**ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ**

**Τμήμα Μηχανικών Πληροφορικής & Τηλεπικοινωνιών**

# Mini εγχειρίδιο Python

Ταμπάκη Ειρήνη- Μαρία

**Φεβρουάριος 2014**

**Επιβλέπων: Μηνάς Δασυγένης**

**http://arch.icte.uowm.gr**

Το εγχειρίδιο αυτό δημιουργήθηκε στα πλαίσια του Project Nand2Tetris για το μάθημα της Αρχιτεκτονικής Ηλεκτρονικών Υπολογιστών

Τα σχόλια σε Python ξεκινούν με το χαρακτήρα # και επεκτείνονται ως το τέλος της γραμμής. Ένα σχόλιο μπορεί να εμφανιστεί στην αρχή μιας γραμμής ή μετά από κώδικα, αλλά όχι μέσα σε ένα string. Μερικά παραδείγματα:

```
# this is the first comment
spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

Στα ακόλουθα παραδείγματα, εισόδου και εξόδου διακρίνονται από την παρουσία ή την απουσία προτροπών ( >>>and... ): πρέπει να πληκτρολογείτε τα πάντα μετά την προτροπή. Γραμμές που δεν ξεκινούν με μια προτροπή σημαίνουν έξοδο από το διερμηνέα. Σημειώστε ότι μια δευτερεύουσα προτροπή σε μια γραμμή μόνη της, σημαίνει ότι θα πρέπει να πληκτρολογήσετε μια κενή γραμμή. Αυτό χρησιμοποιείται για να τερματίσετε μια εντολή multi-line.

**Χρησιμοποιώντας τη Python ως Υπολογιστή χειρός**

Ας δοκιμάσουμε μερικές απλές εντολές Python. Ξεκινήστε τον διερμηνέα και περιμένετε για την πρωτογενή προτροπή,. >>> Ο διερμηνέας δρα σαν ένα απλό κομπιουτεράκι: μπορείτε να πληκτρολογήσετε μια έκφραση σε αυτόν και θα γράψει την τιμή. Η σύνταξη της έκφρασης είναι απλή: οι φορείς +, -, * . Λειτουργεί ακριβώς όπως και στις περισσότερες άλλες γλώσσες (για παράδειγμα, Pascal ή C). Παρενθέσεις (()) μπορούν να χρησιμοποιηθεί για την ομαδοποίηση των εκφράσεων, για παράδειγμα:

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating point number
1.6
```

Οι ακέραιοι αριθμοί αριθμοί (π.χ. 2, 4, 20) έχουν τύπο int, εκείνοι με ένα κλασματικό μέρος (π.χ. 5.0, 1.6) έχουν τον τύπο float.

Ο τελεστής της διαίρεσης (/) επιστρέφει πάντα έναν float αριθμό. Για να το κάνετε διαίρεση και να πάρετε ένα ακέραιο αποτέλεσμα (απορρίπτοντας κάθε κλασματικό αποτέλεσμα), μπορείτε να χρησιμοποιήσετε το (/ /). Για να υπολογίσετε το υπόλοιπο μπορείτε να χρησιμοποιήσετε το (%):

```
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3  # floor division discards the fractional part
5
>>> 17 % 3  # the % operator returns the remainder of the
division
2
>>> 5 * 3 + 2  # result * divisor + remainder
17
```

Με την Python είναι δυνατόν να χρησιμοποιήσετε τον τελεστή (**) για να υπολογίσετε δυνάμεις:

```
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

Το σύμβολο της ισότητας (=) χρησιμοποιείται για να εκχωρήσετε μια τιμή σε μια μεταβλητή. Στη συνέχεια, δεν εμφανίζεται κανένα αποτέλεσμα πριν από την επόμενη διαδραστική προτροπή:

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Αν μια μεταβλητή δεν είναι "ορίζεται" (αποδίδεται αξία), προσπαθώντας να το χρησιμοποιήσετε θα σας εμφανίσει ένα σφάλμα:

```
>>> n  # try to access an undefined variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Υπάρχει πλήρης υποστήριξη για float κινητής υποδιαστολής. Φορείς με τελεστές μικτού τύπου μετατρέπονται σε float κινητής υποδιαστολής:

```
>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5
```

Όταν χρησιμοποιείτε η Python ως υπολογιστής χειρός, είναι κάπως πιο εύκολο να συνεχίσει κανείς τους υπολογισμούς του με τη χρήση της μεταβλητής _,αφού η τελευταία έντυπη έκφραση αποδίδεται στη μεταβλητή αυτή. Για παράδειγμα:

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

Αυτή η μεταβλητή θα πρέπει να αντιμετωπίζονται ως μόνο για ανάγνωση από τον χρήστη. Μην εκχωρήσετε ρητά μια τιμή σε αυτή ( _ ) θα δημιουργήσει μια ανεξάρτητη τοπική μεταβλητή με το ίδιο όνομα καλύπτοντας την ενσωματωμένη μεταβλητή.

Εκτός από τους τύπους δεδομένων int και να float, η Python υποστηρίζει και άλλους τύπους δεδομένων, όπως τους decimal και fraction. Η Python έχει επίσης ενσωματωμένη υποστήριξη για μιγαδικούς αριθμούς, και χρησιμοποιεί το j ή

J ως κατάληξη για να δείξει το φανταστικό μέρος (π.χ. 3 +5 j).

**Strings**

Εκτός από τους αριθμούς, η Python μπορεί να χειριστεί και αλφαριθμητικά, τα οποία μπορούν να εκφραστούν με διάφορους τρόπους. Μπορούν να περικλείονται σε μονά εισαγωγικά ('...') ή διπλά εισαγωγικά ("..."), με το ίδιο αποτέλεσμα.:

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

Στο διαδραστικό διερμηνέα, το string της εξόδου περικλείεται σε εισαγωγικά και οι ειδικοί χαρακτήρες διέφυγαν με backslashes. Ενώ αυτό μπορεί μερικές φορές να φαίνεται διαφορετικά από την είσοδο, οι δύο χορδές είναι ισοδύναμες. Το string περικλείονται σε μονά ή διπλά εισαγωγικά. Η συνάρτηση print () παράγει μια πιο ευανάγνωστη έξοδο, παραλείποντας τα εισαγωγικά και εκτυπώνοντας ειδικούς χαρακτήρες:

```
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
>>> print('"Isn\'t," she said.')
"Isn't," she said.
>>> s = 'First line.\nSecond line.'  # \n means newline
>>> s  # without print(), \n is included in the output
'First line.\nSecond line.'
>>> print(s)  # with print(), \n produces a new line
First line.
Second line.
```

Αν δεν θέλετε χαρακτήρες που προλογίζεται από \ να ερμηνευθούν ως ειδικοί χαρακτήρες, μπορείτε να χρησιμοποιήσετε string προσθέτοντας ένα r πριν από το πρώτο απόσπασμα:

```
>>> print('C:\some\name')  # here \n means newline!
C:\some
ame
>>> print(r'C:\some\name')  # note the r before the quote
C:\some\name
```

Οι συμβολοσειρές μπορεί να εκτείνονται σε πολλαπλές γραμμές. Ένας τρόπος είναι η χρήση τριπλών εισαγωγικών: """..."""  ή  '''...'''  . Το τέλος των γραμμών συμπεριλαμβάνεται αυτόματα στα strings, αλλά είναι δυνατόν να αποφευχθεί αυτό με την προσθήκη ενός \ στο τέλος της γραμμής. Ακολουθεί παράδειγμα:

```
      print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
```

Παράγετε το ακόλουθο αποτέλεσμα (σημειώστε ότι η αρχική αλλαγή γραμμής δεν συμπεριλαμβάνεται):

```
      Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

Τα strings μπορούν να είναι συνεχόμενα (κολλημένες) με τον τελεστή +, και να επαναλαμβάνονται με *:

```
>>> # 3 times 'un', followed by 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```

Δύο ή περισσότερες συμβολοσειρές η μία δίπλα στην άλλη αυτόματα αποτελούν μία ενιαία συνεχόμενη συμβολοσειρά.

```
      >>> 'Py' 'thon'
'Python'
```

Αυτό λειτουργεί μόνο με string ,αλλά ΟΧΙ με μεταβλητές ή εκφράσεις:

```
>>> prefix = 'Py'
>>> prefix 'thon'  # can't concatenate a variable and a
string literal
  ...
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
  ...
SyntaxError: invalid syntax
```

Αν θέλετε να ενώσετε μεταβλητές ή μια μεταβλητή και ένα string, χρησιμοποιήστε +:

```
>>> prefix + 'thon'
'Python'
```

Τα παραπάνω χαρακτηριστικά είναι ιδιαίτερα χρήσιμα όταν θέλετε να διαιρέσετε σε μικρότερα τμήματα strings μεγάλου μεγέθους:

```
>>> text = ('Put several strings within parentheses '
            'to have them joined together.')
>>> text
'Put several strings within parentheses to have them joined
together.'
```

Τα strings μπορούν να αναπροσαρμόζονται (subscripted), σύμφωνα με τον πρώτο χαρακτήρα που έχει δείκτη 0. Δεν υπάρχει ξεχωριστός τύπος χαρακτήρα. Ένας χαρακτήρας είναι απλά μια συμβολοσειρά μεγέθους ένα(1):

```
>>> word = 'Python'
```

```
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
```

Οι δείκτες μπορεί επίσης να είναι αρνητικοί αριθμοί. Σε αυτή την περίπτωση αρχίζουν να μετρούν από τα δεξά προς τα αριστερά:

```
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'
```

Σημειώστε ότι εφόσον το -0 είναι το ίδιο με το 0, οι αρνητικά δείκτες ξεκινούν από -1.

Επιπλέον από την Python υποστηρίζεται ο τεμαχισμός, ο οποίος επιτρέπει να αποκτήσετε ένα τμήμα της συμβολοσειράς:

```
>>> word[0:2]  # characters from position 0 (included) to 2
(excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5
(excluded)
'tho'
```

Σημειώστε πως η αρχή περιλαμβάνεται πάντα, ενώ το τέλος εξαιρείται. Αυτό εξασφαλίζει ότι s [: i] + s [i:] είναι πάντα ίσο με s:

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

Ένα τμήμα δεικτών έχει χρήσιμες προεπιλογές. Είναι δυνατόν να παραλείπεται το πρώτο στοιχείο, όταν η τιμή του

δείκτη ισούται με μηδέν ή να παραλείπεται το τέλος αυτού αφού το μέγεθος της συμβολοσειράς είναι σταθερό.

```
>>> word[:2]  # character from the beginning to position 2 (excluded)
'Py'
>>> word[4:]  # characters from position 4 (included) to the end
'on'
>>> word[-2:] # characters from the second-last (included) to the end
'on'
```

Ένας τρόπος για να θυμάστε πώς λειτουργούν οι τομές είναι να σκεφτούμε τους δείκτες που δείχνουν μεταξύ των χαρακτήρων, με το αριστερό άκρο του πρώτου χαρακτήρα αριθμούνται από το 0. Τότε η δεξιά ακμή του τον τελευταίο χαρακτήρα μιας συμβολοσειράς N χαρακτήρων έχει δείκτη n, για παράδειγμα:

```
+---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Η πρώτη σειρά των αριθμών δίνει τη θέση των δεικτών άπό 0 έως 6 της συμβολοσειράς, η δεύτερη γραμμή δίνει τους αντίστοιχους αρνητικούς δείκτες. Τα τμήματα από i έως j αποτελούνται από όλους τους χαρακτήρες μεταξύ των άκρων i και j, αντίστοιχα.

Για τους μη-αρνητικούς δείκτες, το μήκος ενός τμήματος είναι η διαφορά των δεικτών, με την προϋπόθεση ότι και οι δύο είναι εντός των ορίων. Για παράδειγμα, το μήκος της λέξης [1:03] είναι 2.

Αν επιχειρήσετε να χρησιμοποιήσετε ένα δείκτη που είναι πάρα πολύ μεγάλος, θα οδηγηθείτε σε ένα σφάλμα:

```
>>> word[42]  # the word only has 7 characters
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Ωστόσο, η λειτουργία διεκπεραιώνεται ομαλά όταν χρησιμοποιείται για τεμαχισμό:

```
>>> word[4:42]
'on'
>>> word[42:]
''
```

Τα αλφαριθμητικά σε Python δεν μπορούν να αλλάξουν - είναι αμετάβλητα. Επομένως, η απόδοση σε δείκτη της θέση μιας συμβολοσειράς οδηγεί σε λάθη:

```
>>> word[0] = 'J'
  ...
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
  ...
TypeError: 'str' object does not support item assignment
```

Εάν χρειάζεστε μια διαφορετική συμβολοσειρά, θα πρέπει να δημιουργήσετε ένα νέο:

```
>>> 'J' + word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```

Η ενσωματωμένη συνάρτηση len () επιστρέφει το μήκος μιας συμβολοσειράς:

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

**Λίστες**

Η Python ξέρει ένα αριθμό τύπων δεδομένων ένωσης, που χρησιμοποιούνται για την ομαδοποίηση των άλλων τιμών. Η πιο ευέλικτη είναι η λίστα, η οποία μπορεί να γραφεί ως μια λίστα τιμών αντικειμένων) διαχωρισμένων με κόμματα μέσα σε αγκύλες[].Οι λίστες μπορεί να περιέχουν αντικείμενα διαφόρων τύπων, αλλά συνήθως τα στοιχεία έχουν όλα τον ίδιο τύπο.

