπει, μεταβείτε στο SQLite Manager και προσθέστε μια εγγραφή με το αναγνωριστικό που λείπει, αφήνοντας όλα τα άλλα πεδία κενά και επανεκκινήστε το *gmane.py*. Αυτό θα ξεκολλήσει τη διαδικασία ανίχνευσης και θα της επιτρέψει να συνεχιστεί. Αυτά τα κενά μηνύματα θα αγνοηθούν στην επόμενη φάση της διαδικασίας.

Το καλό είναι ότι, αφού έχετε ανιχνεύσει όλα τα μηνύματα και τα έχετε στο *content.sqlite*, μπορείτε να εκτελέσετε ξανά το *gmane.py* για να λάβετε τα νέα μηνύματα που αποστέλλονται στη λίστα.

Τα δεδομένα *content.sqlite* είναι αρκετά ακατέργαστα, με ένα αναποτελεσματικό μοντέλο δεδομένων και δεν είναι συμπιεσμένα. Αυτό γίνεται σκόπιμα, καθώς σας επιτρέπει να κοιτάξετε το *content.sqlite*, στο SQLite Manager, για να εντοπίσετε προβλήματα με τη διαδικασία ανίχνευσης. Δεν θα ήταν καλή ιδέα το να εκτελέσετε οποιαδήποτε ερωτήματα σε αυτήν τη βάση δεδομένων, καθώς θα ήταν αρκετά αργό.

Η δεύτερη διαδικασία είναι η εκτέλεση του προγράμματος *gmodel.py*. Αυτό το πρόγραμμα διαβάζει τα ακατέργαστα δεδομένα από το *content.sqlite* και παράγει μια καθαρισμένη και καλά διαμορφωμένη έκδοση των δεδομένων στο αρχείο *index.sqlite*. Αυτό το αρχείο θα είναι πολύ μικρότερο (συχνά έως και 10 φορές μικρότερο) από το *content.sqlite*, επειδή συμπιέζει την κεφαλίδα και το σώμα κειμένου.

Κάθε φορά που εκτελείται το *gmodel.py* αυτό διαγράφει και αναδημιουργεί το *index.sqlite*, επιτρέποντάς σας να προσαρμόσετε τις παραμέτρους του και να επεξεργαστείτε τους πίνακες αντιστοίχισης στο *content.sqlite*, για να τροποποιήσετε τη διαδικασία καθαρισμού δεδομένων. Αυτό είναι ένα δείγμα εκτέλεσης του *gmodel.py*. Εκτυπώνει μια γραμμή, κάθε φορά που υποβάλλονται σε επεξεργασία 250 μηνύματα αλληλογραφίας, ώστε να μπορείτε να δείτε κάποια πρόοδο που συμβαίνει, καθώς αυτό το πρόγραμμα μπορεί να εκτελείται για αρκετό χρονικό διάστημα, μιας και επεξεργάζεται σχεδόν ένα Gigabyte δεδομένων αλληλογραφίας.

```
Loaded allsenders 1588 and mapping 28 dns mapping 1
1 2005-12-08T23:34:30-06:00 ggolden22@mac.com
251 2005-12-22T10:03:20-08:00 tpamsler@ucdavis.edu
501 2006-01-12T11:17:34-05:00 lance@indiana.edu
751 2006-01-24T11:13:28-08:00 vrajgopalan@ucmerced.edu
...
```

Το πρόγραμμα *gmodel.py* χειρίζεται μια σειρά από εργασίες καθαρισμού δεδομένων.

Τα ονόματα τομέα διασπώνται σε δύο τμήματα αναζητώντας τα .com, .org, .edu και .net.

Άλλα ονόματα τομέα διασπώνται σε τρία επίπεδα. Έτσι το si.umich.edu γίνεται umich.edu και το caret.cam.ac.uk γίνεται cam.ac.uk. Οι διευθύνσεις email επίσης εξαναγκάζονται σε πεζά γράμματα, και ορισμένες από τις διευθύνσεις @gmane.org, όπως η παρακάτω

```
arwhyte-63aXycvo3TyHXe+LvDLADg@public.gmane.org
```

μετατρέπονται στην πραγματική διεύθυνση, όποτε υπάρχει μια αντίστοιχη πραγματική διεύθυνση ηλεκτρονικού ταχυδρομείου σε άλλο σημείο του σώματος του μηνύματος.

