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

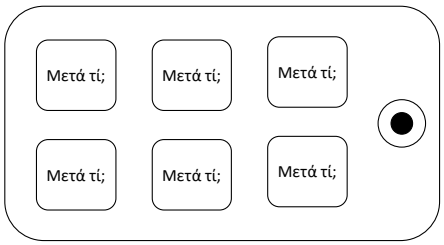
Ο Charles Severance είναι Clinical Associate Professor στο University of Michigan School of Information.

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 κειμένονται 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 πρόγραμματίζετε 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

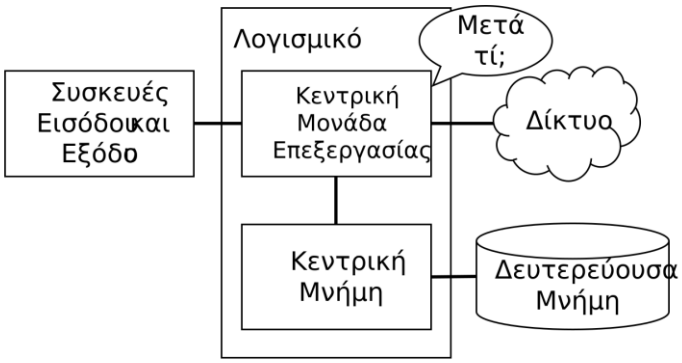
εξαιρετική εμπειρία χρήσης. Σε ορισμένες περιπτώσεις, όταν επιλέγετε ένα κομμάτι λογισμικού, οι προγραμματιστές αποζημιώνονται άμεσα λόγω της επιλογής σας.

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:

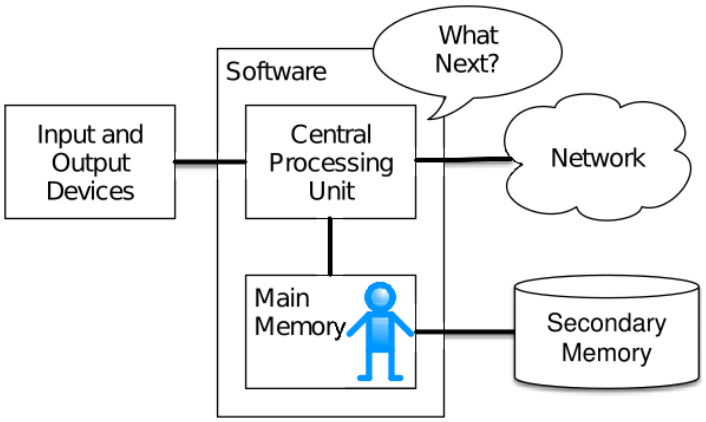
Οι ορισμοί υψηλού επιπέδου αυτών των τμημάτων είναι οι εξής:

- 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 : When a program displays a message and pauses for the user to type some input to the program.

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 *S*ubtraction/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
>>> print(name)
```

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

```
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
```

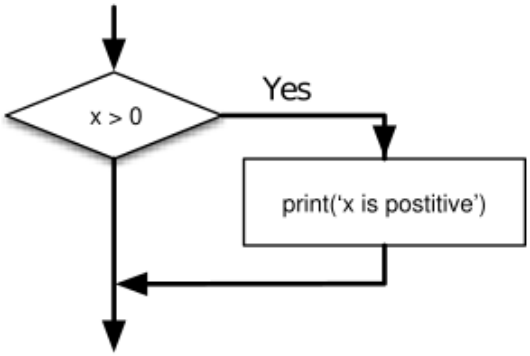
Αυτή η ευελιξία μπορεί να είναι χρήσιμη, αλλά υπάρχουν κάποιες λεπτομέρειες σε αυτό, που μπορεί να προκαλέσουν σύγχυση. Ίσως θα ήταν καλύτερα να το αποφύγετε μέχρι να είστε σίγουροι ότι ξέρετε τι κάνετε.

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
```