```
>>> squares = [1, 2, 4, 9, 16, 25]
>>> squares
[1, 2, 4, 9, 16, 25]
```

Όπως τα αλφαριθμητικά ,οι λίστες μπορούν να αναπροσαρμόζονται και να διαιρούνται σε τμήματα:

```
>>> squares[0]  # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]
```

Όλες οι πράξεις επιστρέφουν μια νέα λίστα που περιέχει τα ζητούμενα στοιχεία. Αυτό σημαίνει ότι το ακόλουθο κομμάτι επιστρέφει ένα νέο αντίγραφο του καταλόγου:

```
>>> squares[:]
[1, 2, 4, 9, 16, 25]
```

Οι λίστες υποστηρίζουν επίσης λειτουργίες όπως αυτή της συνένωσης:

```
>>> squares + [36, 49, 64, 81, 100]
[1, 2, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Σε αντίθεση με τα strings, που είναι αμετάβλητα. Οι λίστες είναι ένα μεταβλητού τύπου, δηλαδή είναι δυνατόν να αλλάξει το περιεχόμενό τους:

```
>>> cubes = [1, 8, 27, 65, 125]  # something's wrong here
>>> 4 ** 3  # the cube of 4 is 64, not 65!
64
>>> cubes[3] = 64  # replace the wrong value
>>> cubes
[1, 8, 27, 64, 125]
```

Μπορείτε επίσης να προσθέσετε νέα στοιχεία στο τέλος της λίστας, με τη χρήση της append():

```
>>> cubes.append(216)  # add the cube of 6
>>> cubes.append(7 ** 3)  # and the cube of 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

Είναι δυνατόν, να αλλάξετε το μέγεθος της λίστας ή ακόμα και να την καταργήσετε εντελώς:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # clear the list by replacing all the elements with an
empty list
>>> letters[:] = []
>>> letters
[]
```

Η ενσωματωμένη συνάρτηση len () ισχύει και για τις λίστες:

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

Τέλος, είναι δυνατή η δημιουργία εμφωλευμένων λιστών να καταλόγους φωλιά, για παράδειγμα:

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

**Τα πρώτα βήματα προγραμματισμού.**

Φυσικά, μπορούμε να χρησιμοποιήσουμε την Python για πιο απαιτητικές εργασίες. Για παράδειγμα, μπορούμε να γράψουμε μια πρώτη υπο-ακολουθία της σειράς Fibonacci ως εξής:

```
>>> # Fibonacci series:
    #the sum of two elements defines the next
... a, b = 0, 1
>>> while b < 10:
...     print(b)
...     a, b = b, a+b
1
1
2
3
5
8
```

Αυτό το παράδειγμα εισάγει πολλά νέα χαρακτηριστικά.

- Η πρώτη γραμμή περιέχει μια πολλαπλή ανάθεση: η μεταβλητές a και b ταυτόχρονα παίρνουν τις νέες τιμές 0 και 1. Στην τελευταία γραμμή αυτό χρησιμοποιείται και πάλι, αποδεικνύοντας ότι οι εκφράσεις στην δεξιά πλευρά αξιολογούνται πριν από οποιαδήποτε ανάθεση. Οι πλευρικές εκφράσεις αξιολογούνται από τα αριστερά προς τα δεξιά .
- Ο βρόχος while εκτελείται όσο η συνθήκη ( εδώ : b < 10 ), εξακολουθεί να ισχύει. Στην Python , όπως και στη C , οποιαδήποτε ακέραια μη μηδενική τιμή είναι αληθής, ενώ η μηδενική τιμή είναι ψευδής. Η συνθήκη μπορεί επίσης να είναι μια συμβολοσειρά ή λίστα τιμών. Σε αυτή την περίπτωση οποιαδήποτε ακολουθία με μη μηδενικό μήκος είναι αληθής. Αν μια ακολουθία είναι κενή, τότε είναι ψευδής. Ο έλεγχος των παραπάνω στο παράδειγμα μας πραγματοποιείται με μια απλή σύγκριση. Οι πρότυπες τελεστές σύγκρισης γράφονται το ίδιο όπως και στη C: < ( μικρότερο από) , > (μεγαλύτερο από ) , == (ίσο με ) , < = ( μικρότερο ή ίσο με) , > = (μεγαλύτερο από ή ίσο με ) και ! = ( διάφορο με) .

- Το εσοχή στο περιεχόμενο του βρόχου. Η εσοχή είναι ο τρόπος, με τον οποίο η Python ομαδοποιεί τις καταστάσεις. Στην διαδραστική προτροπή, θα πρέπει να πληκτρολογήσετε μια καρτέλα ή ένα κενώ (α) για κάθε χαραγμένη γραμμή. Στην πράξη θα προετοιμάσετε πιο περίπλοκη εισόδου για την Python με ένα πρόγραμμα επεξεργασίας κειμένου. Όλα τα αξιοπρεπή προγράμματα επεξεργασίας κειμένου έχουν μια αυτόματη στοίχηση. Όταν μια δήλωση εγγράφεται διαδραστικά, θα πρέπει να ακολουθείται από μια κενή γραμμή για να δείξει την ολοκλήρωση (καθώς το πρόγραμμα ανάλυσης δεν μπορεί να μαντέψει πότε έχετε πληκτρολογήσει την τελευταία γραμμή. Σημειώστε ότι κάθε γραμμή εντός μίας απλής ομάδας πρέπει να έχει εσοχή κατά το ίδιο ποσό.
- Η συνάρτηση print () γράφει την τιμή των επιχειρημάτων, που είναι είναι δεδομένη. Διαφέρει από την απλή εμφάνιση της εκφράσεως (όπως κάναμε νωρίτερα στα παραδείγματα υπολογισμού) με τον τρόπο που χειρίζεται πολλαπλά επιχειρήματα, float κινητής υποδιαστολής, ποσότητες, και strings. Τα strings εκτυπώνονται χωρίς εισαγωγικά, με ένα κενό να παρεμβάλλεται μεταξύ των στοιχείων, ώστε να μπορείτε να διαμορφώσετε τα πράγματα ωραία, όπως αυτό:

```
>>> i = 256*256
>>> print('The value of i is', i)
The value of i is 65536
```

Τελικά λέξεις-κλειδιά μπορεί να χρησιμοποιηθούν για να αποφευχθεί η αλλαγή γραμμής μετά την έξοδο, ή να τερματίσετε την έξοδο με ένα διαφορετικό string:

```
>>> a, b = 0, 1
>>> while b < 1000:
...     print(b, end=',')
...     a, b = b, a+b
...
1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```

**Υποσημειώσεις**

[1] Καθώς το ** έχει υψηλότερη προτεραιότητα από -, η πράξη -3 ** 2 θα πρέπει να ερμηνευθεί ως - (3 ** 2) και ως εκ τούτου να οδηγήσει σε -9. Για να αποφευχθεί αυτό και να πάρει 9, μπορείτε να χρησιμοποιήσετε (-3) ** 2.

[2] Σε αντίθεση με άλλες γλώσσες, οι ειδικοί χαρακτήρες όπως \n έχουν την ίδια έννοια με δύο μονά ('...') ή ένα διπλό ("...") εισαγωγικό. Η μόνη διαφορά μεταξύ των δύο είναι ότι μέσα σε μονά εισαγωγικά δεν χρειάζεται να ξεφύγει "(αλλά θα πρέπει να ξεφύγουν \ ') και αντιστρόφως.

## Εργαλεία ελέγχου ροής

Εκτός του while η Python γνωρίζει τις συνήθεις εντολές ελέγχου ροής, που είναι γνωστές από άλλες γλώσσες, με ορισμένες ανατροπές.

Ίσως ο πιο γνωστός τύπος ελέγχου ροής είναι η if. Για παράδειγμα:

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

Μπορεί να μην υπάρχουν ή να υπάρχουν περισσότερα από ένα τμήματα elif, ενώ το τμήμα else είναι προαιρετικό. Η λέξη-κλειδί «elif» προκύπτει από το "else if», και είναι χρήσιμο να αποφευχθεί η υπερβολική εσοχή. Μια if ... elif ... elif ... ακολουθία είναι ένα υποκατάστατο για τις δηλώσεις διακοπής ή περίπτωση που διαπιστώνονται σε άλλες γλώσσες προγραμματισμού.

Η δήλωση for στην Python διαφέρει λίγο από αυτό που μπορεί να χρησιμοποιείτε στις C και Pascal. Η for στην Python επαναλαμβάνει τη δήλωση πάνω από τα στοιχεία της κάθε σειράς (μιας λίστα ή μιας συμβολοσειράς), με τη σειρά που εμφανίζονται στην αλληλουχία. Για παράδειγμα:

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

Αν χρειαστεί να τροποποιήσετε τη σειρά πάνω από την επανάληψη, ενώ στο εσωτερικό του βρόχου, συνιστάται ότι θα πρέπει πρώτα να δημιουργήσετε ένα αντίγραφο. Επανάληψη σε μια ακολουθία δεν δημιουργεί έμμεσα ένα αντίγραφο. Ο τεμαχισμός κάνει αυτή τη λειτουργία ιδιαίτερα βολική:

```
>>> for w in words[:]:  # Loop over a slice copy of the
entire list.
...     if len(w) > 6:
...         words.insert(0, w)
...
>>> words
['defenestrate', 'cat', 'window', 'defenestrate']
```

Εάν χρειάζεται να μετακινηθείτε σε μια ακολουθία αριθμών, η ενσωματωμένη συνάρτηση range() είναι ιδιαίτερα πρακτική, καθώς παράγει μια αριθμητική πρόοδο:

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

Το δοσμένο ως τελικό σημείο δεν είναι ποτέ μέρος της παραγόμενης ακολουθίας. Αν ,λοιπόν, δίνεται η συνάρτηση range(10) παράγει 10 αξίες, των δεικτών για τα στοιχεία μιας ακολουθίας μήκους 10. Είναι, ωστόσο, δυνατόν να ξεκινά το φάσμα από έναν άλλο αριθμό, ή να καθορίζεται από τον χρήστη μια διαφορετική αύξηση (έστω και αρνητικό):