Στη βάση δεδομένων *mapping.sqlite* υπάρχουν δύο πίνακες που σας επιτρέπουν να αντιστοιχίσετε τόσο ονόματα τομέα όσο και μεμονωμένες διευθύνσεις email, που αλλάζουν κατά τη διάρκεια ζωής της λίστας email. Για παράδειγμα, ο Steve Githens χρησιμοποίησε τις ακόλουθες διευθύνσεις ηλεκτρονικού ταχυδρομείου, καθώς άλλαζε θέσεις εργασίας κατά τη διάρκεια ζωής της λίστας προγραμματιστών Sakai:

```
s-githens@northwestern.edu
sgithens@cam.ac.uk
swgithen@mtu.edu
```

Μπορούμε να προσθέσουμε δύο εγγραφές στον πίνακα Mapping στο *mapping.sqlite*, έτσι το *gmodel.py* θα αντιστοιχίσει και τις τρεις σε μία διεύθυνση:

```
s-githens@northwestern.edu ->  swgithen@mtu.edu
sgithens@cam.ac.uk -> swgithen@mtu.edu
```

Μπορείτε επίσης να καταχωρήσετε παρόμοιες εγγραφές στον πίνακα DNSMapping, εάν υπάρχουν πολλά ονόματα DNS που θέλετε να αντιστοιχιστούν σε ένα μόνο DNS. Η ακόλουθη αντιστοίχιση προστέθηκε στα δεδομένα Sakai:

```
iupui.edu -> indiana.edu
```

Έτσι, όλοι οι λογαριασμοί από τις διάφορες πανεπιστημιουπόλεις των Πανεπιστημίων της Ιντιάνα παρακολουθούνται μαζί.

Μπορείτε να εκτελέσετε ξανά το *gmodel.py*, ξανά και ξανά, καθώς εξετάζετε τα δεδομένα και να προσθέτετε αντιστοιχίσεις για να κάνετε τα δεδομένα καθαρότερα και καθαρότερα. Όταν τελειώσετε, θα έχετε μια ωραία ευρετηριασμένη έκδοση του ηλεκτρονικού ταχυδρομείου, στο *index.sqlite*. Αυτό είναι το αρχείο που χρησιμοποιείται για την ανάλυση δεδομένων. Με αυτό το αρχείο, η ανάλυση δεδομένων θα είναι πολύ γρήγορη.

Η πρώτη, απλούστερη ανάλυση δεδομένων είναι να προσδιοριστεί "ποιος έστειλε τα περισσότερα μηνύματα;" και "ποιος οργανισμός έστειλε τα περισσότερα mail"; Αυτό

γίνεται χρησιμοποιώντας το *gbasic.py*:

```
Πόσες διαγραφές; 5
Φορτωμένα μηνύματα= 51330 θέματα= 25033 αποστολείς= 1584

Κορυφαίοι 5 συμμετέχοντες στη λίστα Email
steve.swinsburg@gmail.com 2657
azeckoski@unicon.net 1742
ieb@tfd.co.uk 1591
csev@umich.edu 1304
david.horwitz@uct.ac.za 1184

Κορυφαίοι 5 οργανισμοί στη λίστα Email
gmail.com 7339
umich.edu 6243
uct.ac.za 2451
indiana.edu 2258
unicon.net 2055
```

Σημειώστε πόσο πιο γρήγορα εκτελείται το *gbasic.py* σε σύγκριση με το *gmane.py* ή ακόμα και το *gmodel.py*. Όλα λειτουργούν στα ίδια δεδομένα, αλλά το *gbasic.py* χρησιμοποιεί τα συμπιεσμένα και κανονικοποιημένα δεδομένα του *index.sqlite*. Εάν έχετε πολλά δεδομένα για διαχείριση, μια διαδικασία πολλαπλών βημάτων, όπως αυτή σε αυτήν την εφαρμογή, μπορεί να χρειαστεί λίγο περισσότερο χρόνο για να αναπτυχθεί, αλλά θα σας εξοικονομήσει πολύ χρόνο όταν αρχίσετε πραγματικά να εξερευνάτε και να οπτικοποιείτε τα δεδομένα σας.

