Εκτυπώνοντας τα περιεχόμενα μιας λίστας

Επειδή μια κεντρική δομή δεδομένων που υποστηρίζει η Python είναι οι λίστες, η γλώσσα επιτρέπει την γρήγορη και εύκολη δημιουργία από δυναμικές σύνθετες λίστες που εμπεριέχουν άλλες λίστες χωρίς περιορισμό βάθους. Πρόσβαση στα στοιχεία κάθε λίστας γίνεται με χρήση δεικτών. Θα δούμε απλούς και πιο σύνθετους αλγορίθμους για να αποκτήσουμε πρόσβαση στα μεμονωμένα στοιχεία από σύνθετες λίστες. Σαν παράδειγμα θα πάρουμε την παρακάτω σύνθετη λίστα:

lista =[1, [2, [3, 4], [5, [6, 7]]]]

1 2 3 3 4 5 5 4 2 1

Παρά τους πολλούς ακέραιους που φαίνεται να περιέχει, αυτή η λίστα μας, αποτελείται από μόνο δύο στοιχεία: έναν ακέραιο, το 1, και μία υπολίστα που εμπεριέχει μεμονωμένους ακέραιους και άλλες υπολίστες nested, οπότε το μέγεθος της λίστας, len(lista) επιστρέφει 2: το στοιχείο lista[0] και το lista[1].

```
>>> lista =[ 1, [ 2, [ 3, 4 ], [ 5, [ 6, 7 ] ] ] ]
>>> len(lista)
2
>>> lista[0]
1
>>> lista[1]
[2, [3, 4], [5, [6, 7]]]
>>>
```

Το πρώτο πρόγραμμα αυτής της σειράς διατρέχει τη λίστα με ένα for και τυπώνει ένα-ένα τα περιεχόμενα της:

```
# printList01

def printList(lst):
    for n in lst:
        print(n)

lista =[ 1, [ 2, [ 3, 4 ], [ 5, [ 6, 7 ] ] ] ]
```

```
printList(lista)
```

Όπως αναμενόταν βάσει αυτών που είδαμε το πρόγραμμα αυτό τυπώνει τα δύο στοιχεία που επιστρέφει η for n in lst.

```
============ RESTART: C:/printlist01.py ============
1
[2, [3, 4], [5, [6, 7]]]
>>>
```

Έχουμε μάθει ότι η Python διαθέτει την μέθοδο isinstance, η οποία μπορεί να μας ενημερώσει αν το στοιχείο που έχει επιλεγεί είναι λίστα. Αν χρησιμοποιήσουμε την isinstance μπορούμε να εντοπίσουμε εάν ένα στοιχείο είναι λίστα και να αναλύσουμε τα στοιχεία που περιέχει τυπώνοντας τα. Στο παράδειγμα μας το πρώτο στοιχείο δεν είναι λίστα και το δεύτερο είναι.

```
# printList02

def printList(lst):
    for n in lst:
        if(isinstance(n, list)):
           for m in n:
               print(m)
        else:
            print(n)
```

Βλέπουμε ότι η λίστα που βρίσκεται στο δεύτερο στοιχείο της αρχικής λίστας έχει αναλυθεί κάπως καλύτερα. Το πρώτο του στοιχείο είναι ένας ακέραιος και τα υπόλοιπα λίστες. Η πρώτη από αυτές περιέχει δύο ακέραιους το 3 και το 4, και η δεύτερη έναν ακέραιο και άλλη μία λίστα:

```
========== RESTART: C:/printlist02.py ==========
1
2
[3, 4]
[5, [6, 7]]
>>>
```

Ο αλγόριθμος είναι απλός: ελέγχουμε αν κάθε στοιχείο που επιστρέφει η for-in είναι λίστα ή όχι. Αν δεν είναι, το τυπώνουμε, ενώ εάν είναι, τότε με ένα δεύτερο nested for-in απομονώνουμε ένα-ένα τα στοιχεία από τα οποία βρίσκονται στην λίστα αυτή και ούτω καθεξής μέχρι να απομονώσουμε όλα τα περιεχόμενα της κάθε υπολίστας.

```
# printList03

def printList(lst):
    for n in lst:
        if(isinstance(n, list)):
           for m in n:
                if(isinstance(m, list)):
                   for o in m:
                       print(o)
                else:
                    print(m)
        else:
            print(n)
```

Βλέπουμε ότι η λίστα που βρίσκεται στο τρίτο στοιχείο της αρχικής λίστας έχει αναλυθεί πλήρως και έχουμε απομονώσει τις τιμές 3 και 4. Επίσης, επειδή το πρώτο στοιχεί της επόμενης λίστας δεν είναι λίστα τυπώνεται κανονικά και μένει μία λίστα με δύο στοιχεία: [6, 7]:

```
========== RESTART: C:/printlist03.py ==========
1
2
3
4
5
[6, 7]
>>>
```

Η συγκεκριμένη λίστα χρειάζεται ακόμη ένα επίπεδο επεξεργασίας, έτσι ώστε απομονώσουμε τα στοιχεία που βρίσκονται στην τελική λίστα [6, 7] που δεν έχει αναλυθεί ακόμα:

```
# printList04

def printList(lst):
    for n in lst:
        if(isinstance(n, list)):
           for m in n:
                if(isinstance(m, list)):
                    for o in m:
                        if(isinstance(o, list)):
                            for p in o:
                                print(p)
                        else:
                            print(o)
                else:
                    print(m)
        else:
            print(n)

lista =[ 1, [ 2, [ 3, 4 ], [ 5, [ 6, 7 ] ] ] ]
printList(lista)
```

Βλέπουμε ότι τα στοιχεία από όλα τα επίπεδα έχουν αναλυθεί:

```
========== RESTART: C:/printlist04.py ==========
1
2
3
4
5
6
7
>>>
```

Ενώ το πρόγραμμα αυτό τρέχει σωστά, δίνει τα αποτελέσματα που θέλουμε και μπορεί να αντιμετωπίσει και απλές λίστες, εάν του περάσουμε λίστα με ένα βαθύτερο επίπεδο από υπολίστες επιπλέον, τότε θα χρειαστεί ένα ακόμα επίπεδο ελέγχου έτσι ώστε να μην επιστρέψει λίστα. Προφανώς η σχεδίαση αυτή ενώ λειτουργεί, έχει όρια και είναι απαράδεκτα πολύπλοκη εκτός από περιοριστική.

Η αναδρομική λύση απλοποιεί τον αλγόριθμο και του δίνει την δυνατότητα να επεξεργάζεται λίστες με υπολίστες σε οποιοδήποτε βάθος. Αναδρομή έχουμε όταν μία συνάρτηση καλεί τον εαυτό της.

```
# printListRecu

def printList(lst):
    for n in lst:
        if(isinstance(n, list)):
            printList(n)
        else:
            print(n)


lista =[ 1, [ 2, [ 3, 4 ], [ 5, [ 6, 7 ] ] ] ]
printList(lista)
```