```
range(5, 10)
   5 through 9

range(0, 10, 3)
   0, 3, 6, 9

range(-10, -100, -30)
  -10, -40, -70
```

Στις περισσότερες περιπτώσεις, όμως, είναι βολικό να χρησιμοποιείτε την συνάρτηση enumerate().

Κάτι περίεργο συμβαίνει αν απλά να εκτυπώσετε μια range:

```
>>> print(range(10))
range(0, 10)
```

Από πολλές απόψεις το αντικείμενο που επιστρέφεται από το range() συμπεριφέρεται σαν να είναι λίστα, αλλά στην πραγματικότητα δεν είναι. Είναι ένα αντικείμενο που επιστρέφει τα διαδοχικά στοιχεία της επιθυμητής αλληλουχίας όταν επαναλαμβάνεται, αλλά αυτό δεν δημιουργεί πραγματικά μία λίστα, εξοικονομώντας χώρο.

Λέμε ένα αντικείμενο iterable ,δηλαδή, ότι είναι κατάλληλο ως στόχος για τις συναρτήσεις και τις δομές που περιμένουν κάτι από το οποίο μπορούν να αποκτήσουν διαδοχικά στοιχεία μέχρι να εξαντληθούν τα αποθέματα. Η συνάρτηση list(), δημιουργεί λίστες από iterables:

```
>>> list(range(5))
```

```
[0, 1, 2, 3, 4]
```

Η εντολή break, όπως και στη C, βγαίνει εκτός του εσωτερικότερου βρόχου for ή while, από τον οποίο περικλείεται.

Οι δηλώσεις Loop μπορεί να έχουν έναν όρο else, να εκτελείται όταν ο βρόχος τερματίζει με εξάντληση της λίστας (με for) ή όταν η κατάσταση γίνεται ψευδής (με while), αλλά όχι όταν ο βρόχος τερματίζεται με την εντολή break. Αυτό επεξηγείται από τον επόμενο βρόχο, ο οποίοw ψάχνει για τους πρώτους αριθμούς:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

Ο παραπάνω κώδικας είναι σωστός. Κοιτάξτε προσεκτικά: Η else ανήκει στον βρόχο for και όχι στην if.

Η δήλωση continue, έχει επίσης δανειστεί από την C και συνεχίζει με την επόμενη επανάληψη του βρόχου:

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print("Found an even number", num)
...         continue
```

```
...     print("Found a number", num)
Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9
```

Η εντολή pass δεν κάνει τίποτα. Μπορεί να χρησιμοποιηθεί όταν η δήλωση είναι απαραίτητη συντακτικά, αλλά το πρόγραμμα δεν απαιτεί καμία ενέργεια. Για παράδειγμα:

```
>>> while True:
...     pass  # Busy-wait for keyboard interrupt (Ctrl+C)
...
```

Αυτό η εντολή συνήθως χρησιμοποιείται για τη δημιουργία ελάχιστες κλάσεων:

```
>>> class MyEmptyClass:
...     pass
...
```

Ένα άλλο μέρος που μπορεί να χρησιμοποιηθεί η pass είναι για τη δέσμευση μνήμης μιας λειτουργία ή υπό όρους τμήμα όταν εργάζεστε σε νέο κώδικα. Το πέρασμα αγνοείται σιωπηλά από:

```
>>> def initlog(*args):
...     pass   # Remember to implement this!
...
```

Είναι δυνατόν να δημιουργήσουμε μια συνάρτηση που να γράφει τη σειρά Fibonacci σε ένα αυθαίρετο όριο:

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Η λέξη-κλειδί def εισάγει τον ορισμό μιας συνάρτησης. Θα πρέπει να ακολουθείται από το όνομα της συνάρτησης και εντός παρενθέσεων λίστα με τις τυπικές παραμέτρους. Οι καταστάσεις που αποτελούν το σώμα της συνάρτησης ξεκινούν στην επόμενη γραμμή, και θα πρέπει να έχουν εσοχή.

Η πρώτη δήλωση του σώματος μιας συνάρτησης μπορεί προαιρετικά να είναι ένα string. Αυτό το string αποτελεί την συμβολοσειρά τεκμηρίωσης της συνάρτησης ή docstrins. Υπάρχουν εργαλεία που χρησιμοποιούν τα docstrings να παράγουν αυτόματα online ή έντυπο υλικό τεκμηρίωσης, ή να αφήσει το χρήστη διαδραστικά να περιηγηθεί μέσα σε κώδικα. Είναι καλή πρακτική να συμπεριλάβει docstrings στον κώδικα που έχετε γράψει, έτσι ώστε να γίνει συνήθεια.

Η εκτέλεση μιας συνάρτησης εισάγει ένα νέο πίνακα συμβόλων που χρησιμοποιούνται για τις τοπικές μεταβλητές της συνάρτησης. Ειδικότερα, όλες οι αναθέσεις μεταβλητών σε μία συνάρτηση αποθηκεύουν την αξία σε τοπικούς πίνακες συμβόλων, Οι αναφορές των μεταβλητών αρχικά αποθηκεύονται σε τοπικό πίνακα συμβόλων, στη συνέχεια, σε τοπικούς πίνακες σύμβολων που περικλείουν συναρτήσεις, στη συνέχεια, σε έναν καθολικό πίνακα συμβόλων, και, τέλος, σε πίνακα ενσωματωμένων ονομάτων. Έτσι, στις καθολικές μεταβλητές δεν μπορεί να αποδοθεί άμεσα μια τιμή μέσω μιας συνάρτησης (εκτός και αν είναι χαρακτηρισμένη ως global), αν και μπορεί να υπάρχουν παραπομπές.

Οι πραγματικές παράμετροι με μια κλήση συνάρτησης εισάγονται στο τοπικό πίνακα συμβόλων της καλούμενης

συνάρτησης, όταν αυτή καλείται. Ως εκ τούτου, οι παράμετροι περνούν με τη κλήση της συνάρτησης με βάση την αξία (των οποίων η αξία είναι πάντα μια αναφορά στη συνάρτηση). [1] Όταν μια συνάρτηση καλεί μία άλλη συνάρτηση, ένας νέος τοπικός πίνακας συμβόλων δημιουργείται για την εν λόγω συνάρτηση.

Ένας ορισμός συνάρτησης εισάγει το όνομα της συνάρτησης στον τρέχοντα πίνακα συμβόλων. Η αξία του ονόματος της συνάρτησης έχει ένα τύπο που αναγνωρίζεται από τον διερμηνέα ως συνάρτηση οριζόμενη από το χρήστη. Αυτή η τιμή μπορεί να εκχωρηθεί σε ένα άλλο όνομα το οποίο μπορεί στη συνέχεια να χρησιμοποιηθεί επίσης ως συνάρτηση. Αυτό εξυπηρετεί ως ένας γενικός μηχανισμός μετονομασίας:

```
>>> fib
<function fib at 10042ed0>
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89
>>> fib(0)
>>> print(fib(0))
None
```

Γνωρίζοντας άλλες γλώσσες, μπορεί να αντιτάξετε ότι το fib δεν αποτελεί συνάρτηση, αλλά μια διαδικασία, δεδομένου ότι δεν επιστρέφει μια τιμή. Στην πραγματικότητα, ακόμη και συναρτήσεις χωρίς δήλωση επιστροφής επιστρέφουν μια τιμή, αν και μάλλον βαρετή. Η τιμή αυτή ονομάζεται None. Γράφοντας την τιμή None συνήθως καταστέλλεται από τον διερμηνέα, αν αυτό θα είναι η μόνη τιμή που γράφεται. Μπορείτε να το δείτε, αν θέλετε πραγματικά να χρησιμοποιούν print ():

```
>>> fib(0)
>>> print(fib(0))
None
```

Είναι απλό να γράψετε μια συνάρτηση που να επιστρέφει μια λίστα με τους αριθμούς της σειράς Fibonacci, αντί να τους εκτυπώνει:

```
>>> def fib2(n): # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up
to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100                # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Αυτό το παράδειγμα, ως συνήθως, αποδεικνύει κάποια νέα χαρακτηριστικά Python:

- Η εντολή return επιστρέφει με μια τιμή από μια συνάρτηση. Το return χωρίς να χρησιμοποιεί κάποιο επιχείρημα επιστρέφει None.
- Η δήλωση result.append(a) καλεί μία μέθοδο της λίστας αποτελεσμάτων αντικειμένου. Μια μέθοδος είναι μια συνάρτηση που «ανήκει» σε ένα αντικείμενο και ονομάζεται obj.methodname, όπου obj είναι κάποιο αντικείμενο (αυτό μπορεί να είναι μια έκφραση) και methodname είναι το όνομα μιας μεθόδου που καθορίζεται από τον τύπο του αντικειμένου. Διαφορετικοί τύποι καθορίζουν διαφορετικές μεθόδους. Οι μέθοδοι των διαφόρων τύπων μπορεί να έχουν το ίδιο όνομα χωρίς να δημιουργούν ασάφεια. Η μέθοδος append(), όπως φαίνεται στο παράδειγμα ορίζεται για τη λίστα αντικειμένων, προσθέτοντας ένα νέο στοιχείο στο τέλος της λίστας. Στο παράδειγμα αυτό είναι ισοδύναμο με result= result + [α], αλλά πιο αποτελεσματική.

**Καθορισμός Συναρτήσεων**

Είναι δυνατόν να καθοριστεί μια συνάρτηση με μεταβλητό αριθμό ορισμάτων. Υπάρχουν τρεις μορφές, οι οποίες μπορούν να συνδυαστούν.

Η πιο χρήσιμη μορφή είναι να καθορίσετε μια προεπιλεγμένη τιμή για ένα ή περισσότερα ορίσματα. Αυτό δημιουργεί μια συνάρτηση που μπορεί να κληθεί με λιγότερα ορίσματα από αυτά που επιτρέπονται. Για παράδειγμα:

```
def ask_ok(prompt, retries=4, complaint='Yes or no,
please!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise IOError('refusenik user')
        print(complaint)
```

Αυτή η λειτουργία μπορεί να καλείται με διάφορους τρόπους:

- δίνοντας μόνο τα υποχρεωτικά ορίσματα: ask_ok («Θέλετε πραγματικά να σταματήσουν το κάπνισμα;")
- δίνοντας ένα από τα προαιρετικά ορίσματα: ask_ok (« OK για να αντικαταστήσετε το αρχείο; », 2)
- ή ακόμα και δίνοντας όλα τα ορίσματα ask_ok (« OK για να αντικαταστήσετε το αρχείο »,2 «! Έλα, μόνο ναι ή όχι »)

Αυτό το παράδειγμα εισάγει επίσης τη λέξη-κλειδί in. Αυτό ελέγχει κατά πόσον ή όχι μια ακολουθία περιέχει μια ορισμένη τιμή.

Οι προεπιλεγμένες τιμές αξιολογήθηκαν στο σημείο του ορισμού της συνάρτησης στο προσδιορισμό του πεδίου εφαρμογής, έτσι ώστε να εκτυπώνει 5.

