Σημειώσεις

Προηγούμενο Επόμενο

Ανακύκλωση

Η εντολή while

Η εντολή αναλύκλωσης `while` περιγράφει μία επαναληπτική διαδικασία και το συντακτικό της είναι:

```
while expression:
    statements
```

Οι εντολές *statements* εκτελούνται εφόσον η έκφραση *expression* είναι αληθής. Ο τρόπος ομαδοποίησης των εντολών ανακύκλωσης *statements* είναι πάλι η εσοχή (tab).

Παράδειγμα. Εισάγετε έναν ακέραιο *m* και τυπώστε σε διαδοχικές γραμμές τους ακεραίους από το 0 έως τον *m*.

```
max = int(input("Give a positive integer: "))
i = 0
while i <= max:
    print(i)
    i = i + 1
```

Παράδειγμα. Θα βελτιώσουμε ένα προηγούμενο πρόγραμμά μας ώστε να βρούμε την κυβική ρίζα τέλειου κύβου με τη μέθοδο της *εξαντλητικής απαρίθμησης.*

```
perfect_cube = 216   # this is a perfect cube
x = 0
while x**3 < perfect_cube:
    x = x + 1
    if x**3 < perfect_cube:
        print("No,", x,"is not the cubic root of",perfect_cube)
```

```
    elif x**3 == perfect_cube:
        print("Yes, indeed!", x,"is the cubic root of",perfect_cube)
```

Παράδειγμα. Εύρεση ρίζας θετικού αριθμού με τον *αλγόριθμο του Ήρωνα.*

```
x = input("Give a number")
x = float(x)
g = 1.0                 # first approximation
while abs(g*g - x) > 1.0e-5:
    g = (g+x/g)/2.0
print("The square root of",x,"is approximately",g)
```

Συμβολοσειρές. Μπορούμε να εξαγάγουμε έναν-έναν τους χαρακτήρες μιας συμβολοσειράς.

```
s = input("Give a string: ")
i = 0
while i < len(s):
    print(i,"character is",s[i])
    i = i + 1
```

Παρατήρηση: η εντολή `i += 1` είναι ισοδύναμη με την `i = i + 1`. Γενικότερα, η `x += y` είναι ισοδύναμη με την `x = x + y`. Επίσης υπάρχουν και οι τελεστές -=, *=, /=, //=. **= με ανάλογες σημασίες. Το προηγούμενο πρόγραμμα γράφεται ως

```
s = input("Give a string: ")
i = 0
while i < len(s):
    print(i,"character is",s[i])
    i += 1
```

Εντολή `break`

Μπορούμε να τερματίσουμε μία ανακύκλωση πρόωρα με την εντολή `break`.

```
s = input("Give a string: ")

i = 0
while i < len(s):
    if s[i] == 'o':
        print("The character o is in position",i)
    i = i + 1
if i == len(s):
    print("There is no o in the word", s)
```

Μπορούμε να παραγάγουμε *ατέρμονη ανακύκλωση* αν στη θέση της συνθήκης για την `while` βάλουμε την boolean σταθερά `True`. Η εντολή `break` είναι απαραίτητη στις ατέρμονες ανακυκλώσεις.

Παράδειγμα. Θα ζητήσουμε από τον χρήστη να εισάγει έναν αριθμό από το 1 έως το 100 και θα επαναλαμβάνουμε την προτροπή εάν ο χρήστης δίνει αριθμούς εκτός αυτού του διαστήματος.

```
while True:
    x = int(input("Input a number between 1 and 100: "))
    if  x >= 1 and x <= 100:
        break
print("Well done!")
```

Εντολή `continue`

Οι εντολές της ανακύκλωσης δεν εκτελούνται και η ανακύκλωση επαναλαμβάνεται όταν στη ροή του προγράμματος συναντήσουμε την εντολή `continue`.