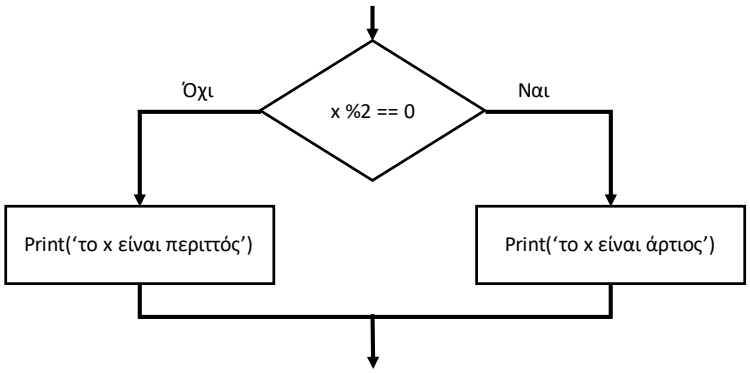
Η κενή γραμμή στο τέλος ενός μπλοκ δηλώσεων δεν είναι απαραίτητη όταν γράφετε και εκτελείτε σενάριο, μπορεί όμως να βελτιώσει την αναγνωσιμότητα του κώδικα σας.

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

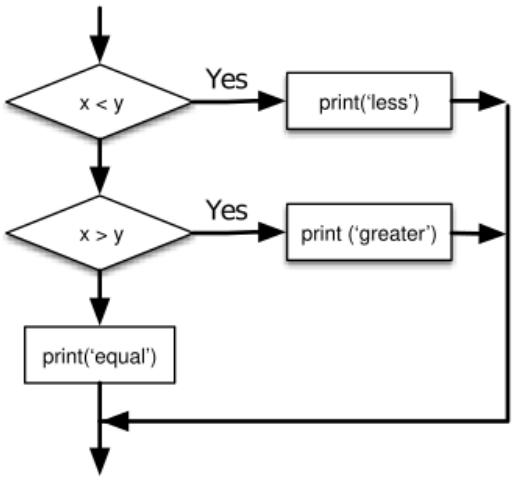
Δεδομένου ότι η συνθήκη πρέπει να είναι αληθής ή ψευδής, θα εκτελεστεί ακριβώς μία από τις εναλλακτικές περιπτώσεις. Οι εναλλακτικές περιπτώσεις ονομάζονται *κλάδοι*, επειδή αποτελούν διακλαδώσεις στη ροή εκτέλεσης.