```
i = 5

def f(arg=i):
    print(arg)

i = 6
f()
```

Σημαντική προειδοποίηση: Η προεπιλεγμένη τιμή υπολογίζεται μόνο μια φορά. Αυτό κάνει τη διαφορά, όταν η προεπιλογή είναι ένα ευμετάβλητο αντικείμενο, όπως μια λίστα, λεξικό, ή περιπτώσεις περισσότερων κλάσεων. Για παράδειγμα, η ακόλουθη συνάρτηση συσσωρεύει τα ορίσματα που περνά σ 'αυτήν και τις επόμενες κλήσεις:

```
def f(a, L=[]):
    L.append(a)
    return L

print(f(1))
print(f(2))
print(f(3))
```

Αυτή θα εκτυπώνει:

```
[1]
[1, 2]
[1, 2, 3]
```

Εάν δεν θέλετε η προεπιλογή να μοιράζεται μεταξύ διαδοχικών κλήσεων, μπορείτε να γράψετε τη συνάρτηση σαν αυτή:

```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

Οι συναρτήσεις μπορούν επίσης να κληθούν με τη χρήση της keyword arguments της μορφής kwarg = value. Για παράδειγμα, η ακόλουθη συνάρτηση:

```
def parrot(voltage, state='a stiff', action='voom',
type='Norwegian Blue'):
    print("-- This parrot wouldn't", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")
```

δέχεται ένα απαιτούμενο όρισμα (voltage) και τρία προαιρετικά επιχειρήματα (state, action, type). Αυτή η συνάρτηση μπορεί να καλείται με οποιοδήποτε από τους ακόλουθους τρόπους:

```
parrot(1000)                                          # 1
positional argument
parrot(voltage=1000)                                  # 1
keyword argument
parrot(voltage=1000000, action='VOOOOOM')             # 2
keyword arguments
parrot(action='VOOOOOM', voltage=1000000)             # 2
keyword arguments
parrot('a million', 'bereft of life', 'jump')         # 3
positional arguments
parrot('a thousand', state='pushing up the daisies')  # 1
positional, 1 keyword
```

αλλά όλες οι ακόλουθες κλήσεις θα είναι άκυρες:

```
parrot()                     # required argument missing
parrot(voltage=5.0, 'dead')  # non-keyword argument after a
keyword argument
parrot(110, voltage=220)     # duplicate value for the same
argument
parrot(actor='John Cleese')  # unknown keyword argument
```

Σε μια κλήση συνάρτησης, η keyword arguments πρέπει να ακολουθεί τα ορίσματα θέσης. Όλα τα ορίσματα με λέξεις κλειδιά που περνούν πρέπει να ταιριάζουν με ένα από τα ορίσματα που έγιναν δεκτά στην συνάρτηση λειτουργία (π.χ. ο actor δεν αποτελεί βάσιμο επιχείρημα για τη συνάρτηση parrot), ενώ η σειρά τους δεν είναι σημαντική. Αυτό περιλαμβάνει επίσης μη-προαιρετικά ορίσματα (π.χ. parrot(voltage = 1000)). Κανένα όρισμα δεν μπορεί να λάβει μια τιμή πάνω από μία φορά. Εδώ είναι ένα παράδειγμα που αποτυγχάνει εξαιτίας αυτού του περιορισμού:

```
>>> def function(a):
...     pass
...
>>> function(0, a=0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: function() got multiple values for keyword
argument 'a'
```

Όταν μια τελική τυπική παράμετρος της μορφής **name είναι παρούσα, λαμβάνει ένα λεξικό που περιέχει όλα τα ορίσματα, εκτός από εκείνα που αντιστοιχούν στην τυπική παράμετρο. Αυτό μπορεί να συνδυαστεί με μια τυπική παράμετρο της μορφής *name, η οποία δέχεται μια πλειάδα που περιέχει τα ορίσματα θέσης πέρα από την επίσημη λίστα παραμέτρων. (*name πρέπει να συμβεί πριν το **name). Για παράδειγμα, αν ορίσουμε μια συνάρτηση όπως αυτή:

```
def cheeseshop(kind, *arguments, **keywords):
    print("-- Do you have any", kind, "?")
```

```
    print("-- I'm sorry, we're all out of", kind)
    for arg in arguments:
        print(arg)
    print("-" * 40)
    keys = sorted(keywords.keys())
    for kw in keys:
        print(kw, ":", keywords[kw])
```

Θα μπορούσε να κληθεί σαν αυτή:

```
cheeseshop("Limburger", "It's very runny, sir.",
           "It's really very, VERY runny, sir.",
           shopkeeper="Michael Palin",
           client="John Cleese",
           sketch="Cheese Shop Sketch")
```

και φυσικά θα εκτυπώσετε:

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
----------------------------------------
client : John Cleese
shopkeeper : Michael Palin
sketch : Cheese Shop Sketch
```

Σημειώστε ότι η λίστα των λέξεων-κλειδιών των ορισμάτων, έχει δημιουργηθεί από το αποτέλεσμα της διαλογής των κλειδιών του λεξικού λέξεις-κλειδιά του (μέθοδος) πριν από την εκτύπωση των περιεχομένων της. Εάν αυτό δεν γίνει, η σειρά με την οποία τα επιχειρήματα εκτυπώνονται είναι απροσδιόριστη.

Τέλος, η λιγότερο συχνή επιλογή που χρησιμοποιείται είναι να καθορίσετε ότι μια συνάρτηση μπορεί να κληθεί με έναν αυθαίρετο αριθμό επιχειρημάτων. Αυτά τα επιχειρήματα θα είναι κλεισμένα σε μια πλειάδα. Πριν από το μεταβλητό αριθμό ορισμάτων, μπορεί να συμβεί μηδενικό ή κάποιο φυσιολογικό όρισμα.

```
def write_multiple_items(file, separator, *args):
    file.write(separator.join(args))
```

Κανονικά, αυτά τα variadic ορίσματα θα είναι τα τελευταία στη λίστα από τυπικών παραμέτρων, επειδή μαζεύονται όλα τα υπόλοιπα επιχειρήματα των εισροών που διαβιβάστηκαν στη συνάρτηση. Τυχόν τυπικές παράμετροι που επέρχονται μετά την παράμετρο *args είναι μόνο λέξεις-κλειδιά, πράγμα που σημαίνει ότι μπορεί να χρησιμοποιηθούν μόνο ως τέτοια και όχι ως ορίσματα θέσης.

```
>>> def concat(*args, sep="/"):
...     return sep.join(args)
...
>>> concat("earth", "mars", "venus")
'earth/mars/venus'
>>> concat("earth", "mars", "venus", sep=".")
'earth.mars.venus'
```

Η αντίστροφη κατάσταση εμφανίζεται όταν τα ορίσματα είναι ήδη σε μια λίστα ή μια πλειάδα, αλλά πρέπει να αποσυμπιεστούν για την κλήση μιας συνάρτησης που απαιτεί διαφορετικά επιχειρήματα θέσης. Για παράδειγμα, η ενσωματωμένη λειτουργία του range() αναμένει ξεχωριστά ορίσματα έναρξης και διακοπής. Εάν δεν είναι διαθέσιμα ξεχωριστά, γράψτε την κλήση της συνάρτησης με το *-operator με σκοπό να ανοίξει τα ορίσματα από μια λίστα ή μια πλειάδα:

```
>>> list(range(3, 6))            # normal call with separate
arguments
[3, 4, 5]
>>> args = [3, 6]
>>> list(range(*args))            # call with arguments
unpacked from a list
[3, 4, 5]
```

Κατά τον ίδιο τρόπο, τα λεξικά μπορούν να προσφέρουν keyword arguments με την **-operator:

```
>>> def parrot(voltage, state='a stiff', action='voom'):
...     print("-- This parrot wouldn't", action, end=' ')
...     print("if you put", voltage, "volts through it.",
end=' ')
...     print("E's", state, "!")
...
>>> d = {"voltage": "four million", "state": "bleedin'
demised", "action": "VOOM"}
>>> parrot(**d)
-- This parrot wouldn't VOOM if you put four million volts
through it. E's bleedin' demised !
```

Μικρές ανώνυμες συναρτήσεις μπορούν να δημιουργηθούν με τη λέξη-κλειδιά lambda. Η συνάρτηση αυτή επιστρέφει το άθροισμα από δύο ορίσματα : lambda a, b: a + b. Οι συναρτήσεις lambda μπορεί να χρησιμοποιηθούν οπουδήποτε και αν βρίσκονται τα αντικείμενα των συναρτήσεων που απαιτούνται. Συντακτικά περιορίζεται σε μία μόνο έκφραση. Σημασιολογικά, είναι μόνο συντακτική για ένα κανονικό ορισμό της συνάρτησης. Όπως ένθετη ορισμούς συναρτήσεων, συναρτήσεις λάμδα μπορεί να παραπέμπει μεταβλητές από το πεδίο που περιέχει:

```
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

Το παραπάνω παράδειγμα χρησιμοποιεί μια έκφραση λάμδα για να επιστρέψει μια συνάρτηση. Μια άλλη χρήση είναι να περάσει ένα μικρή συνάρτηση ως παράμετρο:

```
>>> pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4,
'four')]
>>> pairs.sort(key=lambda pair: pair[1])
>>> pairs
```

```
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
```

Παρακάτω αναφέρονται μερικές συμβάσεις σχετικά με το περιεχόμενο και τη μορφοποίηση των αλφαριθμητικών τεκμηρίωσης.

Η πρώτη γραμμή πρέπει πάντα να είναι μια σύντομη, συνοπτική παρουσίαση του σκοπού του αντικειμένου. Για λόγους συντομίας, δεν θα πρέπει να αναφέρεται ρητά το όνομα ή ο τύπος του αντικειμένου, δεδομένου ότι αυτά είναι διαθέσιμα με άλλα μέσα ( εκτός αν το όνομα που τυχαίνει να είναι ένα ρήμα που περιγράφει τη λειτουργία μιας συνάρτησης). Η γραμμή αυτή θα πρέπει να αρχίζει με ένα κεφαλαίο γράμμα και να τελειώνει με μια περίοδο .

Αν υπάρχουν περισσότερες γραμμές στη συμβολοσειρά τεκμηρίωσης, η δεύτερη γραμμή θα πρέπει να είναι κενή, διαχωρίζοντας οπτικά την περίληψη από το υπόλοιπο της περιγραφής. Οι ακόλουθες κατευθυντήριες γραμμές θα πρέπει να είναι μία ή περισσότερες παράγραφοι που περιγράφουν τις συμβάσεις για το κάλεσμα του αντικειμένου, τις παρενέργειές της, κλπ.

Η πρώτη μη κενή γραμμή μετά την πρώτη γραμμή του αλφαριθμητικού καθορίζει το ποσό της εσοχής για το σύνολο της συμβολοσειράς τεκμηρίωσης. Το κενό που "ισοδύναμεί" με αυτή την εγκοπή στη συνέχεια απογυμνώνεται από την έναρξη του συνόλου των γραμμών του string. Οι γραμμές που βρίσκονται σε μια μεγαλύτερη εσοχή δεν πρέπει να συμβούν , αλλά αν συμβούν όλα οδηγούν κενά τους θα πρέπει να αφαιρεθούν.

