Τμήμα Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Μάθημα: **Τεχνικές Προγραμματισμού Υπολογιστών. (Εργαστηριακό μάθημα)**
Καθηγητής : Πεφάνης Ευάγγελος

3) Εργαστηριακές σημειώσεις στην γλώσσα προγραμματισμού Python.

Πλειάδες (tuples)

Ομαδοποίηση στοιχείων

Μία σειρά στοιχείων (αριθμών, συμβολοσειρών κλπ) μπορούν να ομαδοποιηθούν κάτω από ένα όνομα σε μία δομή η οποία ονομάζεται *πλειάδα*. Παραδείγματα:

```
>>> info = ('Nick',22)
>>> cities = ('Agios','Heraklion','Rethymno','Chania')
```

Μπορούμε να δώσουμε πολλές τιμές σε μία μεταβλητή, οπότε αυτή είναι μία πλειάδα:

```
>>> t1 = 1,2,'three'
>>> print(t1)
(1, 2, 'three')
```

Δείκτες: Μπορούμε να ανακτήσουμε ένα από τα στοιχεία της πλειάδας χρησιμοποιώντας το δείκτη του. Η δεικτοδότηση είναι ακριβώς όπως στις συμβολοσειρές.

```
>>> t1[1]
2
>>> t1[-1]
'three'
```

slicing: Μπορούμε να ανακτήσουμε ένα τμήμα της πλειάδας χρησιμοποιώντας ένα εύρος δεικτών. Το αποτέλεσμα είναι μία νέα πλειάδα.

```
>>> t1[0:2]
(1, 2)
```

Βασικές λειτουργίες

Μπορούμε να χρησιμοποιήσουμε τα σύμβολα της πρόσθεσης (+) για πλειάδες:

```
>>> t1 = (1,2,'three')
>>> t2 = ('a','b')
>>> t1 + t2
(1, 2, 'three', 'a', 'b')
```

καθώς επίσης και του πολλαπλασιασμού (*):

```
>>> 2*t1
(1, 2, 'three', 1, 2, 'three')
```

Μία πλειάδα μπορεί να περιέχει πλειάδες.

```
>>> t  = (t1,t2)
>>> print(t
((1, 2, 'three'), ()'a', 'b'))
```

Ειδικές περιπτώσεις πλειάδων

Έχουμε την κενή πλειάδα.

```
>>> t = ()
```

Έχουμε την πλειάδα με ένα στοιχείο (προσέξτε το κόμμα!).

```
>>> t = (1,)
```

Παράδειγμα. Βρείτε τους διαιρέτες ενός ακεραίου.

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

x = 20
result = findDivisors(x)
print(result)
```

Μη-μεταλλαξιμότητα

Δεν μπορούμε να αλλάξουμε ένα (ή ορισμένα) από τα στοιχεία μίας πλειάδας (οι πλειάδες δεν είναι μεταλλάξιμες).

Πολλαπλές αναθέσεις τιμών

Μπορούμε να μεταφέρουμε τις τιμές μίας πλειάδας μήκους *n* σε ισάριθμες μεταβλητές.

```
```

```
>>> t = (1, 4, 8)
>>> x,y,z = t
>>> print(x)
>>> 1
>>> print(x,y,z)
1 4 8
```

Μπορούμε να δώσουμε τιμές σε σειρά μεταβλητών.

```
>>> x,y,z = (1, 4, 8)
>>> print(x)
>>> 1
>>> print(x,y,z)
1 4 8
```

Τα παραπάνω είναι πολύ χρήσιμα όταν μία συνάρτηση επιστρέφει σειρά μεταβλητών (αποτελεσμάτων).

Παράδειγμα. Σώμα μάζας m εκτοξεύεται προς τα επάνω με ταχύτητα $v0$. Βρείτε την ταχύτητα και θέση του για διαδοχικούς χρόνους.