Μπορείτε να δημιουργήσετε μια απλή απεικόνιση της συχνότητας λέξης, στις γραμμές θέματος του αρχείου *gword.py*:

```
Εύρος μετρήσεων: 33229 129
Έξοδος γραμμένη στο gword.js
```

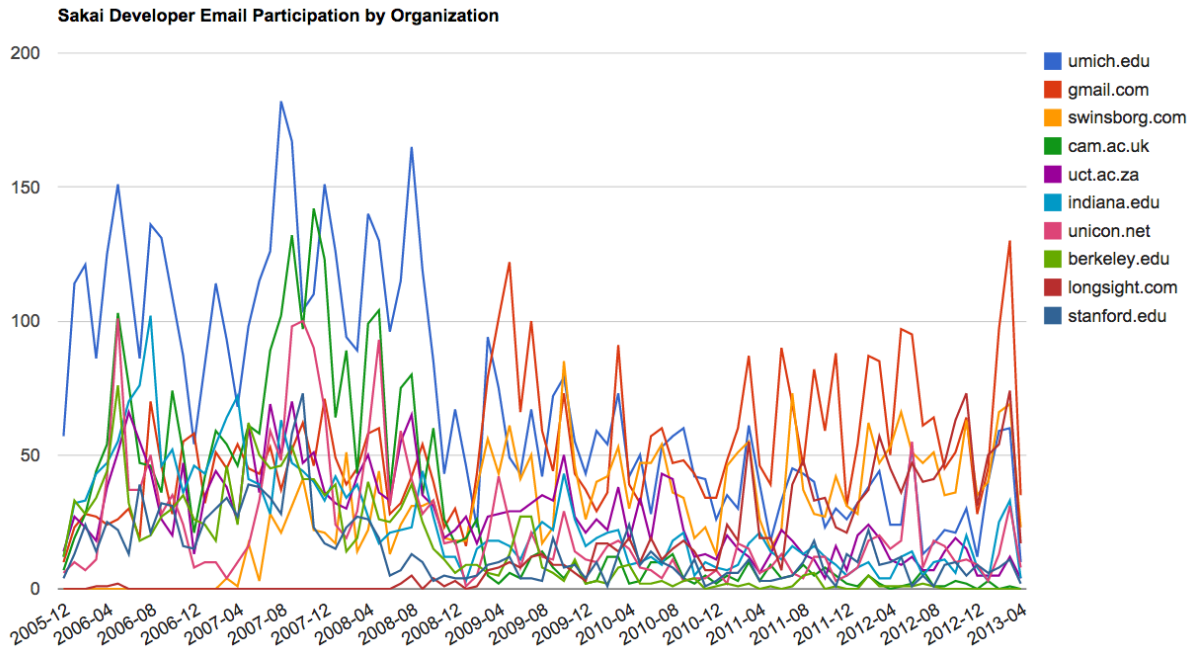
Αυτό παράγει το αρχείο *gword.js* το οποίο μπορείτε να οπτικοποιήσετε, χρησιμοποιώντας *gword.htm*, για να δημιουργήσετε ένα σύννεφο λέξεων παρόμοιο με αυτό στην αρχή αυτής της ενότητας.

Μια δεύτερη οπτικοποίηση παράγεται από το *gline.py*. Υπολογίζει τη συμμετοχή μέσω email από οργανισμούς με την πάροδο του χρόνου.

```
Φορτωμένα μηνύματα= 51330 αποστολείς= 1584
Κορυφαίοι 10 Οργανισμοί
```

```
['gmail.com', 'umich.edu', 'uct.ac.za', 'indiana.edu',
'unicon.net', 'tfd.co.uk', 'berkeley.edu', 'longsight.com',
'stanford.edu', 'ox.ac.uk']
Output written to gline.js
```

Η έξοδός του γράφεται στο *gline.js* το οποίο οπτικοποιείται χρησιμοποιώντας *gline.htm*.