Εδώ είναι ένα παράδειγμα ενός multi-line docstring:

```
>>> def my_function():
...     """Do nothing, but document it.
...
...     No, really, it doesn't do anything.
...     """
...     pass
...
>>> print(my_function.__doc__)
```

```
Do nothing, but document it.

    No, really, it doesn't do anything.
```

**Δομές Δεδομένων**

Ο τύπος δεδομένων λίστας έχει κάποιες μεθόδους παραπάνω. Εδώ είναι όλες οι μέθοδοι λίστας αντικειμένων:

list.**append**(x)
Προσθήκη ενός στοιχείου στο τέλος της λίστας. Ισοδυναμεί με: a[len(a):]=[x].

list.**extend**(L)
Επεκτείνετε τη λίστα με την παράθεση όλων των στοιχείων στην συγκεκριμένη λίστα. Ισοδυναμεί με a[len(a):]=L.

list.**insert**(i,x)
Τοποθετήστε ένα στοιχείο σε μια δεδομένη θέση. Η πρώτη παράμερος είναι ο δείκτης του στοιχείου ενώπιον του οποίου θα εισαχθεί, έτσι το a.insert(0,x) εισάγει στο μπροστινό μέρος της λίστας, και το a.insert(len(a),x) ισοδυναμεί με την a.append(x) .

list.**remove**(x)
Αφαιρέστε το πρώτο στοιχείο από τη λίστα του οποίου η αξία είναι x. Αν δεν υπάρχει τέτοιο στοιχείο, τότε προκαλεί σφάλμα.

list.**pop**([i])
Αφαιρεί το στοιχείο στην δεδομένη θέση στη λίστα, και να το επιστρέφει. Εάν δεν έχει καθοριστεί δείκτης, η a.pop() αφαιρεί και επιστρέφει το τελευταίο στοιχείο στη λίστα. (Οι αγκύλες γύρω από το i, δηλώνουν ότι η παράμετρος αυτή είναι προαιρετική και όχι ότι θα πρέπει να πληκτρολογήσετε αγκύλες σε αυτή τη θέση. Θα δείτε συχνά αυτό το συμβολισμό στην Python. )

list.**clear**()
Αφαιρεί όλα τα στοιχεία από τη λίστα. Ισοδυναμεί με: del a[:].

list.**index**(x)
Επιστρέφει το δείκτη στη λίστα του πρώτου στοιχείου του οποίου η τιμή είναι x. Αν δεν υπάρχει τέτοιο στοιχείο προκαλείται σφάλμα.

list.**count**(x)
Επιστρέφεται ο αριθμός των φορών που το x εμφανίζεται στη λίστα.

list.**sort**()
Ταξινομεί τα αντικείμενα της λίστας σε σειρά.

list.**reverse**()
Αντιστρέφει τα στοιχεία της λίστας σε σειρά.

list.**copy**()
Επιστρέφει ένα αντίγραφο της λίστας . Ισοδυναμεί με a[:].

Ένα παράδειγμα που χρησιμοποιεί τις περισσότερες από τις μεθόδους του παραπάνω καταλόγου:

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print(a.count(333), a.count(66.25), a.count('x'))
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

Μπορεί να έχετε παρατηρήσει ότι σε μεθόδους όπως η εισαγωγή, η αφαίρεση ή η ταξινόμηση μιας λίστας δεν έχει

καμία αξία προς έντυπη επιστροφής -επιστρέφουν None. Αυτή είναι μια αρχή σχεδιασμού για όλες τις μεταβλητές δομές δεδομένων στην Python.

Οι μέθοδοι λίστας, καθιστούν πολύ εύκολη τη χρήση μιας λίστα σαν μία στοίβα, στην οποία το τελευταίο στοιχείο που προστίθεται είναι το πρώτο στοιχείο που ανακτώνται ("last-in, first-out"). Για να προσθέσετε ένα στοιχείο στην κορυφή της στοίβας, χρησιμοποιήστε την append(). Για να ανακτήσετε ένα αντικείμενο από την κορυφή της στοίβας, χρησιμοποιείστε την pop() χωρίς σαφή δείκτη. Για παράδειγμα:

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

Είναι επίσης δυνατόν να χρησιμοποιηθεί μια λίστα, όπως μια ουρά, όπου το πρώτο στοιχείο που προστίθεται είναι το πρώτο στοιχείο, το οποίο ανακτάται ("first-in, first-out"). Ωστόσο, οι λίστες δεν είναι ιδιαίτερα αποτελεσματικές για αυτό το σκοπό. Ενώ οι επισυνάψεις και τα pop από το τέλος της λίστας είναι γρήγορα, οι λειτουργίες αυτές στην αρχή ενός καταλόγου είναι αργότερες (επειδή όλα τα άλλα στοιχεία θα πρέπει να μετατοπιστούν από ένα).

Για την υλοποίηση μιας ουράς, χρησιμοποιήστε την collections.deque που σχεδιάστηκε για να έχουν γρήγορη επισυνάψεις και pop και στα δύο άκρα. Για παράδειγμα:

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arrive
now leaves
'Eric'
>>> queue.popleft()                 # The second to arrive
now leaves
'John'
>>> queue                           # Remaining queue in
order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

Η κατανόηση λίστας παρέχει ένα συνοπτικό τρόπο για να δημιουργίας λιστών. Οι κοινές εφαρμογές δημιουργίας νέων λιστών, όπου κάθε στοιχείο είναι το αποτέλεσμα κάποιων πράξεων που εφαρμόζονται σε κάθε μέλος μιας άλλης ακολουθίας ή δημιουργείτε μία ακολουθία στοιχείων που ικανοποιούν μια συγκεκριμένη συνθήκη.

Για παράδειγμα, ας υποθέσουμε ότι θέλουμε να δημιουργήσουμε μια λίστα με τετράγωνα, όπως:

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Μπορούμε να πετύχουμε το ίδιο αποτέλεσμα με:

```
squares = [x**2 for x in range(10)]
```

Αυτό είναι επίσης ισοδύναμο με squares=list(map(lambda x: x**2, range(10))), αλλά είναι πιο συνοπτικό και ευανάγνωστο.

Η κατανόηση μιας λίστας αποτελείται από αγκύλες περιέχουν μια έκφραση που ακολουθείται από ένα διατάξεις if και for. Το αποτέλεσμα θα είναι μια νέα λίστα που προκύπτει από τον υπολογισμό της έκφρασης στο πλαίσιο των διατάξεων for και if. Για παράδειγμα, αυτή η listcomp συνδυάζει τα στοιχεία των δύο καταλόγων και αν δεν είναι ίσα:

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Και ισοδυναμεί με:

```
>>> combs = []
>>> for x in [1,2,3]:
...     for y in [3,1,4]:
...         if x != y:
...             combs.append((x, y))
...
>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

```
>>> vec = [-4, -2, 0, 2, 4]
>>> # create a new list with the values doubled
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]
>>> # filter the list to exclude negative numbers
>>> [x for x in vec if x >= 0]
[0, 2, 4]
>>> # apply a function to all the elements
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]
>>> # call a method on each element
>>> freshfruit = ['  banana', '  loganberry ', 'passion
fruit  ']
>>> [weapon.strip() for weapon in freshfruit]
```

```
['banana', 'loganberry', 'passion fruit']
>>> # create a list of 2-tuples like (number, square)
>>> [(x, x**2) for x in range(6)]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
>>> # the tuple must be parenthesized, otherwise an error is
raised
>>> [x, x**2 for x in range(6)]
  File "<stdin>", line 1, in ?
    [x, x**2 for x in range(6)]
               ^
SyntaxError: invalid syntax
>>> # flatten a list using a listcomp with two 'for'
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Σημειώστε πως η σειρά των δηλώσεων for και if είναι η ίδια και στα δύο αυτά αποσπάσματα.

```
>>> from math import pi
>>> [str(round(pi, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

Η λίστα μπορεί να περιέχει σύνθετες εκφράσεις και ένθετες συναρτήσεις:

```
>>> from math import pi
>>> [str(round(pi, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

Η αρχική έκφραση σε μια λίστα μπορεί να είναι μια οποιοδήποτε έκφραση, συμπεριλαμβανομένου και μιας άλλης λίστας.

Εξετάστε το ακόλουθο παράδειγμα ενός πίνακα 3x4 που υλοποιείται ως μια λίστα, τριών λιστών μήκους τέσσερα:

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
```

```
... ]
```

Η παρακάτω λίστα μεταφέρει τις γραμμές και σε στήλες:

```
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Όπως είδαμε παραπάνω, η ένθετη listcomp αξιολογείται στο πλαίσιο του γι 'αυτό που ακολουθεί, έτσι ώστε αυτό το παράδειγμα να είναι ισοδύναμο με:

```
>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Που ισοδυναμεί με:

```
>>> transposed = []
>>> for i in range(4):
...     # the following 3 lines implement the nested
listcomp
...     transposed_row = []
...     for row in matrix:
...         transposed_row.append(row[i])
...     transposed.append(transposed_row)
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Στην πραγματικότητα, θα πρέπει να προτιμάτε ενσωματωμένες λειτουργίες σε πολύπλοκες καταστάσεις ροής. Η συνάρτηση zip () κάνει μια σπουδαία εργασία για αυτήν την περίπτωση χρήσης:

```
>>> list(zip(*matrix))
[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```

**Η εντολή del**

Υπάρχει ένας τρόπος για να αφαιρέσετε ένα στοιχείο από μια λίστα δεδομένων μέσω του δείκτη αντί της αξίας του: η εντολή del. Αυτό διαφέρει από την μέθοδο pop(), η οποία επιστρέφει μια τιμή. Η δήλωση del μπορεί επίσης να χρησιμοποιηθεί για να αφαιρέσετει τις τιμές από μια λίστα ή να διαγράψετε ολόκληρη τη λίστα. Για παράδειγμα:

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
```

del μπορεί επίσης να χρησιμοποιηθεί για τη διαγραφή μιας ολόκληρης μεταβλητής:

```
>>> del a
```

Είδαμε ότι οι λίστες και συμβολοσειρές έχουν πολλές κοινές ιδιότητες, όπως την ευρετηρίαση και τον τεμαχισμό πράξεων. Πρόκειται για δύο παραδείγματα τύπων δεδομένων αλληλουχίας. Η Python είναι μια εξελισσόμενη γλώσσα, στην οποία μπορούν να προστεθούν και άλλα είδη δεδομένων ακολουθίας. Υπάρχει επίσης ένα άλλο πρότυπο ακολουθιών τύπου δεδομένων: tuple.

Μια πλειάδα αποτελείται από μια σειρά από τιμές διαχωρισμένες με κόμματα, για παράδειγμα:.

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
```

```
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

Όπως μπορείτε να δείτε, οι πλειάδες εξόδου είναι πάντα μέσα σε παρενθέσεις, έτσι ώστε ένθετα tuples να ερμηνεύονται σωστά. Μπορούν να εισάγονται με ή χωρίς τις παρενθέσεις, αν και συχνά οι παρενθέσεις είναι αναγκαίες ούτως ή άλλως (αν η πλειάδα είναι μέρος μιας μεγαλύτερης έκφρασης). Δεν είναι δυνατόν να αντιστοιχίσετε τα επιμέρους στοιχεία μιας πλειάδας, ωστόσο, είναι δυνατόν να δημιουργηθούν πλειάδες που περιέχουν μεταβλητά αντικείμενα , όπως πίνακες.

Αν και τα tuples μπορεί να φαίνονται παρόμοια με τις λίστες ,συχνά χρησιμοποιούνται σε διάφορες καταστάσεις και για διαφορετικούς σκοπούς. Οι πλειάδες είναι αμετάβλητες, και συνήθως περιέχουν μια ετερογενή σειρά στοιχείων που είναι προσβάσιμες μέσω αποσυμπίεσης ή τιμαριθμικής αναπροσαρμογής. Οι λίστες είναι ευμετάβλητες, και τα στοιχεία τους είναι συνήθως ομοιογενή και είναι προσβάσιμα με την επανάληψη πάνω από τη λίστα.

Ένα ιδιαίτερο πρόβλημα είναι η κατασκευή των πλειάδων που περιέχουν 0 ή 1 στοιχεία: η σύνταξη έχει κάποιες επιπλέον ιδιορρυθμίες για να δεχθεί αυτές. Το άδειασμα ενός tuple υλοποιείτε με τη χρήση ενός άδειου ζευγαριού παρενθέσεων. Μια πλειάδα με ένα στοιχείο κατασκευάζεται ακολουθώντας μια τιμή με ένα κόμμα (δεν αρκεί να επισυνάψουν μια μοναδική τιμή σε παρένθεση). Άσχημο , αλλά αποτελεσματικό. Για παράδειγμα :

```
>>> empty = ()
>>> singleton = 'hello',    # <-- note trailing comma
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)
```

Η δήλωση t=12345,54321, 'hello! ' είναι ένα παράδειγμα tuple: οι τιμές 12345, 54321 και 'hello!' συσκευάζονται μαζί σε μια πλειάδα. Η αντίστροφη λειτουργία είναι επίσης δυνατή:

```
>>> x, y, z = t
```

Αυτό ονομάζεται, αρκετά κατάλληλα, σειρά αποσυμπίεσης και δουλεύει για οποιαδήποτε ακολουθία από την δεξιά πλευρά. Η ακολουθία αποσυμπίεσης απαιτεί την ύπαρξη πολλών μεταβλητών στην αριστερή πλευρά του συμβόλου του ίσον, καθώς υπάρχουν στοιχεία στην ακολουθία. Σημειώστε ότι οι πολλαπλής ανάθεσης είναι πραγματικά ακριβώς ένας συνδυασμός της συμπιεσμένης ακολουθίας.

