Σημειωματάαριο Δευτέρας 16 Οκτ. 2017

Λίστες και ανακύκλωση for

Είδαμε στην αρχή (ξανά) μερικά βασικά πράγματα για λίστες. Λίστα είναι μια πεπερασμένη ακολουθία από αντικείμενα (αριθμούς, strings, άλλες λίστες, οτιδήποτε) που γράφονται μέσα σε αγκύλες και χωρίζονται με κόμμα. Τα στοιχεία της λίστας αριθμούνται από το 0 έως το μήκος της πλην 1 (`len(L)` είναι το μήκος της λίστας L).

Δείτε παρακάτω πώς αναφερόμαστε σε στοιχεία της λίστας.

Στο τελευταίο παράδειγμα `L[3][1]` είναι το στοιχείο υπ' αριθμόν 1 (δεύτερο δηλ.) της λίστας `L[3]`, της λίστας δηλ. που βρίσκεται στην υπ' αριθμόν 3 θέση (τέταρτη) της λίστας L.

In [4]:

```
L = [1, 2, "abc", [True, 3], "abcdef"]
print(L)
print(L[0])
print(L[3])
print(L[len(L)-1])
print(L[3][1])
```

```
[1, 2, 'abc', [True, 3], 'abcdef']
1
[True, 3]
abcdef
3
```

Τα slices δουλεύουν όπως ακριβώς και στα strings.

In [7]:

```
L = [1, 2, "abc", [True, 3], "abcdef"]
print( L[1:] )
print( L[1:len(L)-1])
print( L[1:-1] )
```

```
[2, 'abc', [True, 3], 'abcdef']
[2, 'abc', [True, 3]]
[2, 'abc', [True, 3]]
```

Μπορεί κανείς να συγκολλήσει δύο λίστες με την πράξη +.

Αν κάποιος θέλει απλά να προσθέσει ένα στοιχείο στο τέλος μιας λίστας μπορεί να χρησιμοποιήσει και τη μέθοδο `.append()` που έχουν οι λίστες.

In [10]:

```
L=[1, 2, 3]; M=[True, False]
X = L+M
print(X)
X = X + [ "abc" ]
print(X)
X.append(0.33)
print(X)
```

```
[1, 2, 3, True, False]
[1, 2, 3, True, False, 'abc']
[1, 2, 3, True, False, 'abc', 0.33]
```

Η βασική μορφή της ανακύκλωσης `for` φαίνεται παρακάτω. Έχουμε μια μεταβλητή, x στη συγκεκριμένη περίπτωση, και μια λίστα (την L εδώ) και το `for` μας κάνει τη μεταβλητή να πάρει μια προς μια όλες τις τιμές της λίστας, αρχίζοντας από την πρώτη και τελειώνοντας με την τελευταία.

Στο παράδειγμα παρακάτω τυπώνουμε τα τετράγωνα όλων των αριθμών που βρίσκονται στη λίστα L.

In [1]:

```
L = [2.1, 3.2, 4, -7]
for x in L:
    print("Το τετράγωνο του {} είναι το {}".format(x, x*x))
```

```
Το τετράγωνο του 2.1 είναι το 4.41
Το τετράγωνο του 3.2 είναι το 10.240000000000002
Το τετράγωνο του 4 είναι το 16
Το τετράγωνο του -7 είναι το 49
```

Εδώ ξεκινάμε από την L και υπολογίζουμε μια νέα λίστα, την S, η οποία θα περιέχει τα τετράγωνα όλων των αριθμών της L. Πριν το `for` loop θέτουμε τη λίστα S να είναι κενή και για κάθε στοιχείο x της L κάνουμε append στο τέλος της S το τετράγωνο του x.

