ΜΕΜ104 Γλώσσα Προγραμματισμού Ι

Μιχάλης Πλεξουσάκης

1 Δεκεμβρίου 2019

Μαθηματικά και Εφαρμοσμένα Μαθηματικά

Περιεχόμενα

Συμπερίληψη λίστας

Συμπερίληψη λίστας

Αν θέλαμε να κατασκευάσουμε μια λίστα με τα τετράγωνα των ακέραιων αριθμών στο διάστημα $[1, 20]$ θα μπορούσαμε να γράψουμε, για παράδειγμα,

```
squares = []
for i in range(1,21):
    squares.append(i**2)
```

Ένας συντομότερος τρόπος ο οποίος επιτρέπει την εφαρμογή μιας πράξης στις τιμές μιας λίστας ή γενικότερα μιας ακολουθίας είναι η λεγόμενη *συμπερίληψη λίστας* (list comprehension):

```
squares = [i**2 for i in range(1, 21)]
```

Συμπερίληψη λίστας

Ο όρος `for` μπορεί να ακολουθείται από μια ή περισσότερες εντολές `if` ή `for` οι οποίες εφαρμόζονται στις τιμές της ακολουθίας που παράγονται από την πρώτη εντολή `for`. Για παράδειγμα, η λίστα `squares` μετά την εκτέλεση της εντολής

```
squares = [i**2 for i in range(1, 21) if i%3 == 0]
```

θα είναι η `[9, 36, 81, 144, 225, 324]`.

Ακόμα, οι εντολές

```
L = [2, 'a', 3.1, 3, 'maria', 4.1, -9.0, 5]
print([x**2 for x in L if type(x) == int])
```

θα εκτυπώσουν τη λίστα `[4, 9, 25]`.

Συμπερίληψη λίστας

Μπορούμε να εξάγουμε τα στοιχεία των φωλιασμένων λιστων από μια λίστα με

```
>>> L = [[1, 2, 3], [4, 5, 6], [7, 8]]
>>> M = [x for Y in L for x in Y]
>>> M
[1, 2, 3, 4, 5, 6, 7, 8]
```

Αν η λίστα περιέχει άλλα στοιχεία, όχι λίστες, χρησιμοποιούμε την εντολή `isinstance` για να τα προσπεράσουμε:

```
>>> L = [[1, 2, 3], 3.5, 'a', [4, 5, 6], [7, 8], 9]
>>> M = [x for Y in L
         if isinstance(Y, list) for x in Y]
>>> M
[1, 2, 3, 4, 5, 6, 7, 8]
```

Πλειάδες

Πλειάδες

Οι πλειάδες (tuples) είναι διατεταγμένες ακολουθίες στοιχείων οποιουδήποτε τύπου. Οι τιμές τύπου tuple γράφονται ως στοιχεία διαχωρισμένα με κόμματα μέσα σε παρενθέσεις:

```
t1 = ()
t2 = (1,)  # <-- Προσοχή!
t3 = (1, 'two', 3)
t4 = ('MEM104', 'Python I', ('Tuesday', 'Thursday'))
```

Σε αντίθεση με τις λίστες, οι πλειάδες είναι *μη μεταλλάξιμες* (immutable), δηλαδή, αντικείμενα τύπου tuple δεν μπορούν να τροποποιηθούν μετά τη δημιουργία τους.

Πλειάδες

Όπως και οι λίστες, οι πλειάδες μπορούν να συνενώνονται να προσπελάζονται με αριθμοδείκτες και να τεμαχίζονται:

```
>>> t1 = (1, 'two', 3)
>>> t2 = ('Mary', 2005, 'George', 2001)
>>> print(t1+t2)
(1, 'two', 3, 'Mary', 2005, 'George', 2001)
>>> print((t1+t2)[2])
3
>>> print((3*t2)[6])
George
```

```
>>> t1[1] = 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Πλειάδες

Η συνάρτηση που ακολουθεί επιστρέφει τους διαιρέτες ενός θετικού ακέραιου n:

```
def divisors(n):
    d = ()
    for i in range(1,n+1):
        if n % i == 0:
            d += (i,)
    return d
```

και οι εντολές που ακολουθούν εκτυπώνουν το άθροισμά τους

```
n = 220;  d = divisors(n)
s = 0
for e in d: s += e
print('sum of divisors of', n, 'is =', s)
```

Πλειάδες

Ο μηχανισμός πολλαπλής ανάθεσης τιμών χρησιμοποιείται συχνά με ακολουθίες σταθερού μεγέθους όπως στις αναθέσεις

```
x, y = (3, 4)
a, b, c = 'xyz'
```

ή με τις συναρτήσεις που επιστρέφουν περισσότερες από μια τιμές

```
def minmax(L):
    minL = L[0];  maxL = L[0]
    for e in L:
        if e < minL: minL = e
        if e > maxL: maxL = e
    return (minL, maxL)