Η Python περιλαμβάνει επίσης έναν τύπο δεδομένων για τα σύνολα. Ένα σύνολο είναι μια μη διατεταγμένη συλλογή χωρίς διπλότυπα στοιχεία. Βασικές χρήσεις περιλαμβάνουν δοκιμές των μελών και την εξάλειψη των διπλών καταχωρίσεων. Σύνολα αντικειμένων υποστηρίζουν επίσης μαθηματικές πράξεις, όπως ένωση, τομή, διαφορά, και συμμετρική διαφορά.

Άγκιστρα ή η συνάρτηση set(), μπορεί να χρησιμοποιηθεί για να δημιουργήσει σύνολα. Σημείωση: Για να δημιουργήσετε ένα άδειο σύνολο θα πρέπει να χρησιμοποιήσετε set(), not{}, το τελευταίο δημιουργεί ένα άδειο λεξικό.

Εδώ είναι μια σύντομη επίδειξη:

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange',
'banana'}
>>> print(basket)                      # show that
duplicates have been removed
```

```
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # fast membership
testing
True
>>> 'crabgrass' in basket
False

>>> # Demonstrate set operations on unique letters from two
words
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # letters in a but
not in b
{'r', 'd', 'b'}
>>> a | b                              # letters in either a
or b
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # letters in both a
and b
{'a', 'c'}
>>> a ^ b                              # letters in a or b
but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Επίσης:

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
{'r', 'd'}
```

Ένας άλλος χρήσιμος τύπος δεδομένων ενσωματωμένος στη Python είναι το λεξικό. Τα λεξικά βρίσκονται μερικές φορές και σε άλλες γλώσσες ως "συνειρμικές αναμνήσεις" ή " συνειρμικές συστοιχίες". Σε αντίθεση με τις ακολουθίες, οι οποίες αναπροσαρμόζονται από μια σειρά αριθμών, τα λεξικά αναπροσαρμόζονται από τα κλειδιά, τα οποία μπορεί να είναι οποιοσδήποτε αμετάβλητος τύπος. Τα strings και οι αριθμοί μπορεί να είναι πάντα τα κλειδιά. Τα tuples μπορούν να

χρησιμοποιηθούν ως κλειδιά εάν περιέχουν μόνο συμβολοσειρές , αριθμούς, ή άλλα tuples. Εάν μια πλειάδα περιέχει οποιοδήποτε ευμετάβλητο αντικείμενο άμεσα ή έμμεσα, δεν μπορεί να χρησιμοποιηθεί ως κλειδί. Δεν μπορείτε να χρησιμοποιήσετε τις λίστες, όπως τα κλειδιά, δεδομένου ότι οι λίστες μπορούν να τροποποιηθούν στη θέση τους χρησιμοποιώντας αναθέσεις δείκτη, ή μεθόδους όπως οι συναρτήσεις append() και extend().

Είναι καλύτερο να σκέπτεστε ένα λεξικό ως μία μη διατεταγμένη σειρά κλειδιών: ζεύγη τιμών, με την προϋπόθεση ότι τα κλειδιά είναι μοναδικά. Ένα ζευγάρι αγκύλες δημιουργεί ένα άδειο λεξικό:{}. Η τοποθέτηση μιας λίστας κλειδιών διαχωρισμένης με κόμματα: ζεύγη τιμών μέσα στις αγκύλες προσθέτει το αρχικό κλειδί.

Οι κύριες λειτουργίες σε ένα λεξικό είναι η αποθήκευση μιας τιμής με κάποιο κλειδί και η εξαγωγή μιας αξίας σύμφωνα με το κλειδί που δίνεται. Είναι επίσης δυνατό να διαγράψετε ένα κλειδί(ζεύγος αξίας με del). Εάν αποθηκεύσετε μια τιμή χρησιμοποιώντας ένα κλειδί που είναι ήδη σε χρήση, η προηγούμενη τιμή που αντιστοιχίζεται στο κλειδί θα χαθεί. Είναι λάθος να εξαγάγετε μια τιμή, χρησιμοποιώντας ένα ανύπαρκτο κλειδί.

Οι κύριες λειτουργίες σε ένα λεξικό αποθήκευση μιας τιμής με ορισμένα βασικά και εκχύλιση την αξία που δίνεται το κλειδί . Είναι επίσης δυνατό να διαγράψετε ένα κλειδί : ζεύγος αξίας με del . Εάν αποθηκεύσετε χρησιμοποιώντας ένα κλειδί που είναι ήδη σε χρήση , έχει ξεχαστεί η παλιά τιμή που συνδέεται με αυτό το κλειδί . Είναι λάθος να εξαγάγετε μια τιμή, χρησιμοποιώντας ένα ανύπαρκτο κλειδί .

Η εκτέλεση της συνάρτησης λίστας list(d.keys()) σε ένα λεξικό επιστρέφει μια λίστα με όλα τα κλειδιά που χρησιμοποιούνται στο λεξικό, σε αυθαίρετη σειρά (αν θέλετε να ταξινομούνται, απλά χρησιμοποιήστε την sorted(d.keys()), αντ 'αυτού). Για να ελέγξετε αν ένα μόνο κλειδί είναι στο λεξικό, χρησιμοποιείσε τη λέξη-κλειδί in.

Εδώ είναι ένα μικρό παράδειγμα χρήσης ενός λεξικού:

```
>>> tel = {'jack': 4098, 'sape': 4139}
```

```
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> list(tel.keys())
['irv', 'guido', 'jack']
>>> sorted(tel.keys())
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

Ο δημιουργός λεξικών dict() δημιουργεί απευθείας ακολουθίες με ζευγάρια κλειδί-τιμή:

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

Επιπλέον, η έννοια dict μπορεί να χρησιμοποιηθεί για τη δημιουργία λεξικών από αυθαίρετες εκφράσεις που περιλαμβάνουν κλειδιά και αξίες:

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

Όταν τα πλήκτρα είναι απλές συμβολοσειρές, μερικές φορές είναι ευκολότερο να καθορίσετε τα ζεύγη με ορίσματα λέξεις-κλειδία:

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

Όταν σε βρόγχους εντός λεξικών, το κλειδί και η αντίστοιχη αξία μπορεί να ανακτηθεί ταυτόχρονα, χρησιμοποιώντας την μέθοδο items().

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

Όταν σε βρόγχους μιας αλληλουχίας, ο δείκτης θέσης και η αντίστοιχη αξία μπορούν να ανακτηθούν κατά την ίδια χρονική στιγμή χρησιμοποιώντας την συνάρτηση enumerate().

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

Για την επανάληψη δύο ή περισσότερων αλληλουχιών ταυτόχρονα, οι καταχωρήσεις μπορεί να συνδυαστούν με τη συνάρτηση zip().

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q, a))
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue
```

Για την επανάληψη μιας ακολουθίας με την αντίστροφη, πρώτα προσδιορίστε την αλληλουχία σε μία κατεύθυνση προς τα εμπρός και στη συνέχεια καλέστε την συνάρτηση reserved().

```
>>> for i in reversed(range(1, 10, 2)):
```

```
...     print(i)
...
9
7
5
3
1
```

Για επανάληψη μιας ακολουθίας σε ταξινομημένη σειρά, χρησιμοποιήστε την συνάρτηση sorted(), η οποία επιστρέφει μια νέα ταξινομημένη λίστα αφήνοντας αμετάβλητη την πηγή.

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange',
'banana']
>>> for f in sorted(set(basket)):
...     print(f)
...
apple
banana
orange
pear
```

Για να αλλάξετε μια σειρά με κάποια άλλη, στο εσωτερικό του βρόχου (για παράδειγμα, να επαναλάβει ορισμένα στοιχεία), συνιστάται ότι θα πρέπει πρώτα να δημιουργήσετε ένα αντίγραφο.

```
>>> words = ['cat', 'window', 'defenestrate']
>>> for w in words[:]:  # Loop over a slice copy of the
entire list.
...     if len(w) > 6:
...         words.insert(0, w)
...
>>> words
['defenestrate', 'cat', 'window', 'defenestrate']
```

Οι όροι που χρησιμοποιούνται στις while και if δηλώσεις μπορεί να περιέχουν επιχειρήματα και όχι μόνο συγκρίσεις.

Οι τελεστές σύγκρισης in και not in στον ελέγχουν εάν μια τιμή εμφανίζεται (ή δεν εμφανίζεται) σε μια ακολουθία. Οι τελεστές is και is not συγκρίνουν αν δύο αντικείμενα είναι πραγματικά το ίδιο αντικείμενο. Αυτό αφορά μόνο μεταβλητά αντικείμενα όπως οι λίστες. Όλοι οι τελεστές σύγκρισης έχουν την ίδια προτεραιότητα, η οποία είναι χαμηλότερη από εκείνη όλων των αριθμητικών φορέων.

Οι συγκρίσεις μπορούν να συνδυαστούν αλυσιδωτά. Για παράδειγμα, το a < b ==c εξετάζει αν το a είναι μικρότερο από b και επιπλέον, αν b ισούται με c.

Οι συγκρίσεις μπορούν να συνδυαστούν με τη χρήση λογικών τελεστών AND και OR, και το αποτέλεσμα της σύγκρισης (ή οποιασδήποτε άλλης λογικής έκφρασης) μπορεί να εξαλειφθεί με NOT. Αυτά έχουν χαμηλότερες προτεραιότητες από τους τελεστές σύγκρισης. Μεταξύ τους, το NOT έχει την υψηλότερη προτεραιότητα και το OR έχει τη χαμηλότερη, έτσι ώστε A AND NOT B OR C να ισοδυναμεί με (A AND (NOT B)) OR C. Όπως πάντα, μπορούν να χρησιμοποιηθούν παρενθέσεις για να εκφράσουν την επιθυμητή σύνθεση.

Τα επιχειρήματά των λογικών τελεστών αξιολογούνται από αριστερά προς τα δεξιά, και η αξιολόγηση σταματά αμέσως μόλις καθοριστεί το αποτέλεσμα. Για παράδειγμα, εάν το A και C είναι αληθής, αλλά B ψευδής, A AND B AND C δεν αξιολογεί την έκφραση C. Όταν χρησιμοποιείται ως μια γενική αξία και όχι ως λογική, η επιστρεφόμενη τιμή του ενός φορέα βραχυκυκλώματος είναι το τελευταίο.

Είναι δυνατόν να ορίσετε το αποτέλεσμα μιας σύγκρισης ή άλλων λογικών εκφράσεων σε μια μεταβλητή. Για παράδειγμα,

```
>>> string1, string2, string3 = '', 'Trondheim', 'Hammer
Dance'
>>> non_null = string1 or string2 or string3
>>> non_null
'Trondheim'
```

Σημειώστε ότι στην Python, σε αντίθεση με τη C, η εκχώρηση δεν μπορεί να συμβεί μέσα σε εκφράσεις. Στην C οι προγραμματιστές μπορούν να γκρινιάζουν γι 'αυτό, αλλά

αποφεύγεται μια κοινή κλάση των προβλημάτων που ανέκυψαν στα προγράμματα C.