In [2]:

```
L = [2.1, 3.2, 4, -7]
S = []
for x in L:
    S.append(x*x)
print(S)
```

```
[4.41, 10.240000000000002, 16, 49]
```

Η συνάρτηση `range(N)` επιστρέφει ουσιαστικά μια λίστα που ξεκινάει από το 0 και πάει μέχρι το N-1. Είναι ο πιο βολικός τρόπος να κάνουμε μια ανακύκλωση ένα συγκεκριμένο αριθμό φορών. Στο παράδειγμα που ακολουθεί τυπώνουμε τα τετράγωνα όλων των αριθμών από 0 έως 10.

In [14]:

```
for i in range(11):
    print(i*i)
```

```
0
1
4
9
16
25
36
49
64
81
100
```

Εδώ βλέπουμε πώς μπορούμε να τυπώσουμε κάτι με τη συνάρτηση `print` χωρίς να αλλάξει αυτόματα γραμμή. Αυτό γίνεται προσθέτοντας την (ονομασμένη) παράμετρο `end=''` στο τέλος.

In [3]:

```
print(1, end=''); print(2) # Με το σύμβολο ; μπορούμε να διαχωρίσουμε δύο εντολές για να τις δώσουμε στην ίδια γραμμή
```

```
12
```

Εδώ χρησιμοποιούμε μια ανακύκλωση `for` μέσα σε μια άλλη ανακύκλωση `for` για να τυπώσουμε όλα τα ζεύγη (x, y) με x και y να παίρνουν όλες τις τιμές από 1 έως 10 το καθένα. Αλλάζουμε γραμμή μόνο αφού τελειώσει η εσωτερική ανακύκλωση (για το y).

In [22]:

```
for x in range(1, 11): # το x παίρνει όλες τις τιμές από 1 έως 10
    for y in range(1, 11): # το y παίρνει όλες τις τιμές από 1 έως 10. Αυτό συμβαίνει για κάθε τιμή του x.
        print( "({},{}) ".format(x, y), end='')
    print() # Αυτό είναι για να αλλάξουμε γραμμή εκτελείται μια φορά για κάθε τιμή του x
```

```
(1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (1,8) (1,9) (1,10)
(2,1) (2,2) (2,3) (2,4) (2,5) (2,6) (2,7) (2,8) (2,9) (2,10)
(3,1) (3,2) (3,3) (3,4) (3,5) (3,6) (3,7) (3,8) (3,9) (3,10)
(4,1) (4,2) (4,3) (4,4) (4,5) (4,6) (4,7) (4,8) (4,9) (4,10)
(5,1) (5,2) (5,3) (5,4) (5,5) (5,6) (5,7) (5,8) (5,9) (5,10)
(6,1) (6,2) (6,3) (6,4) (6,5) (6,6) (6,7) (6,8) (6,9) (6,10)
(7,1) (7,2) (7,3) (7,4) (7,5) (7,6) (7,7) (7,8) (7,9) (7,10)
(8,1) (8,2) (8,3) (8,4) (8,5) (8,6) (8,7) (8,8) (8,9) (8,10)
(9,1) (9,2) (9,3) (9,4) (9,5) (9,6) (9,7) (9,8) (9,9) (9,10)
(10,1) (10,2) (10,3) (10,4) (10,5) (10,6) (10,7) (10,8) (10,9) (10,1
0)
```

Εδώ δείχνουμε πώς να αποφασίσουμε αν ένας φυσικός αριθμός N (που τον δίνει ο χρήστης) είναι πρώτος ή όχι (ένας φυσικός αριθμός λέγεται πρώτος αριθμός αν οι μόνοι φυσικοί αριθμοί που τον διαιρούν είναι ο 1 και ο εαυτός του).