L = [13, 7, 12, 47, 11, 9]
minL, maxL = minmax(L)
```

Λίστες, πλειάδες και συμβολοσειρές

Τόσο οι συμβολοσειρές (str) όσο και οι λίστες (list) αλλά και οι πλειάδες (tuple) αποτελούν διαφορετικούς τύπους ακολουθιών (sequences).

Ένα κοινό χαρακτηριστικό που έχουν αυτοί οι τύποι είναι ότι μπορούμε να εφαρμόσουμε στα αντικείμενά τους τις παρακάτω πράξεις:

- `seq[i]` για το `i`-στό στοιχείο της ακολουθίας
- `seq[i:j]` για τους χαρακτήρες από τη θέση `i` μέχρι και τη θέση `j-1`
- `len(seq)` για το μήκος της ακολουθίας
- `seq1 + seq2` για τη συνένωση των δύο ακολουθιών
- `n * seq` για την επανάληψη της ακολουθίας `seq` `n` φορές

Λίστες, πλειάδες και συμβολοσειρές

αλλά και τις

- `e in seq` επιστρέφει `True` αν `e` περιέχεται στην ακολουθία `seq`
- `e not in seq` επιστρέφει `True` αν `e` δεν περιέχεται στην ακολουθία `seq`
- `for e in seq` για τη σειριακή διάνυση της ακολουθίας `seq`

Λίστες, πλειάδες και συμβολοσειρές

Ειδικότερα, για τις συμβολοσειρές, οι οποίες είναι μη μεταλλάξιμες ακολουθίες χαρακτήρων, η Python έχει πολλές ενσωματωμένες μεθόδους, όπως τις παρακάτω:

`s.count(t)` Μετράει πόσες φορές εμφανίζεται η συμβολοσειρά `t` μέσα στη συμβολοσειρά `s`

```
>>> s = 'Python Programming'
>>> print(s.count('o'))
```

`s.find(t)` Επιστρέφει τον δείκτη θέσης της πρώτης παρουσίας της συμβολοσειράς `t` στη συμβολοσειρά `s`, και -1 αν η `t` δεν υπάρχει μέσα στην `s`

```
>>> frst = s.find('ro')
```

Λίστες, πλειάδες και συμβολοσειρές

`s.rfind(t)` Όμοια με τη `find` αλλά ξεκινώντας από το τέλος της συμβολοσειράς `s`.

```
>>> s = 'Python Programming'
>>> last = s.rfind('n')    # returns 16
```

`s.index(t)` Όμοια με την `find` αλλά με μήνυμα λάθους (exception) αν η `t` δεν εμφανίζεται στην `s`

```
>>> s.index('x')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

Λίστες, πλειάδες και συμβολοσειρές

`s.rindex(t)` Όμοια με την `index` αλλά ξεκινώντας από το τέλος της `s`

`s.lower()` Μετατρέπει σε πεζά όλα τα κεφαλαία γράμματα στη συμβολοσειρά `s`

```
>>> print('YouTube'.lower())
youtube
```

`s.upper()` Μετατρέπει σε κεφαλαία όλα τα πεζά γράμματα στη συμβολοσειρά `s`

```
>>> print('She said: ' + 'Stop it!'.upper())
She said: STOP IT!
```

Λίστες, πλειάδες και συμβολοσειρές

`s.replace(old, new)` Αντικαθιστά όλες τις παρουσίες της συμβολοσειράς `old` στην `s` με τη συμβολοσειρά `new`

```
>>> fruit = 'Strawberry'
>>> print(fruit.replace('r', 'R'))
StRawbeRRy
```

`s.strip()` Αφαιρεί λευκούς χαρακτήρες (white space) από την αρχή και το τέλος της συμβολοσειράς `s`. Χρήσιμο στην είσοδο δεδομένων

```
name = input('Enter your name: ').strip()
```

`s.rstrip()` αφαιρεί λευκούς χαρακτήρες μόνο από το τέλος της συμβολοσειράς `s`.

Λίστες, πλειάδες και συμβολοσειρές

`s.split(d)` Χωρίζει τη συμβολοσειρά `s` σε μέρη, με τον χαρακτήρα `d` ως διαχωριστικό, και επιστρέφει μια λίστα με τις υπο-συμβολοσειρές της `s`.

Αν το `d` παραληφθεί τότε νοείται ως διαχωριστικό οποιοσδήποτε λευκός χαρακτήρας (κενό, tab, χαρακτήρας αλλαγής γραμμής). Για παράδειγμα:

```
>>> s = 'Python is the most popular language'
>>> s.split()
['Python', 'is', 'the', 'most', 'popular', 'language']

>>> date = '01/12/2020'
>>> date.split('/')
['01', '12', '2020']
```

Αρχεία

Αρχεία

Ένα αρχείο (file) είναι ένα σύνολο από πληροφορίες ή δεδομένα, οργανωμένα με συγκεκριμένο τρόπο, διαθέσιμο στο λειτουργικό σύστημα του υπολογιστή ή σε χρήστες του για επεξεργασία.