Μια ακολουθία αντικειμένων μπορεί να συγκριθεί με άλλα αντικείμενα με τον ίδιο τύπο αλληλουχίας. Η σύγκριση χρησιμοποιεί λεξικογραφική σειρά: πρώτα τα δύο πρώτα στοιχεία συγκρίνονται και αν διαφέρουν αυτό καθορίζει το αποτέλεσμα της σύγκρισης. Εάν είναι ίσα, τα επόμενα δύο στοιχεία συγκρίνονται και ούτω καθεξής, έως ότου η ακολουθία να εξαντληθεί. Εάν τα δύο στοιχεία που πρέπει να συγκριθούν είναι ίδιες αλληλουχίες του ίδιου τύπου, η λεξικογραφική σύγκριση πραγματοποιείται αναδρομικά.

Εάν όλα τα στοιχεία των δύο αλληλουχιών που συγκρίνονται είναι ίσα, οι ακολουθίες θεωρούνται ίσες. Αν μία αλληλουχία είναι μια αρχική υπο-αλληλουχία του άλλου, η βραχύτερη αλληλουχία είναι η μικρότερη.

```
(1, 2, 3)              < (1, 2, 4)
[1, 2, 3]              < [1, 2, 4]
'ABC' < 'C' < 'Pascal' < 'Python'
(1, 2, 3, 4)           < (1, 2, 4)
(1, 2)                 < (1, 2, -1)
(1, 2, 3)             == (1.0, 2.0, 3.0)
(1, 2, ('aa', 'ab'))   < (1, 2, ('abc', 'a'), 4)
```

Σημειώστε ότι η σύγκριση αντικειμένων διαφορετικών τύπων με <OP> είναι νόμιμη με την προϋπόθεση ότι τα αντικείμενα έχουν τις κατάλληλες μεθόδους σύγκρισης. Για παράδειγμα, οι μικτοί αριθμητικοί τύποι σε σύγκριση σύμφωνα με την αριθμητική τους αξία, οπότε 0 είναι ίση με 0,0, κ.λπ. Διαφορετικά, αντί να παρέχουν μια αυθαίρετη παραγγελία, ο διερμηνέας θα αυξήσει την εξαίρεση TypeError.

**Είσοδοι και έξοδοι**

Υπάρχουν διάφοροι τρόποι για να παρουσιάσει την έξοδο ενός προγράμματος. Τα δεδομένα μπορούν να εκτυπωθούν σε μια μορφή κατανοητή προς τον άνθρωπο ή να καταγράφονται σε ένα αρχείο για μελλοντική χρήση.

Μέχρι στιγμής έχουμε αντιμετωπίσει δύο τρόποι γραφής τιμών, τις δηλώσεις εκφράσεων και τη συνάρτηση print(). ( Ένας τρίτος τρόπος είναι με τη χρήση της write() των αντικειμένων ενός αρχείου. Το πρότυπο αρχείο εξόδου μπορεί να αναφέρεται ως sys.stdout.

Συχνά θα θέλετε περισσότερο έλεγχο πάνω από τη μορφοποίηση της εξόδου από την απλή εκτύπωση διαχωρισμένων τιμών. Υπάρχουν δύο τρόποι για να μορφοποιήσετε την έξοδο σας. Ο πρώτος τρόπος είναι να κάνετε όλα μόνοι σας με τη μέθοδο διαχωρισμού αλφαριθμητικών τη συνένωση μπορείτε να δημιουργήσετε οποιαδήποτε διάταξης μπορείτε να φανταστείτε. Ο τύπος αλφαριθμητικών έχει κάποιες μεθόδους που εκτελούν χρήσιμες λειτουργίες για ένα δεδομένο πλάτος της στήλης. Ο δεύτερος τρόπος είναι να χρησιμοποιήσετε την μέθοδο str.format().

Ένα βασικό ερώτημα είναι το πώς να μετατρέψετε τις τιμές σε strings. Ευτυχώς , η Python έχει τρόποι για να μετατρέψετε οποιαδήποτε τιμή σε μια συμβολοσειρά μέσω των λειτουργιών repr() ή str ().

Η λειτουργία str() έχει ως στόχο να επιστρέψει αναπαραστάσεις των αξιών που είναι αρκετά αναγνώσιμη από τον άνθρωπο, ενώ η repr() έχει ως στόχο να δημιουργήσουν αναπαραστάσεις που μπορεί να διαβαστούν από τον διερμηνέα ή θα οδηγήσει σε SyntaxError αν δεν υπάρχει ισοδύναμη σύνταξη . Για αντικείμενα που δεν έχουν ιδιαίτερη παράσταση στην ανθρώπινη μορφή, η str ( ) θα επιστρέψει την ίδια αξία με την repr( ). Πολλές αξίες , όπως είναι αριθμοί ή δομές , όπως λίστες και λεξικά.

Μερικά παραδείγματα:

```
>>> s = 'Hello, world.'
>>> str(s)
'Hello, world.'
>>> repr(s)
```

```
"'Hello, world.'"
>>> str(1/7)
'0.14285714285714285'
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x) + ', and y is ' +
repr(y) + '...'
>>> print(s)
The value of x is 32.5, and y is 40000...
>>> # The repr() of a string adds string quotes and
backslashes:
... hello = 'hello, world\n'
>>> hellos = repr(hello)
>>> print(hellos)
'hello, world\n'
>>> # The argument to repr() may be any Python object:
... repr((x, y, ('spam', 'eggs')))
"(32.5, 40000, ('spam', 'eggs'))"
```

Εδώ είναι δύο τρόποι για να γράψει έναν πίνακα των τετραγώνων και των κύβων:

```
>>> for x in range(1, 11):
...     print(repr(x).rjust(2), repr(x*x).rjust(3), end=' ')
...     # Note use of 'end' on previous line
...     print(repr(x*x*x).rjust(4))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000

>>> for x in range(1, 11):
...     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
...
 1   1    1
 2   4    8
```

```
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

(Σημειώστε ότι στο πρώτο παράδειγμα, ένα κενό διάστημα ανάμεσα σε κάθε στήλη προστέθηκε από τον τρόπο print(): προσθέτει πάντα κενά μεταξύ των ορισμάτων της.)

Αυτό το παράδειγμα καταδεικνύει τη μέθοδο str.rjust() των αντικειμένων string, η οποία δικαιολογεί μια συμβολοσειρά σε ένα πεδίο ενός δεδομένου πλάτους με γεμίσει με κενά στα αριστερά. Υπάρχουν παρόμοιες μεθόδους str.ljust() και str.center(). Αυτές οι μέθοδοι δεν γράφουν τίποτα, επιστρέφουν απλά μια νέα σειρά. Εάν η συμβολοσειρά εισόδου είναι πολύ μεγάλη, δεν διαμερίζεται, αλλά επιστρέφεται αμετάβλητη. Αυτό θα χαλάσουν στήλη σας, αλλά αυτό είναι συνήθως καλύτερο αποτέλεσμα από την εναλλακτική λύση, η οποία θα βρίσκεται για μια τιμή.

Υπάρχει και μια άλλη μέθοδος, str.zfill(), η οποία γεμίζει μια αριθμητική σειρά στα αριστερά με μηδενικά. Κατανοώντας τα σήματα συν και πλην :

```
>>> '12'.zfill(5)
'00012'
>>> '-3.14'.zfill(7)
'-003.14'
>>> '3.14159265359'.zfill(5)
'3.14159265359'
```

Η βασική χρήση της μεθόδου str.format() μοιάζει με αυτή:

```
>>> print('We are the {} who say "{}!"'.format('knights',
'Ni'))
We are the knights who say "Ni!"
```

Οι παρενθέσεις και οι χαρακτήρες στο εσωτερικό τους (που ονομάζονται πεδία αντικαθίστανται με τα αντικείμενα που περνούν στην μέθοδο στο str.format(). Ένας αριθμός μέσα στις παρενθέσεις μπορεί να χρησιμοποιηθεί για να παραπέμψει στη θέση του αντικειμένου που διέρχεται εντός της μεθόδου str.format().

```
>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

Αν τα ορίσματα της λέξης-κλειδί χρησιμοποιούνται στη μέθοδο str.format(), οι τιμές τους αναφέρονται χρησιμοποιώντας το όνομα του επιχειρήματος.

```
>> print('This {food} is {adjective}.'.format(
...       food='spam', adjective='absolutely horrible'))
This spam is absolutely horrible.
```

Οι θέσης και οι λέξεις-κλειδία μπορούν να συνδυαστούν αυθαίρετα:

```
>>> print('The story of {0}, {1}, and
{other}.'.format('Bill', 'Manfred',

other='Georg'))
The story of Bill, Manfred, and Georg.
```

'!a' → ascii(), '!s' → str(),'!r! → repr() := μπορεί να χρησιμοποιηθούν για τη μετατροπή της αξίας πριν από τη διαμόρφωσή:

```
>>> import math
>>> print('The value of PI is approximately
{}.'.format(math.pi))
The value of PI is approximately 3.14159265359.
>>> print('The value of PI is approximately {!
r}.'.format(math.pi))
```

```
The value of PI is approximately 3.141592653589793.
```

Προαιρετικά μία ':' και ένας προσδιοριστής μορφής μπορεί να ακολουθήσει το όνομα του πεδίου. Αυτό επιτρέπει μεγαλύτερο έλεγχο για το πώς έχει διαμορφωθεί η τιμή. Το ακόλουθο παράδειγμα εκτυπώνει το Pi με ακρίβεια τριών δεκαδικών.

```
>>> import math
>>> print('The value of PI is approximately
{0:.3f}.'.format(math.pi))
The value of PI is approximately 3.142.
```

Περνώντας έναν ακέραιο μετά την ':' το πεδίο θα είναι ένα ελάχιστο αριθμό χαρακτήρων ευρους ίσο με τον ακέραιο. Αυτό είναι αρκετά χρήσιμο για την κατασκευή πινάκων .

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print('{0:10} ==> {1:10d}'.format(name, phone))
...
Jack       ==>       4098
Dcab       ==>       7678
Sjoerd     ==>       4127
```

Εάν έχετε μια πραγματικά ένα μεγάλο αλφαριθμητικό που δεν θέλετε να διαχωριστεί, θα ήταν ωραίο αν θα μπορούσατε να αναφέρετε τις μεταβλητές που πρέπει να διαμορφωθούν με βάση το όνομα τους αντί της θέσης τους. Αυτό μπορεί να γίνει απλά περνώντας τη dict και χρησιμοποιώντας αγκύλες [] να έχουν πρόσβαση τα κλειδιά.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {0[Jack]:d}; Sjoerd: {0[Sjoerd]:d}; '
...       'Dcab: {0[Dcab]:d}'.format(table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

Αυτό θα μπορούσε επίσης να γίνει με το πέρασμα του πίνακα ως όρισμα λέξης κλειδιού με την σημειογραφία '**'.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {Jack:d}; Sjoerd: {Sjoerd:d}; Dcab:
{Dcab:d}'.format(**table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

Αυτό είναι ιδιαίτερα χρήσιμο σε συνδυασμό με τις ενσωματωμένη συνάρτηση vars(), η οποία επιστρέφει ένα λεξικό που περιέχει όλες τις τοπικές μεταβλητές.

Ο τελεστής % μπορεί επίσης να χρησιμοποιηθεί για τη μορφοποίηση των συμβολοσειρών. Ερμηνεύει το αριστερό όρισμα σαν ένα sprintf()-στυλ που πρέπει να εφαρμόζεται πάντα στο σωστό όρισμα, και επιστρέφει το string που προκύπτει από αυτή τη λειτουργία μορφοποίησης. Για παράδειγμα:

```
>>> import math
>>> print('The value of PI is approximately %5.3f.' %
math.pi)
The value of PI is approximately 3.142.
```

**Ανάγνωση και γραφή αρχείων**

Η open() επιστρέφει ένα αντικείμενο του αρχείου. Πιο συχνά χρησιμοποιείται με δύο επιχειρήματα : open(όνομα αρχείου, λειτουργία).