Παράδειγμα. Θα εισαγάγουμε αριθμούς στο πρόγραμμα και θα βρούμε το άθροισμα των πρώτων 10 θετικών αριθμών (αγνοώντας τους αρνητικούς).

```
iter = 0
summ = 0
while iter < 10:
    x = int(input('Enter a positive integer: '))
    if x <= 0:
```

```
            continue
        summ = summ + x
        iter = iter + 1
    print('The sum of the positive numbers is', sum)
```

Ανακύκλωση με την εντολή for

Έχουμε χρησιμοποιήσει ορισμένες φορές ανακύκλωση όπου μία μεταβλητή (μετρητής) παίρνει τιμές από μια ακολουθία ακεραίων.

Η εντολή `for` μπορεί να χρησιμοποιηθεί για να σαρώσουμε επαναληπτικά μία ακολουθία ακεραίων (η μία οποιαδήποτε ακολουθία). Για παράδειγμα, θα μπορούμε να τυπώσουμε τα τετράγωνα των ακεραίων από το 1 έως το 10.

```
for i in range(1,11):
    print('The square of', i, 'is', i**2)
```

Η μεταβλητή `i` που εμφανίζεται μετά τη λέξη-κλειδί `for` λαμβάνει διαδοχικά τις τιμές της ακολουθίας `range(1,11)` και η εντολή `print(i)` εκτελείται. Η διαδικασία επαναλαμβάνεται μέχρι να εξαντληθούν οι τιμές της ακολουθίας.

Το συντακτικό της εντολής `for` είναι

```
for variable in sequence:
    statements
```

Ο τελεστής `in` δίνει στη μεταβλητή *variable* της τιμές που περιέχονται στην ακολουθία *sequence* με σειριακό τρόπο (την μία μετά την άλλη). Οι εντολές *statements* εκτελούνται για κάθε τιμή της *variable* και η διαδικασία επαναλαμβάνεται μέχρι να εξαντληθούν οι τιμές της ακολουθίας. Εάν θελήσουμε να τερματήσουμε πρόωρα την ανακύκλωση μπορούμε να το κάνουμε με την εντολή `break` (όπως είδαμε και στην `while`).

Παράδειγμα. Για την εύρεση κυβικής ρίζας (τέλειου κύβου, έστω *pc*) με εξαντλητική απαρίθμηση σαρώσαμε μία σειρά ακεραίων από το 1 έως το πολύ *pc*.

```
pc = int(input("Give an integer: "))
```

```
for x in range(0,pc+1):
    if x**3 == abs(pc):
        print("The cubic root of",pc,"is",x)
        break
```

Μία βελτίωση του παραπάνω προγράμματος είναι η ακόλουθη (σκεφτείτε γιατί αυτό αποτελεί βελτίωση).

```
pc = int(input("Give an integer: "))
for x in range(0,abs(pc)+1):
    if x**3 >= abs(pc):
       break

if x**3 != abs(pc):
   print(pc,"is not a perfect cube")
else:
   if pc < 0:
        x = -x
   print("The cubic root of",pc,"is",x)
```

Σάρωση λιστών και συμβολοσειρών

Παράδειγμα. Ας βρούμε το άθροισμα των στοιχείων μίας λίστας (της οποίας τα στοιχεία είναι αριθμοί).

```
L = [1,2,3,4,5,6,8,9]

sumL = 0
for num in L:
    sumL += num

print(sumL)
```

Παράδειγμα. Μία λίστα λέγεται παλινδρομική αν η σειρά των στοιχείων της είναι η ίδια είτε διαβάζεται από αριστερά προς τα δεξιά είτε αντίστροφα. Ας ελέγξουμε αν μία λίστα είναι παλινδρομική.

```
n = len(L)
flag = True
for i in range(0,n//2):
    if L[i] != L[n-i-1]:
        flag = False
        break

print(flag)
```