Εικόνα 16.4: Δραστηριότητα Αλληλογραφίας Sakai ανά Οργανισμό

Αυτή είναι μια σχετικά περίπλοκη και εξελιγμένη εφαρμογή που διαθέτει δυνατότητες για την ανάκτηση, τον καθαρισμό και την οπτικοποίηση πραγματικών δεδομένων.

Παράρτημα A

Άτομα που Συνεισέφεραν - Contributors

A.1 Λίστα των ατόμων που συνεισέφεραν για το Python για Όλους

Κωνσταντία Κιουρτίδου

Μπορείτε να δείτε λεπτομέρειες των ατόμων που συνεισέφεραν στη διεύθυνση: https://github.com/csev-gr/py4e/graphs/contributors

A.2 Λίστα των ατόμων που συνεισέφεραν για το Python for Everybody

Andrzej Wójtowicz, Elliott Hauser, Stephen Catto, Sue Blumenberg, Tamara Brunnock, Mihaela Mack, Chris Kolosiwsky, Dustin Farley, Jens Leerssen, Naveen KT, Mirza Ibrahimovic, Naveen (@togarnk), Zhou Fangyi, Alistair Walsh, Erica Brody, Jih-Sheng Huang, Louis Luangkesorn, and Michael Fudge

Μπορείτε να δείτε λεπτομέρειες των ατόμων που συνεισέφεραν στη διεύθυνση: https://github.com/csev/py4e/graphs/contributors

A.3 Λίστα των ατόμων που συνεισέφεραν για το Python for Informatics

Bruce Shields for copy editing early drafts, Sarah Hegge, Steven Cherry, Sarah Kathleen Barbarow, Andrea Parker, Radaphat Chongthammakun, Megan Hixon, Kirby Urner, Sarah Kathleen Barbrow, Katie Kujala, Noah Botimer, Emily Alinder, Mark Thompson-Kular, James Perry, Eric Hofer, Eytan Adar, Peter Robinson, Deborah J. Nelson, Jonathan C. Anthony, Eden Rassette, Jeannette Schroeder, Justin Feezell, Chuanqi Li, Gerald Gordinier, Gavin Thomas Strassel, Ryan Clement, Alissa Talley, Caitlin Holman, Yong-Mi Kim, Karen Stover, Cherie Edmonds, Maria Seiferle, Romer Kristi D. Aranas (RK), Grant Boyer, Hedemarrie Dussan,

A.4 Πρόλογος για το "Think Python"

A.4.1 Η περίεργη ιστορία του "Think Python"

(Allen B. Downey)
Τον Ιανουάριο του 1999 ετοιμαζόμουν να διδάξω ένα εισαγωγικό μάθημα προγραμματισμού στην Java. Το είχα ήδη διδάξει τρεις φορές και είχα απογοητευτεί. Το ποσοστό αποτυχίας στην τάξη ήταν πολύ υψηλό αλλά, ακόμη και για τους φοιτητές που πέτυχαν, το συνολικό επίπεδο επίδοσης ήταν πολύ χαμηλό.

Ένα από τα προβλήματα που εντόπισα ήταν τα βιβλία. Ήταν πολύ μεγάλα, με πάρα πολλές, περιττές λεπτομέρειες σχετικά με την Java και όχι αρκετή καθοδήγηση υψηλού επιπέδου σχετικά με τον τρόπο προγραμματισμού. Και όλοι υπέφεραν από το φαινόμενο της παγίδας: ξεκινούσαν εύκολα, προχωρούσαν σταδιακά και μετά κάπου γύρω στο Κεφάλαιο 5 έχανες τη γη κάτω από τα πόδια σου. Οι φοιτητές θα δεχόταν πολύ νέο υλικό, πολύ γρήγορα, και έπρεπε να περάσουν το υπόλοιπο του εξαμήνου μαζεύοντας τα κομμάτια.

