ΜΕΜ104 Γλώσσα Προγραμματισμού Ι

Μιχάλης Πλεξουσάκης

19 Οκτωβρίου 2020

Μαθηματικά και Εφαρμοσμένα Μαθηματικά

Περιεχόμενα

1. Επανάληψη `while`
2. Η επανάληψη `for`

Επανάληψη `while`

Επανάληψη `while`

Ο μηχανισμός επανάληψης `while` της Python είναι:

`while` *συνθήκη*`:`
    εντολές

Για παράδειγμα, ο ακόλουθος κώδικας Python υπολογίζει το άθροισμα των αριθμών από το 1 ως το N:

```
N = 25
k, s = 1, 0

while k <= N:
    s = s + k
    k = k + 1

print('sum =', s)
```

Επανάληψη while

Μπορούμε να χρησιμοποιήσουμε την εντολή while για να εκτελέσουμε κάποιες εντολές έναν συγκεκριμένο αριθμό φορών, όπως για παράδειγμα, να διαβάσουμε δεδομένα:

```
i = 0   # counter
while i < 10:
    x = int(input('Enter an integer: '))
    if x <= 0:
        print('The number is non-positive')
    else:
        print('The number is positive')
    i = i + 1
print('Done!')
```

Επανάληψη while

Μπορούμε ακόμα να τερματίσουμε την επανάληψη πρόωρα, χρησιμοποιώντας την εντολή `break`, όπως στο παρακάτω παράδειγμα

```
s = 'This is a rather long string'
i = 0
while i < len(s):
    if s[i] == 'o':
        print('The letter o is in position', i)
        break
    i = i + 1
if i == len(s):
    print('There is no o in the word', s)
```

Επανάληψη `while`

Η εντολή `break` είναι χρήσιμη για να διαφύγουμε από τις λεγόμενες *ατέρμονες επαναλήψεις*, όπως αυτή που ακολουθεί:

```
while True:
    x = int(input('Pick a number between 1 and 10: '))
    if x >= 1 and x <= 10:
        break

print('You chose the number', x)
```

Επανάληψη while

Μπορούμε να παραλείψουμε, προσωρινά, τις εντολές της επανάληψης με την εντολή `continue`:

```
iter = 0
s = 0
while iter < 10:
    x = int(input('Enter a positive integer: '))
    if x <= 0:
        continue
    s = s + x
    iter = iter + 1

print('The sum is', s)
```

Πρόβλημα: Βρείτε τη μικρότερη δύναμη του 2 που είναι μεγαλύτερη ή ίση από έναν πραγματικό αριθμό x.

Για παράδειγμα, αν $x = 1000$, η μικρότερη δύναμη του 2 που είναι μεγαλύτερη ή ίση του x είναι η $2^{10} = 1024$.

```
x = float(input('Enter a real number: '))
p = 1
while p < x:
    p = p * 2
print('Smallest power of two >=', x, 'is', p)
```

Άσκηση: Βρείτε τη μεγαλύτερη δύναμη του 2 που είναι μικρότερη ή ίση από έναν ακέραιο $n \geq 1$.

Επανάληψη `while`

Σημαντικό: η εντολή ανάθεσης `x += y` είναι ισοδύναμη με την εντολή `x = x + y`.
Έτσι, μπορούμε να γράψουμε `x += 1` για να αυξήσουμε την τιμή της μεταβλητής `x` κατά μια μονάδα.
Ανάλογα ισχύουν για τις εντολές ανάθεσης

`x -= y`, `x *= y`, `x /= y`, `x //= y` και `x **= y`.

Προσοχή: αν η τιμή της μεταβλητής `x` είναι δύο και η τιμή της μεταβλητής `y` είναι τρία τότε, μετά την εντολή `x *= y + 1` η τιμή της μεταβλητής `x` θα είναι 8.

Επανάληψη while

Το πρόγραμμα Python που ακολουθεί, εξετάζει αν ο ακέραιος που δίνεται ως είσοδος είναι κύβος κάποιου αριθμού:

```
num = int( input('Enter an integer: ') )

ans = 0
while ans**3 < abs(num):
    ans += 1

if ans**3 != abs(num):
    print(num, 'is not a perfect cube')
else:
    if num < 0:
        ans = -ans
    print('Cube root of', num, 'is', ans)
```

Επανάληψη while

Το παρακάτω πρόγραμμα προσεγγίζει τον αριθμό $\sqrt{2}$ χρησιμοποιώντας τη μέθοδο της διχοτόμισης:

```
x = 2; eps = 1.0e-3

low = 1; high = 2;
mid = (low + high) / 2

while abs(x - mid**2) > eps:
    print('low =', low, 'high =', high, 'Approximation =', mid)
    if mid**2 < x:
        low = mid
    else:
        high = mid
    mid = (low + high) / 2

print('The square root of 2 is approximately', mid)
```

Επανάληψη `while`

Παρατήρηση: Αν a είναι θετικός πραγματικός αριθμός και x είναι μια προσέγγιση του αριθμού $\sqrt{a}$ τότε, ο αριθμός $\frac{1}{2}\left(x + \frac{a}{x}\right)$ είναι μια καλύτερη προσέγγισή του.

Αυτό το γεγόνός, το οποίο σήμερα μπορούμε εύκολα να δικαιολογήσουμε, ήταν γνωστό στον Ήρωνα της Αλεξάνδρειας (10-70 μ.Χ.) και πιθανώς πολύ νωρίτερα.