In [4]:

```
N = int(input("Δώστε ένα φυσικό αριθμό > 1: ")) # Ο χρήστης δίνει ένα string που μετατρέπεται σε ακέραιο με τη
                                                # συνάρτηση int και αποθηκεύεται στη μεταβλητή N
isprime = True # Η λογική αυτή μεταβλητή θα είναι True στο τέλος αν και μόνο αν ο N είναι πρώτος.
               # Κατ' αρχήν τη θέτουμε True, μια και δεν έχουμε ακόμη ανακαλύψει κάποιο διαιρέτη του N
for k in range(2, N): # Με το for αυτό διανύουμε όλους του φυσικούς αριθμούς από 2 έως N-1
                      # και δοκιμάζουμε αν είναι διαιρέτες του N.
    if N % k == 0: # Διαιρεί ο k τον N;
        isprime = False # Αν ναι, θέτουμε isprime = False
if isprime: # Για να συμβεί αυτό πρέπει το if μέσα στο loop να μην έχει "πιάσει" ποτέ
    print("Ο {} είναι πρώτος".format(N))
else:
    print("Ο {} είναι σύνθετος".format(N))
```

```
Δώστε ένα φυσικό αριθμό > 1: 23864
Ο 23864 είναι σύνθετος
```

Αν βρούμε ένα διαιρέτη k του N δε χρειάζεται να ψάξουμε και τους μετέπειτα φυσικούς για να αποφανθούμε αν ο N είναι πρώτος. Ξέρουμε ήδη ότι δεν είναι. Με την εντολή `break` που εκτελείται όταν βρούμε διαιρέτη "σπάμε" το for. Όποτε εκτελεστεί το `break` μέσα σε μια ανακύκλωση αυτή παύει να εκετελείται και η ροή του προγράμματος μεταφέρεται μετά την ανακύκλωση αυτή.

In [30]:

```
N = int(input("Δώστε ένα φυσικό αριθμό > 1: "))
isprime = True
for k in range(2, N):
    if N % k == 0:
        isprime = False
        break # Σταματάει το for.
if isprime:
    print("Ο {} είναι πρώτος".format(N))
else:
    print("Ο {} είναι σύνθετος".format(N))
```

```
Δώστε ένα φυσικό αριθμό > 1: 2734572354
Ο 2734572354 είναι σύνθετος
```

Αν ένας φυσικός k διαιρεί τον N τότε έχουμε $d = N/k$ να είναι ακέραιος. Επίσης, αν $k > \sqrt{N}$ και ο k διαιρεί τον N έπεται ότι $d = N/k < \sqrt{N}$. Άρα αν έχουμε ελέγξει όλα τα k που είναι $\sqrt{N}$ και δεν έχουμε βρεί κανένα διαιρέτη του N έπεται ότι δε θα βρούμε ούτε από κει και πέρα. Άρα σε αυτή την περίπτωση ο N είναι πρώτος.

In [31]:

```
N = int(input("Δώστε ένα φυσικό αριθμό > 1: "))
isprime = True
for k in range(2, N):
    if k*k > N: # Είναι το k μεγαλύτερο του ρίζα N;
        break # Σπάμε την ανακύκλωση αν έχουμε ελέγξει όλα τα k μικρότερα ή ίσα
 από ρίζα N.
    if N % k == 0:
        isprime = False
        break
if isprime:
    print("Ο {} είναι πρώτος".format(N))
else:
    print("Ο {} είναι σύνθετος".format(N))
```

```
Δώστε ένα φυσικό αριθμό > 1: 1562343
Ο 1562343 είναι σύνθετος
```

Αν το N είναι σύνθετο θέλουμε να βρούμε όλους τους διαιρέτες του (τους μη τετριμμένους, αυτούς δηλ. διάφορους του 1 και του N).

Ανοίγουμε μια λίστα `divisors` μέσα στην οποία βάζουμε κάθε k που βρίσκουμε ότι διαιρεί το N. Η λίστα αυτή είανι αρχικά κενή. Δεν σταματάμε το ψάξιμο πριν το $\sqrt{N}$ γιατί τώρα δε μας ενδιαφέρει απλά να πούμε ότι κάτι είναι σύνθετο αλλά και να βρούμε **όλους** τους διαιρέτες του.