Δύο εβδομάδες πριν από την πρώτη μέρα των μαθημάτων, αποφάσισα να γράψω το δικό μου βιβλίο. Οι στόχοι μου ήταν:

- Κράτα το σύντομο. Είναι καλύτερο για τους μαθητές να διαβάσουν 10 σελίδες παρά να μην διαβάσουν 50 σελίδες.
- Να είσαι προσεκτικός με το λεξιλόγιο. Προσπάθησα να ελαχιστοποιήσω την ορολογία και να ορίσω κάθε έννοια στην πρώτη χρήση της.
- Προχώρα βήμα βήμα. Για να αποφύγω τις παγίδες, πήρα τα πιο δύσκολα θέματα και τα χώρισα σε μια σειρά από μικρά βήματα.
- Εστίασε στον προγραμματισμό, όχι στη γλώσσα προγραμματισμού. Συμπεριέλαβα το ελάχιστο χρήσιμο υποσύνολο της Java και άφησα έξω τα υπόλοιπα.

Χρειαζόμουν έναν τίτλο, οπότε από μια ιδιοτροπία επέλεξα το *How to Think Like a Computer Scientist*.

Η πρώτη μου έκδοση ήταν άκομψη, αλλά λειτούργησε. Οι μαθητές διάβασαν και κατάλαβαν αρκετά ώστε μπορούσα να αφιερώσω χρόνο στην τάξη για τα δύσκολα θέματα, τα ενδιαφέροντα θέματα και (το πιο σημαντικό) μπορούσα να αφήσω τους φοιτητές να εξασκηθούν.

Κυκλοφόρησα το βιβλίο με την άδεια GNU Free Documentation License, η οποία επιτρέπει στους χρήστες να αντιγράφουν, να τροποποιούν και να διανέμουν το βιβλίο.

Το καλύτερο είναι αυτό που συνέβη στη συνέχεια. Ο Jeff Elkner, καθηγητής γυμνασίου στη Βιρτζίνια, υιοθέτησε το βιβλίο μου και το μετέγραψε σε Python. Μου έστειλε ένα αντίγραφό του και είχα την ασυνήθιστη εμπειρία να μάθω Python διαβάζοντας το δικό μου βιβλίο.

Ο Jeff και εγώ αναθεωρήσαμε το βιβλίο, ενσωματώσαμε μια μελέτη περίπτωσης από τον Chris Meyers και το 2001 κυκλοφορήσαμε το *How to Think Like a Computer Scientist Learning with Python*, επίσης υπό την άδεια GNU Free Documentation. Ως Green Tea Press, δημοσίευσα το βιβλίο και άρχισα να πουλάω έντυπα αντίτυπα μέσω του Amazon.com και των κολεγιακών βιβλιοπωλείων. Άλλα βιβλία από το Green Tea Press είναι διαθέσιμα στο greenteapress.com.

Το 2003 άρχισα να διδάσκω στο Olin College και διδάσκω για πρώτη φορά Python. Η σύγκριση με την Java ήταν εντυπωσιακή. Οι μαθητές δυσκολεύτηκαν λιγότερο, έμαθαν περισσότερα, δούλεψαν σε πιο ενδιαφέρουσες εργασίες και γενικά διασκέδασαν πολύ περισσότερο.

Τα επόμενα πέντε χρόνια συνέχισα να αναπτύσσω το βιβλίο, διορθώνοντας λάθη, βελτιώνοντας ορισμένα από τα παραδείγματα και προσθέτοντας υλικό, ειδικά ασκήσεις. Το 2008 άρχισα να εργάζομαι σε μια σημαντική αναθεώρηση — την ίδια στιγμή, επικοινώνησε μαζί μου ένας συντάκτης του Cambridge University Press που ενδιαφερόταν να δημοσιεύσει την επόμενη έκδοση. Καλός συγχρονισμός!

Ελπίζω να σας απολαύσετε τη δουλειά με αυτό το βιβλίο και να σας βοηθήσει να μάθετε να προγραμματίζετε και να σκέφτεστε, τουλάχιστον λίγο, σαν επιστήμονας υπολογιστών.

A.4.2 Ευχαριστίες για το "Think Python"

(Allen B. Downey)
Πρώτον και πιο σημαντικό, ευχαριστώ τον Jeff Elkner, ο οποίος μετέγραψε το Java βιβλίο μου σε Python, με το οποίο ξεκίνησε αυτό το έργο και με σύστησε σε αυτήν που αποδείχθηκε η αγαπημένη μου γλώσσα.