Για παράδειγμα, αν $a = 2$ και χρησιμοποιήσουμε $x = 1$ ως αρχική προσέγγιση του αριθμού $\sqrt{2}$, τότε έχουμε $y = 1.5$. Αν τώρα θέσουμε $x = 1.5$ υπολογίζουμε $y = \frac{17}{12} \approx 1.41666$, μια προσέγγιση με σφάλμα μόνο 0.0025.

Αν τώρα $x = \frac{17}{12}$ έχουμε $y = \frac{577}{408} = 1.414215686$, μια προσέγγιση του $\sqrt{2}$ με πέντε σωστά δεκαδικά ψηφία!

Επανάληψη `while`

Η μέθοδος για την προσέγγιση της τετραγωνικής ρίζας του a που προκύπτει είναι μια επαναληπτική διαδικασία η οποία τερματίζει όταν η προσέγγιση είναι, για τις ανάγκες μας, αρκετά καλή:

$x \leftarrow$ αρχική προσέγγιση του a
όσο το x δεν είναι αρκετά καλή προσέγγιση
   $y \leftarrow \frac{1}{2}\left(x + \frac{a}{x}\right)$
   $x \leftarrow y$

Η υλοποίηση σε Python είναι εύκολη με την εντολή επανάληψης `while`.

Επανάληψη `while`

Ο κώδικας παρακάτω υπολογίζει μια προσέγγιση x της $\sqrt{2}$ τέτοια ώστε $|x^2 - 2| < 10^{-5}$

```
a = 2
x = 1; k = 0  # First approximation and loop counter
while abs(x*x - a) > 1.0e-5:
    x = 0.5*(x + a/x)
    k += 1
print('Iterations =', k)
print('The square root of', a, 'is ~', x)
```

Αν εκτελέσουμε αυτό τον κώδικα παίρνουμε

```
Iterations = 3
The square root of 2 is ~ 1.4142156862745097
```

Η επανάληψη `for`

Η επανάληψη `for`

Για τη σάρωση μιας ακολουθίας ακεραίων, όπως σε όλα τα προηγούμενα παραδείγματα, η Python παρέχει τον μηχανισμό επανάληψης `for`, η γενική μορφή του οποίου είναι:

`for` *μεταβλητή* `in` *ακολουθία*`:`
    εντολές

Η *μεταβλητή* συνδέεται διαδοχικά με κάθε μία τιμή από την *ακολουθία* και εκτελείται το τμήμα κώδικα στο σώμα της `for`.

Η διαδικασία επαναλαμβάνεται μέχρι να εξαντληθούν τα στοιχεία της ακολουθίας, ή να εκτελεστεί μια εντολή `break` στο σώμα της `for`.

Η *ακολουθία* δημιουργείται, συνήθως, με τη συνάρτηση `range` η οποία επιστρέφει μια *αριθμητική πρόοδο*.

Η επανάληψη `for`

Η συνάρτηση `range` δέχεται τρία ακέραια ορίσματα: `start`, `stop` και `step` και παράγει την αριθμητική πρόοδο:

`start`, `start+step`, `start+2*step`, …

- Αν το `step` είναι θετικό, ο τελευταίος όρος της ακολουθίας είναι ο μεγαλύτερος ακέραιος `start+i*step` ο οποίος είναι μικρότερος από τον `stop`.
- Αν το `step` είναι αρνητικό, ο τελευταίος όρος της ακολουθίας είναι ο μικρότερος ακέραιος `start+i*step` ο οποίος είναι μεγαλύτερος από τον `stop`
- Αν παραληφθεί το `start` νοείται ως 0. Αν παραληφθεί το `step` νοείται ως 1.

Η επανάληψη `for`

Για παράδειγμα, η συνάρτηση `range(5, 20, 3)` παράγει την ακολουθία `5, 8, 11, 14, 17`.

Όμοια, η συνάρτηση `range(20, 5, -4)` παράγει την ακολουθία `20, 16, 12, 8`.

Πράγματι, εκτελώντας το πρόγραμμα

```
for i in range(5, 20, 3):
    print(i)
```

παίρνουμε

```
5
8
11
14
17
```

Η επανάληψη for

Το πρόβλημα της εύρεσης της κυβικής ρίζας θα μπορούσε να γραφεί ως

```
num = int( input('Enter an integer: ') )
for ans in range(0, abs(num)+1):
    if ans**3 >= abs(num):
        break
if ans**3 !=  abs(num):
    print(num, 'is not a perfect cube')
else:
    if num < 0:
        ans = -ans
    print('Cube root of', num, 'is', ans)
```

Η επανάληψη `for`

Η εντολή `for` μπορεί επίσης να χρησιμοποιηθεί για την εύκολη προσπέλαση των χαρακτήρων μιας συμβολοσειράς:

```
s = 'programming'
print('The vowels in the word', s, 'are:')
for c in s:
    if c in 'aeiou':
        print(c)
```

Θα μπορούσαμε ακόμα να γράψουμε:

```
for i in range(len(s)):
    if s[i] in 'aeiou':
        print(s[i])
```

αλλά, πραγματικά, δεν υπάρχει <u>κανένας απολύτως λόγος</u> να χρησιμοποιήσουμε τον δείκτη θέσης...