- Μέσα στην ανακύκλωση ελέγχουμε αν το στοιχείο το οποίο βρίσκεται *i* θέσεις μετά το πρώτο (δηλαδή, το `L[i]`) είναι ίδιο με αυτό που βρίσκεται *i* θέσεις πριν το τελευταίο (δηλαδή, το `L[n-1-i]`).
- Αρκεί να διατρέξουμε τη λίστα μέχρι την μέση (`n//2`) αφού πάντα κάνουμε συγκρίσεις ενός στοιχείου στο πρώτο μισό με στοιχείο στο δεύτερο μισό της λίστας. Αν η λίστα έχει περιττό αριθμό στοιχείων δεν χρειάζεται να γίνει έλεγχος στο μεσαίο στοιχείο.
- Χρησιμοποιήσαμε μία λογική μεταβλητή `flag` η οποία παίρνει την τιμή `False` αν έστω και δύο αντίστοιχα γράμματα δεν είναι ίδια. Η `flag` διετηρεί την αρχική τιμή της `True` αν ποτέ δεν πάρει την τιμή `False` εντός του `for`.

Ερώτηση. Πώς θα μπορούσαμε να ελέγξουμε αν μία λίστα είναι παλινδρομική με χρήση της μεθόδου `reverse`;

strings ως ακολουθίες χαρακτήρων. Οι μεταβλητές τύπου string είναι ακολουθίες χαρακτήρων και μπορούμε να γράψουμε εντολές ανακύκλωσης που σαρώνουν τους χαρακτήρες μιας ακολουθίας χαρακτήρων:

Παράδειγμα. Ας τυπώσουμε έναν-έναν με τη σειρά τούς χαρακτήρες μίας συμβολοσειράς.

```
s = 'Nick Papadakis'
for c in s:
    print(c)
```

Μπορούμε επίσης να διατρέξουμε την συμβολοσειρά χρησιμοποιώντας τους δείκτες των χαρακτήρων.

```
s = 'Nick Papadakis'
for i in range(len(s)):
    print('The',i,'th character is',c)
```

Παράδειγμα. Θα διατρέξουμε (σαρώσουμε) τη λίστα `colors`, θα αφαιρέσουμε τα χρώματα που δεν μας αρέσουν και θα προσθέσουμε ορισμένα που μας αρέσουν.

```
colors = ['red','green','blue','cyan','magenta','yellow','black']

colors1 = colors[:]
for c in colors1:
    yesno = input("Should I keep " + c + "? (y/n) ")
    if yesno == 'n': colors.remove(c)

while True:
    newColor = input("Should I add a new color? (color/n) ")
    if newColor != 'n':
        colors.append(newColor)
    else:
        break

print("Your new list of colors is\n",colors)
```

Παράδειγμα. Γράψτε ένα πρόγραμμα το οποίο ελέγχει αν δυο λίστες έχουν τουλάχιστον ένα κοινό στοιχείο. Εδώ θα χρειαστεί να ελέγχουμε τα στοιχεία της δεύτερης λίστας, ενώ διατρέχουμε την πρώτη λίστα. Αυτό θα το πετύχουμε με την χρήση δύο ενθυλακωμένων `for`.

```
L1 = [1,2,3,4,5,6]
L2 = [5,6,7,8,9]

flag = False
for e1 in L1:
    for e2 in L2:
         if e1 == e2:
           flag = True
           break
```

```
if flag: print('Yes')
else: print('No')
```

Η python μας προσφέρει έναν απλό τρόπο για να ελέγχουμε αν ένα στοιχείο υπάρχει σε μία λίστα. Γράφουμε για στοιχείο `e` σε λίστα `L` την έκφραση: `e in L`. Η έκφραση αυτή παίρνει τις τιμές `True, False`. Το προηγούμενο πρόγραμμα γράφεται με απλούστερο τρόπο ως εξής:

```
L1 = [1,2,3,4,5,6]
L2 = [5,6,7,8,9]

flag = False
for e in L1:
    if e in L2:
        flag = True
        break

if flag: print('Yes')
else: print('No')
```