```
>>> f = open('workfile', 'w')
```

Το πρώτο όρισμα είναι ένα string που περιέχει το όνομα του αρχείου. Το δεύτερο όρισμα είναι ένα άλλο string που περιέχει μερικούς χαρακτήρες που περιγράφουν τον τρόπο με τον οποίο θα χρησιμοποιηθεί το αρχείο. Η λειτουργία μπορεί να είναι 'r', όταν το αρχείο θα διαβαστεί μόνο , 'w' για μόνο γραπτώς (εαν υπάρχον αρχείο με το ίδιο όνομα θα

διαγραφούν) , και 'a' ανοίγει το αρχείο για επισύναψη, οποιαδήποτε στοιχεία που γράφονται στο αρχείο προστίθεται αυτόματα στο τέλος. Το 'r+' ανοίγει το αρχείο τόσο για την ανάγνωση όσο και για γραφή ως ορίσμα σε μιας συνάρτησης είναι προαιρετικό. Το 'r' μπορεί να παραλείπεται.

Τα αρχεία ανοίγουν σε μορφή κειμένου. Αυτό σημαίνει, ότι μπορείτε να διαβάσετε και να γράψετε αλφαριθμητικά από και προς το αρχείο , τα οποία είναι κωδικοποιημένα σε μια συγκεκριμένη κωδικοποίηση (η προεπιλογή είναι UTF-8). Το 'b' που επισυνάπτεται στη λειτουργία ανοίγει το αρχείο σε δυαδική κατάσταση: τώρα τα δεδομένα διαβάζονται και να γράφονται με τη μορφή bytes. Αυτή η λειτουργία θα πρέπει να χρησιμοποιείται για όλα τα αρχεία που δεν περιέχουν κείμενο .

Σε λειτουργία κειμένου, η προεπιλογή κατά την ανάγνωση είναι να μετατρέψει την πλατφόρμα ώστε να έχει συγκεκριμένες καταλήξεις γραμμών (\n για Unix , \r\n στα Windows). Όταν γράφετε σε μορφή κειμένου , η προεπιλογή είναι να μετατρέψει τις εμφανίσεις του \n πίσω στην πλατφόρμα σε συγκεκριμένες καταλήξεις γραμμής. Να είστε πολύ προσεκτικοί για να χρησιμοποιήσετε δυαδική κατάσταση κατά την ανάγνωση και τη γραφή των αρχεία σας.

Για τα παρακάτω παραδείγματα, θα υποθέσουμε ότι ένα αντικείμενο που ονομάζεται αρχείο f έχει ήδη δημιουργηθεί.

Για να διαβάσετε το περιεχόμενο ενός αρχείου, καλέστε f.read (sizeς), η οποία διαβάζει κάποια ποσότητα των δεδομένων και τα επιστρέφει ως ένα string ή ένα αντικείμενο σε bytes. το μέγεθος είναι ένας προαιρετικός αριθμός παραμέτρων. Όταν το μέγεθος παραλείπεται ή είναι αρνητική, όλο το περιεχόμενο του αρχείου θα πρέπει να διαβαστεί και να επιστραφεί. Αυτό είναι το πρόβλημά σας, εάν το αρχείο είναι διπλάσιο σε μέγεθος από τη μνήμη του μηχανήματός σας. Σε αντίθετη περίπτωση, τα bytes διαβάζονται και επιστρέφονται. Αν το τέλος του αρχείου έχει επιτευχθεί, f.read() θα επιστρέψει μια κενή συμβολοσειρά ('').

```
>>> f.read()
'This is the entire file.\n'
>>> f.read()
''
```

Η f.readline() διαβάζει μια ενιαία γραμμή από το αρχείο με ένα χαρακτήρα νέας γραμμής (\n) να βρίσκεται στα αριστερά στο τέλος του string, και παραλείπεται μόνο στην τελευταία γραμμή του αρχείου αν το αρχείο δεν τελειώνει σε μια αλλαγή γραμμής. Αυτό κάνει τη τιμή επιστροφής σαφή. Αν η f.readline () επιστρέφει ένα κενό string, το τέλος του αρχείου έχει επιτευχθεί, ενώ μια κενή γραμμή αντιπροσωπεύεται από '\n', ένα string που περιέχει μόνο μία αλλαγή γραμμής.

```
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'Second line of the file\n'
>>> f.readline()
''
```

Για την ανάγνωση των γραμμών από ένα αρχείο, μπορείτε να επαναλάβετε το αντικείμενο του αρχείου. Αυτή είναι η μνήμη αποτελεσματική, γρήγορη, και οδηγεί σε απλό κώδικα:

```
>>> for line in f:
...     print(line, end='')
...
This is the first line of the file.
Second line of the file
```

Αν θέλετε να διαβάσετε όλες τις γραμμές ενός αρχείου σε έναν κατάλογο μπορείτε επίσης να χρησιμοποιήσετε τη list(f) ή την f.readlines().

Η f.write (string) γράφει τα περιεχόμενα της συμβολοσειράς στο αρχείο, επιστρέφει τον αριθμό των χαρακτήρων.

```
>>> f.write('This is a test\n')
15
```

Για να γράψετε κάτι άλλο εκτός των string , θα πρέπει να μετατραπεί σε μια συμβολοσειρά πρώτα:

```
>>> value = ('the answer', 42)
>>> s = str(value)
>>> f.write(s)
18
```

Η f.tell() επιστρέφει έναν ακέραιο δίνει την τρέχουσα θέση του αντικειμένου αρχείου στο αρχείο αντιπροσωπεύεται από έναν αριθμό των bytes από την αρχή του αρχείου, σε δυαδική κατάσταση ,ενώ είναι αδιαφανής, όταν βρίσκεται σε κατάσταση κειμένου.

Για να αλλάξετε τη θέση του αντικειμένου ενός αρχείου, χρησιμοποιήστε f.seek(offset,from_what). Η θέση υπολογίζεται από την προσθήκη και αντισταθμίζεται σε ένα σημείο αναφοράς. Το σημείο αναφοράς επιλέγεται από το όρισμα from_what. Μια from_what τιμή 0 μέτρα από την αρχή του αρχείου, 1 χρησιμοποιεί την τρέχουσα θέση του αρχείου, και 2 χρησιμοποιεί το τέλος του αρχείου ως σημείο αναφοράς. Η from_what μπορεί να παραλειφθεί και η προκαθορισμένη τιμή 0, χρησιμοποιεί την αρχή του αρχείου ως σημείο αναφοράς.

```
>>> f = open('workfile', 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5)      # Go to the 6th byte in the file
5
>>> f.read(1)
b'5'
>>> f.seek(-3, 2) # Go to the 3rd byte before the end
13
>>> f.read(1)
b'd'
```

Στα αρχεία κειμένου (χωρίς ένα b στη σειρά αναμονής), αναζητείται μόνο σε σχέση με την αρχή του αρχείου επιτρέπεται (με εξαίρεση όταν ζητείται το τέλος του αρχείου με αναζήτηση(0, 2)) και η μόνες έγκυρες offset τιμές επιστρέφονται από την f.tell(), ή είναι μηδέν. Οποιαδήποτε άλλη τιμή μετατόπισης παράγει απροσδιόριστη συμπεριφορά.

Όταν τελειώσετε με ένα αρχείο, καλέστε f.close() για να το κλείσετε και να ελευθερώσει τυχόν πόρους του συστήματος που λαμβάνονται από το άνοιγμα του αρχείου. Μετά την κλήση f.close(), αν κάποιο όρισμα χρησιμοποιήσει το αντικείμενο του αρχείου θα αποτύχει αυτόματα.

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

Είναι καλή πρακτική να χρησιμοποιούνται οι λέξεις-κλειδιά όταν ασχολείστε με τα αντικείμενα του αρχείου. Αυτό έχει το πλεονέκτημα ότι το αρχείο έχει κλείσει σωστά μετά την ολοκλήρωση της διεργασίας, ακόμη και αν μια εξαίρεση τεθεί στην πορεία. Επίσης, γραπτώς είναι πολύ μικρότερη από ό,τι ισοδύναμα τελικά τμήματα:

```
>>> with open('workfile', 'r') as f:
...     read_data = f.read()
>>> f.closed
True
```

Οι συμβολοσειρές μπορούν εύκολα να γραφτούν και να διαβαστούν από ένα αρχείο. Οι αριθμοί απαιτούν λίγη περισσότερη προσπάθεια, αφού η μέθοδος read() επιστρέφει μόνο αλφαριθμητικά, τα οποίες θα πρέπει να περάσουν σε μια λειτουργία, όπως η int() , η οποία λαμβάνει ένα string όπως το '123 ' και επιστρέφει την αριθμητική τιμή 123 . Όταν θέλετε να αποθηκεύσετε πιο σύνθετους τύπους δεδομένων, όπως ένθετες λίστες τα λεξικά , η ανάλυση σε συνέχειες γίνεται περίπλοκη.

Η Python σας επιτρέπει να χρησιμοποιήσετε το δημοφιλές μορφή ανταλλαγής δεδομένων που ονομάζεται JSON ( JavaScript Object Notation). Η βασική λειτουργική μονάδα που ονομάζεται json μπορεί να πάρει ιεραρχίες δεδομένων Python, και να τις μετατρέψει τους σε παραστάσεις συμβολοσειρών. Η διαδικασία αυτή ονομάζεται Serializing. Η ανακατασκευή των δεδομένων από την αναπαράσταση συμβολοσειράς ονομάζεται deserializing . Ανάμεσα σε serializing και deserializing , το string που αντιπροσωπεύει το αντικείμενο μπορεί να έχουν αποθηκευτεί σε ένα αρχείο ή δεδομένα , ή αποστέλλονται μέσω μιας σύνδεσης δικτύου σε κάποιο μακρινό μηχάνημα.

Σημείωση Η μορφή JSON χρησιμοποιείται συνήθως από τις σύγχρονες εφαρμογές και επιτρέπει την ανταλλαγή δεδομένων. Πολλοί προγραμματιστές είναι ήδη εξοικειωμένοι με αυτό, το οποίο είναι μια καλή επιλογή για τη διαλειτουργικότητα .

Εάν έχετε ένα αντικείμενο x , μπορείτε να δείτε τη JSON αναπαράσταση ως συμβολοσειρά με μια απλή γραμμή κώδικα :

```
>>> json.dumps([1, 'simple', 'list'])
'[1, "simple", "list"]'
```

Μια άλλη παραλλαγή των συναρτήσεων dumps(), που ονομάζεται χdump(), κάνει serialize απλά το αντικείμενο σε ένα αρχείο κειμένου. Έτσι, αν η f είναι ένα αντικείμενο αρχείο κειμένου που άνοιξε για τη γραφή, μπορούμε να το κάνουμε αυτό:

```
json.dump(x, f)
```

Για να αποκωδικοποιήσει το αντικείμενο ξανά, αν η f είναι ένα αντικείμενο αρχείο κειμένου που έχει ανοίξει ανάγνωση:

```
x = json.load(f)
```

Αυτή η απλή τεχνική μπορεί να χειριστεί serialize λίστες και λεξικά, αλλά σε συνέχεις αυθαίρετες περιπτώσεις κατηγορίας σε JSON απαιτεί λίγη επιπλέον προσπάθεια.

Μετάφραση από: http://docs.python.org/3.3/tutorial/index.html