In [5]:

```
N = int(input("Δώστε ένα φυσικό αριθμό > 1: "))
isprime = True
divisors = []
for k in range(2, N):
#    if k*k > N:
#        break
    if N % k == 0:
        isprime = False
        divisors.append(k) # Το k είναι διαιρέτης, άρα μπαίνει στη λίστα divisor
s
if isprime:
    print("Ο {} είναι πρώτος".format(N))
else:
    print("Ο {} είναι σύνθετος".format(N))
    print("Οι διαιρέτες του {} είναι οι: ".format(N), end='')
    for d in divisors: # Εδώ τυπώνουμε όλους τους διαιρέτες του N.
        print("{} ".format(d), end='')
```

```
Δώστε ένα φυσικό αριθμό > 1: 213432
Ο 213432 είναι σύνθετος
Οι διαιρέτες του 213432 είναι οι: 2 3 4 6 8 12 24 8893 17786 26679 3
5572 53358 71144 106716
```

Πώς μπορούμε ξανά να εξοικονομήσουμε χρόνο από το ψάξιμο εκμεταλλευόμενοι την ιδιότητα του $\sqrt{N}$ όπως πριν;

Παρατηρούμε ότι αν βρούμε ένα διαιρέτη k του N μικρότερο ή ίσο από $\sqrt{N}$ τότε ο διαιρέτης N/k είναι μεγαλύτερος ή ίσος από $\sqrt{N}$. Όλοι οι διαιρέτες του N δηλ. είναι ζευγαρωμένοι σε ζεύγη της μορφής (k, d) με $N = kd$ και $k \leq \sqrt{N}$ και $d \geq \sqrt{N}$.

Φτιάχνουμε λοιπόν τη λίστα των k στη λίστα `divisors1` (οι διαιρέτες $\leq \sqrt{N}$) και τη λίστα των d στη λίστα `divisors2` (οι διαιρέτες $> \sqrt{N}$).

Όταν βάλουμε ένα αριθμό k στη λίστα `divisors1` βάζουμε και τον αριθμό $d = N/k$ στη λίστα `divisors2`, εκτός αν $k = d$ (που συμβαίνει μόνο όταν $N = k^2 = d^2$.

Έτσι όμως η λίστα `divisors2` βγαίνει σε φθίνουσα σειρά. Πριν την τυπώσουμε λοιπόν τη διανύουμε ανάποδα. Η λίστα `divisors2[::-1]` είναι η λίστα των στοιχείων της `divisors2` αλλά με την ανάποδη σειρά. Αυτό επιτυγχάνεται με το slice `::-1` μας λέει να διανύσουμε τη λίστα με βήμα -1, ανάποδα δηλ.

In [6]:

```
N = int(input("Δώστε ένα φυσικό αριθμό > 1: "))
isprime = True
divisors1 = [] # Εδώ μπαίνουν οι διαιρέτες μικρότεροι ή ίσοι του ρίζα N
divisors2 = [] # Εδώ μπαίνουν οι διαιρέτες μεγαλύτεροι του ρίζα N
for k in range(2, N):
    if k*k > N:
        break
    if N % k == 0:
        isprime = False
        divisors1.append(k) # Βρήκαμε τον διαιρέτη k που είναι μικρότερος ή ίσος
 του ρίζα N
        if k*k != N: # Αν το k δεν είναι ακριβώς ρίζα N
            divisors2.append(N//k) # βάζουμε το N/k στην άλλη λίστα
if isprime:
    print("Ο {} είναι πρώτος".format(N))
else:
    print("Ο {} είναι σύνθετος".format(N))
    print("Οι διαιρέτες του {} είναι οι: ".format(N), end='')
    for d in divisors1+divisors2[::-1]: # Πριν τυπώσουμε τους διαιρέτες γυρνάμε
 ανάποδα τη λίστα divisors2
        print("{} ".format(d), end='')
```

```
Δώστε ένα φυσικό αριθμό > 1: 123431
Ο 123431 είναι σύνθετος
Οι διαιρέτες του 123431 είναι οι: 7 11 49 77 229 539 1603 2519 11221
17633
```