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')
```

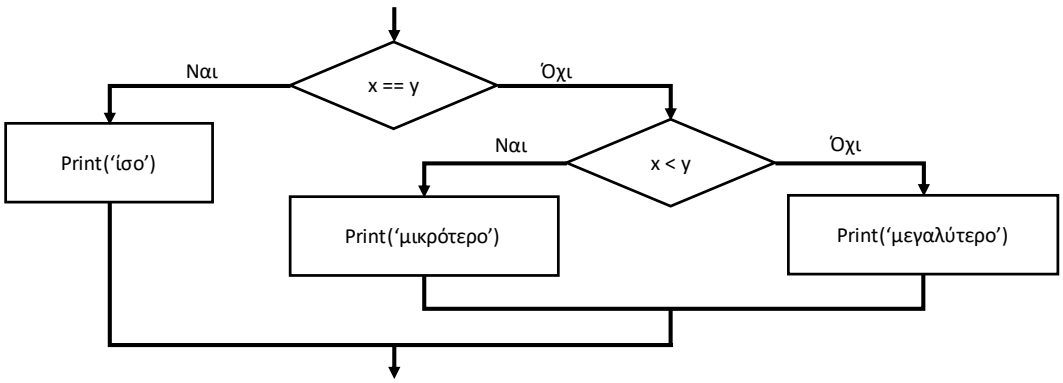
Κάθε συνθήκη ελέγχεται με τη σειρά. Εάν η πρώτη είναι ψευδής, ελέγχεται η επόμενη και ούτω καθεξής. Εάν μία από αυτές είναι αληθής, ο αντίστοιχος κλάδος εκτελείται και η εντολή τελειώνει. Ακόμα κι αν περισσότερες από μία συνθήκες είναι αληθείς, εκτελείται μόνο ο πρώτος αληθής κλάδος.

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^[Θα εξετάσουμε

λεπτομερέστερα τις λίστες σε επόμενο κεφάλαιο.] 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

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).

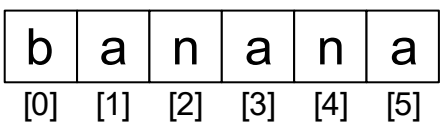
Αλλά μπορεί να μην πάρετε αυτό που περιμένετε:

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

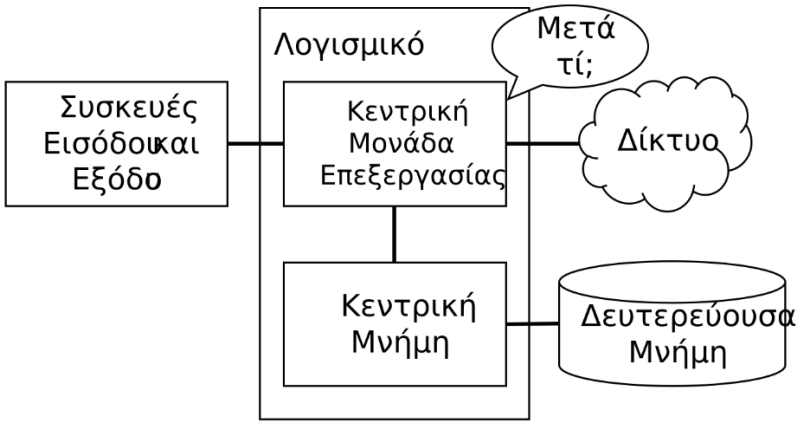
Κεφάλαιο 7

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.

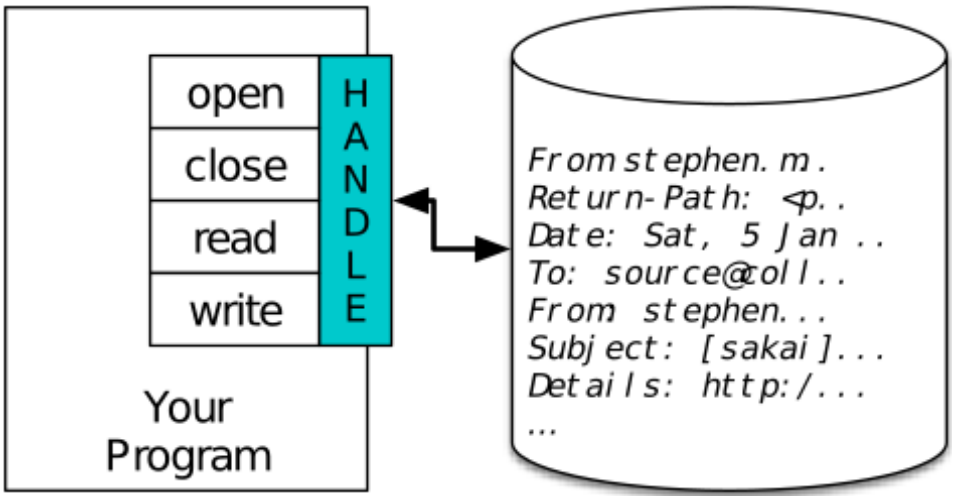
Θα επικεντρωθούμε κυρίως στην ανάγνωση και τη σύνταξη αρχείων κειμένου όπως αυτά που δημιουργούμε σε ένα πρόγραμμα επεξεργασίας κειμένου. Αργότερα θα δούμε

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 σωστά:

   ```
   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
```

Παράρτημα A

Άτομα που Συνεισέφεραν - Contributors

A.1 Λίστα συνεισφερόντων 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 συνεισφερόντων στη διεύθυνση:

https://github.com/csev/py4e/graphs/contributors

A.2 Λίστα συνεισφερόντων 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.3 Πρόλογος για το "Think Python"

A.3.1 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.3.2 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.

Α.4 Λίστα συνεισφερόντων 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