Υπάρχουν διαφορετικοί τύποι αρχείων, σχεδιασμένοι για διαφορετικούς σκοπούς. Ένα αρχείο μπορεί να αποθηκεύει μια εικόνα, ένα γραπτό μήνυμα, ένα βίντεο, ένα πρόγραμμα υπολογιστή ή μια μεγάλη ποικιλία άλλων ειδών δεδομένων.

Ορισμένοι τύποι αρχείων μπορούν να αποθηκεύσουν πολλούς τύπους πληροφοριών ταυτόχρονα.

Χρησιμοποιώντας προγράμματα, μπορεί κανείς να ανοίξει (open), να διαβάσει (read), να αλλάξει (edit), να αποθηκεύσει (save) και να κλείσει (close) ένα αρχείο.

Αρχεία

Τα αρχεία στα περισσότερα σύγχρονα συστήματα αρχείων αποτελούνται από τρία κύρια μέρη:

- Την κεφαλίδα (header): Περιέχει μετα-δεδομένα σχετικά με τα περιεχόμενα του αρχείου, π.χ., όνομα αρχείου, μέγεθος, τύπος κ.λπ.
- Τα δεδομένα (data): τα περιεχόμενο του αρχείου
- Ο χαρακτήρας τέλος αρχείου (EOF): ειδικός χαρακτήρας που δείχνει το τέλος του αρχείου

Αρχεία

Η Python παρέχει συναρτήσεις για τη δημιουργία και την προσπέλαση αρχείων. Αυτές οι συναρτήσεις χρησιμοποιούν τα λεγόμενα *χειριστήρια αρχείων* (file handles). Η εντολή

```
fh = open('myfile.txt', 'r')
```

υποδεικνύει στο λειτουργικό σύστημα να ανοίξει το αρχείο με το όνομα `myfile.txt` για ανάγνωση (reading).

Επειδή η Python χειρίζεται τα αρχεία ως μια ακολουθία γραμμών, μπορούμε να χρησιμοποιήσουμε μια εντολή `for` για να προσπελάσουμε τα περιεχόμενα του αρχείου:

```
for line in fh:
  print(line)
fh.close()
```

Αρχεία

Ένας δεύτερος τρόπος να ανοίξουμε ένα αρχείο για ανάγνωση είναι με την εντολή `with`:

```
with open('myfile.txt', 'r') as fh:
  for line in fh:
    print(line)
```

Το πλεονέκτημα είναι ότι δεν χρειάζεται να καλέσουμε την `close()` ακόμα και στην περίπτωση σφάλματος!

Η δεύτερη παράμετρος της `open` θα μπορούσε να είναι '`w`' για να γράψουμε ή '`a`' για να προσθέσουμε δεδομένα σε ένα αρχείο.

Αρχεία

Σημαντικό: Η προεπιλεγμένη κωδικοποίηση (encoding) των χαρακτήρων σε ένα αρχείο εξαρτάται από την πλατφόρμα. Στα λειτουργικό σύστημα Windows, είναι η cp1252, ενώ στο macos και στο linux ή utf-8.

Συνεπως, όταν εργαζόμαστε με αρχεία κειμένου, συνιστάται να καθορίζετε τον τύπο κωδικοποίησης, για παράδειγμα

```
fh = open('myfile.txt', mode = 'r',
          encoding = 'utf-8')
```

Αρχεία

Είναι το ίδιο εύκολα να γράψουμε δεδομένα σε ένα αρχείο:

```
fh = open('names.txt', 'w')
for i in range(10):
    name = input('Enter your name: ')
    fh.write(name)
fh.close()
```

Μπορούμε επίσης να προσαρτήσουμε δεδομένα σε ένα ήδη υπάρχον αρχείο χρησιμοποιώντας το όρισμα `'a'` στην εντολή `open`:

```
fh = open('names.txt', 'a')
fh.write('Maria')
fh.write('John')
fh.close()
```

Αρχεία

Άλλες χρήσιμες συναρτήσεις για την προσπέλαση και επεξεργασία αρχείων είναι οι

`fh.read()`. Επιστρέφει μια συμβολοσειρά που περιέχει τα περιεχόμενα του αρχείου που σχετίζεται με το χειριστήριο αρχείου `fh`. Για παράδειγμα, οι εντολές

```
fh = open('names.txt', 'r')
s = fh.read()
fh.close()
print(s)
```

θα τυπώσουν τα περιεχόμενα του αρχείου `names.txt`.

`fh.readline()`. Επιστρέφει την επόμενη γραμμή του αρχείου που σχετίζεται με το χειριστήριο αρχείου `fh`.

Αρχεία

Για παράδειγμα, θα μπορούσαμε να χρησιμοποιήσουμε τον παρακάτω κώδικα Python για να διαβάσουμε ένα αρχείο:

```
fh = open('names.txt', 'r')
while True:
    line = fh.readline()
    if not line:
        break
#   Do something with line
    ...

fh.close()
```