Παράδειγμα. Μια Πυθαγόρεια τριάδα είναι ένα σύνολο τριών φυσικών αριθμών $a < b < c$ τέτοιων ώστε $a^2 + b^2 = c^2$. Για παράδειγμα, το σύνολο $\{3, 4, 5\}$ είναι μια Πυθαγόρεια τριάδα γιατί $3^2 + 4^2 = 5^2$. Γράψτε ένα πρόγραμμα το οποίο θα βρίσκει τις Πυθαγόρειες τριάδες για $1 < a, b < 20$

```
import math

m = 20
counter = 0
for a in range(1,m+1):
    for b in range(a+1,m+1):
        c2 = a**2 + b**2
        c  = math.sqrt(c2)
        if int(c) == c:
           counter += 1
           c = int(c)
           print(a,b,c)
```

```
print("I found",counter,"Pythagorean triads")
```

Πιο εκτεταμένα προβλήματα

Εύρεση κυβικής ρίζας τέλειου κύβου με *εξαντλητική απαρίθμηση* (μία σημαντική βελτίωση παλαιότερου κώδικα).

```
perfect_cube = int(input("Give an integer: "))
x = 0

while x**3 < abs(perfect_cube):
    x = x + 1

if x**3 != perfect_cube:
    print(perfect_cube,"is not a perfect cube")
else:
    if perfect_cube < 0:
       x = -x
    print("The cubic root of", perfect_cube,"",x)
```

Μέθοδος διχοτόμησης [Δείτε J.V. Guttag (Παράγραφος 3.3)]. Για την αναζήτηση προσέγγισης τατραγωνικής ρίζας αριθμού. Έστω $x > 1$ τότε γνωρίζουμε ότι $1 < \sqrt{x} < x$. Θα αρχίσουμε περιορίζοντας τη λύση στο διάστημα $[0, x]$ και θα διχοτομούμε το διάστημα ώσπου να πετύχουμε μία συγκεκριμένη ακρίβεια της προσέγγισης (epsilon).

```
x = 2;              # θα βρούμε την τετραγωνική ρίζα του 2
epsilon = 1.0e-3    # η ακρίβεια του αποτελέσματος
numGuesses = 0      # μετρητής για τις επαναλήψεις που θα γίνουν

low = 1; high = x;       # το αποτέλεσμα θα είναι στο διάστημα [low,high]
mid = (low + high)/2.0

while abs(x - mid**2) >= epsilon:
    print('low =', low, 'high =', high, 'Approximation =', mid)
    numGuesses += 1
    if mid**2 < x:
```

```
            low = mid
        else:
            high = mid
        mid = (low + high)/2.0

    print('Number of iterations = ',numGuesses)
    print('The square root of',x,'is approximately', mid)
```

Κρεμάλα. Φτιάξτε ένα πρόγραμμα με το οποίο μπορούμε να παίξουμε το παιχνίδι Κρεμάλα.

- Κώδικας, σε μία αρχική μορφή με αρκετές ελλείψεις.
- Κώδικας, ο οποίος κάνει κάποιους ελέγχους και τυπώνει καλύτερα της κρεμάλα.
- Πλήρης κώδικας, στον οποίο όμως πρέπει να δίνετε εσείς τη ζητούμενη λέξη μέσα στο πρόγραμμα.
- Πλήρης κώδικας, ο οποίος διαβάζει μία τυχαία λέξη από αρχείο.

List comprehension (Υπολογιζόμενη λίστα).

Έχουμε την ακόλουθη μέθοδο δημιουργίας μιας λίστας.

```
comprehensionList = [εκφραση for μεταβλητη in ακολουθια]
```

```
>>> squares = [i**2 for i in range(11)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Μελέτη

Βιβλιογραφία

1. J.V. Guttag, *Υπολογισμοί και προγραμματισμός με την python*, (Κεφάλαιο 3).
2. Κ. Μαγκούτης, Χ. Νικολάου, *Εισαγωγή στον αντικειμενοστραφή προγραμματισμό με Python*, (Αποθετήριο "Κάλλιπος", 2016) - Κεφάλαιο 5. Επανάληψη με την εντολή while.
3. Style Guide for Python Code.

Προηγούμενο Επόμενο