Ευχαριστώ επίσης τον Chris Meyers, ο οποίος συνέβαλε σε πολλές ενότητες του *How to Think Like a Computer Scientist*.

Και ευχαριστώ το Free Software Foundation για την ανάπτυξη της GNU Free Documentation License, η οποία βοήθησε να καταστεί δυνατή η συνεργασία μου με τον Jeff και τον Chris.

Ευχαριστώ επίσης τους συντάκτες του Lulu που εργάστηκαν στο *How to Think Like a Computer Scientist*.

Ευχαριστώ όλους τους φοιτητές που εργάστηκαν με προηγούμενες εκδόσεις αυτού του βιβλίου και όλους τους συντελεστές (που αναφέρονται σε ένα Παράρτημα), που έστειλαν διορθώσεις και προτάσεις.

Και ευχαριστώ τη σύζυγό μου, Lisa, για τη δουλειά της σε αυτό το βιβλίο, και το Green Tea Press, καθώς και οτιδήποτε άλλο.

Allen B. Downey
Needham MA

Ο Allen Downey είναι Associate Professor της Επιστήμης Υπολογιστών στο Franklin W. Olin College of Engineering.

A.5 Λίστα των ατόμων που συνεισέφεραν για το "Think Python"

(Allen B. Downey)

Περισσότεροι από 100 οξυδερκείς και προσεκτικοί αναγνώστες έχουν στείλει προτάσεις και διορθώσεις τα τελευταία χρόνια. Η συνεισφορά τους και ο ενθουσιασμός τους για αυτό το έργο ήταν τεράστια βοήθεια.

Για λεπτομέρειες σχετικά με τη φύση καθεμιάς από τις συνεισφορές από αυτά τα άτομα, δείτε το κείμενο "Think Python".

Lloyd Hugh Allen, Yvon Boulianne, Fred Bremmer, Jonah Cohen, Michael Conlon, Benoit Girard, Courtney Gleason and Katherine Smith, Lee Harr, James Kaylin, David Kershaw, Eddie Lam, Man-Yong Lee, David Mayo, Chris McAloon, Matthew J. Moelter, Simon Dicon Montford, John Ouzts, Kevin Parks, David Pool, Michael Schmitt, Robin Shaw, Paul Sleigh, Craig T. Snydal, Ian Thomas, Keith Verheyden, Peter Winstanley, Chris Wrobel, Moshe Zadka, Christoph Zwerschke, James Mayer, Hayden McAfee, Angel Arnal, Tauhidul Hoque and Lex Berezhny, Dr. Michele Alzetta, Andy Mitchell, Kalin Harvey, Christopher P. Smith, David Hutchins, Gregor Lingl, Julie Peters, Florin Oprina, D. J. Webre, Ken, Ivo Wever, Curtis Yanko, Ben Logan, Jason Armstrong, Louis Cordier, Brian Cain, Rob Black, Jean-Philippe Rey at Ecole Centrale Paris, Jason Mader at George Washington University made a number Jan Gundtofte-Bruun, Abel David and Alexis Dinno, Charles Thayer, Roger Sperberg, Sam Bull, Andrew Cheung, C. Corey Capel, Alessandra, Wim Champagne, Douglas Wright, Jared Spindor, Lin Peiheng, Ray Hagtvedt, Torsten Hübsch, Inga Petuhhov, Arne Babenhauserheide, Mark E. Casida, Scott Tyler, Gordon Shephard, Andrew Turner, Adam Hobart, Daryl Hammond and Sarah Zimmerman, George Sass, Brian Bingham, Leah Engelbert-Fenton, Joe Funke, Chao-chao Chen, Jeff Paine, Lubos Pintes, Gregg Lind and Abigail Heithoff, Max Hailperin, Chotipat Pornavalai, Stanislaw Antol, Eric Pashman, Miguel Azevedo, Jianhua Liu, Nick King, Martin Zuther, Adam Zimmerman, Ratnakar Tiwari, Anurag Goel, Kelli Kratzer, Mark Griffiths, Roydan Ongie, Patryk Wolowiec, Mark Chonofsky, Russell Coleman, Wei Huang, Karen Barber, Nam Nguyen, Stéphane Morin, Fernando Tardio, and Paul Stoop.

Παράρτημα Β

Λεπτομέρειες πνευματικών δικαιωμάτων

Θα προτιμούσα να αδειοδοτήσω το βιβλίο με τη λιγότερο περιοριστική άδεια CC-BY-SA. Όμως, δυστυχώς, υπάρχουν μερικοί αδίστακτοι οργανισμοί, που αναζητούν και βρίσκουν βιβλία με ελεύθερη άδεια και στη συνέχεια δημοσιεύουν και πωλούν σχεδόν αυτούσια αντίγραφα των βιβλίων σε κάποια υπηρεσία εκτύπωσης κατ' απαίτηση, όπως η LuLu ή η KDP. Το KDP έχει προσθέσει (ευτυχώς) μια πολιτική που κατοχυρώνει στις επιθυμίες του πραγματικού κατόχου πνευματικών δικαιωμάτων έναντι ενός μη κατόχου πνευματικών δικαιωμάτων, που προσπαθεί να δημοσιεύσει ένα έργο με ελεύθερη άδεια. Δυστυχώς, υπάρχουν πολλές υπηρεσίες εκτύπωσης κατ' απαίτηση και πολύ λίγες έχουν σκεφτεί τόσο καλά την πολιτική τους όσο το KDP.

Δυστυχώς, πρόσθεσα το στοιχείο NC στην άδεια χρήσης αυτού του βιβλίου για να έχω διέξοδο, σε περίπτωση που κάποιος προσπαθήσει να αντιγράψει αυτό το βιβλίο και να το πουλήσει εμπορικά. Δυστυχώς, η προσθήκη NC περιορίζει τις χρήσεις αυτού του υλικού που θα ήθελα να επιτρέψω. Έτσι, έχω προσθέσει αυτήν την ενότητα του εγγράφου για να περιγράψω συγκεκριμένες καταστάσεις όπου δίνω εκ των προτέρων την άδειά μου να χρησιμοποιηθεί το υλικό αυτού του βιβλίου σε καταστάσεις που κάποιοι μπορεί να θεωρήσουν εμπορικές.

- Εάν εκτυπώνετε περιορισμένο αριθμό αντιγράφων ολόκληρου ή μέρους αυτού του βιβλίου για χρήση σε ένα μάθημα (π.χ., όπως ένα πακέτο μαθημάτων), τότε σας εκχωρείται άδεια CC-BY για αυτό το υλικό για αυτόν τον σκοπό.
- Εάν είστε καθηγητής σε πανεπιστήμιο και μεταφράζετε αυτό το βιβλίο σε άλλη γλώσσα, εκτός από τα αγγλικά και διδάσκετε χρησιμοποιώντας το μεταφρασμένο βιβλίο, τότε μπορείτε να επικοινωνήσετε μαζί μου και θα σας χορηγήσω άδεια CC-BY-SA για αυτό το υλικό, σε σχέση με τη δημοσίευση της μετάφρασής σας. Συγκεκριμένα, θα σας επιτραπεί να πουλήσετε εμπορικά το μεταφρασμένο βιβλίο που προκύπτει.

Εάν σκοπεύετε να μεταφράσετε το βιβλίο, μπορεί να επικοινωνήσετε μαζί μου για να βεβαιωθούμε ότι έχετε όλο το σχετικό υλικό μαθημάτων ώστε να μπορέσετε να το μεταφράσετε και αυτό.

Φυσικά, μπορείτε να επικοινωνήσετε μαζί μου και να ζητήσετε άδεια εάν αυτές οι ρήτρες δεν επαρκούν. Σε όλες τις περιπτώσεις, η άδεια επαναχρησιμοποίησης και αναμίξεως αυτού του υλικού θα χορηγείται εφόσον υπάρχει σαφής προστιθέμενη αξία ή όφελος για μαθητές ή καθηγητές που θα προκύψουν ως αποτέλεσμα της νέας εργασίας.

Charles Severance
www.dr-chuck.com
Ann Arbor, MI, ΗΠΑ
9 Σεπτεμβρίου 2013

Ευρετήριο