```
import math

def fallingParticle(t):
    global v0,g
    velocity = v0 - g*t
    position = v0*t - 0.5*g*t**2
    return velocity,position

global v0,g
v0 = 20.0
g = 10.0
time = 0.0
while time < 2.0:
    v,y = fallingParticle(time)
    print(time,v,y)
    time += 0.2
```

Συναρτήσεις με εφαρμογή σε πλειάδες

```
t1 = (1,3,6,8)
```

Μπορούμε να πάρουμε:

- Τον αριθμό στοιχείων πλειάδας: `len(t1)`.
- Το μέγιστο και ελάχιστο στοιχείο πλειάδας (όταν αυτά είναι συγκρίσιμα μεταξύ τους): `max(t1)`, `min(t1)`.

Παράδειγμα. Κατασκευάστε μία πλειάδα η οποία θα περιέχει την ακολουθία αριθμών Fibonacci $Fi,i=1,…,nFi,i=1,…,n$ για έναν θετικό ακέραιο nn (την οποία θα εισάγετε όταν εκτελείται το πρόγραμμα).

Λίστες

Σειρά στοιχείων

Μία σειρά στοιχείων (αριθμών, συμβολοσειρών κλπ) μπορούν να οργανωθούν σε μία δομή η οποία ονομάζεται *λίστα*. Παραδείγματα:

```
>>> rgb = ['red','green','blue']
>>> cities = ['Agios','Heraklion','Rethymno','Chania']
>>> digits = [0,1,2,3,4,5,6,7,8,9]
```

Το μήκος μίας λίστας μπορεί να βρεθεί με χρήση της συνάρτησης `len`:

```
>>> len(digits)
10
```

Μπορούμε να ανακτήσουμε ένα από τα στοιχεία της λίστας χρησιμοποιώντας τον δείκτη του. Η δεικτοδότηση είναι ακριβώς όπως στις συμβολοσειρές.

```
>>> rgb[1]
'green'
>>> rgb[-1]
'blue'
```

Παρατήρηση. Σημαντική διαφορά των λιστών από τις πλειάδες είναι ότι οι πρώτες είναι *μεταλλάξιμες (mutable)*.

slicing: Μπορούμε να ανακτήσουμε ένα τμήμα της λίστας χρησιμοποιώντας ένα εύρος δεικτών. Το αποτέλεσμα είναι μία νέα λίστα.

```
>>> digits[1:3]
[1,2]
>>> digits[7:]
[7,8,9]
```

Βασικές λειτουργίες

Μπορούμε να χρησιμοποιήσουμε τα σύμβολα της πρόσθεσης (+) για λίστες:

```
>>> cmyk = ['cyan','magenta','yellow','black']
>>> colors = rgb + cmyk
>>> colors
['red','green','blue','cyan','magenta','yellow','black']
```

καθώς επίσης και του πολλαπλασιασμού (*):

```
>>> 2*rbg
['red','green','blue','red','green','blue']
>>> 2*[0,1]
[0,1,0,1]
```

Μέθοδοι

Σε μία υπάρχουσα λίστα μπορούμε να προσθέσουμε ένα στοιχείο:

```
>>> colours.append('white')
>>> colors
['red','green','blue','cyan','magenta','yellow','black','white']
```

append `[λιστα.append(στοιχειο)]`: Έχουμε χρησιμοποιήσει μία *μέθοδο* η οποία ονομάζεται `append` και επέδρασε στη λίστα `colors`. Η σύνταξη είναι `λιστα.μεθοδος(ορισμα)`. code>ορισμα προστίθεται ως τελευταίο στοιχείο στη `λιστα` και η υπάρχουσα λίστα μεγαλώνει κατά ένα στοιχείο.

λιστα.remove(στοιχειο): Αφαιρεί το `στοιχειο` από λίστα.

```
Π.χ., colors.remove('white')
```

λιστα.pop(): Αφαιρεί το τελευταίο στοιχείο λίστας (και το επιστρέφει). Επίσης μπορούμε να αφαιρέσουμε στοιχείο με συγκεκριμένο δείκτη.

```
>>> colors.pop()
'white'
>>> colors
['red','green','blue','cyan','magenta','yellow', 'black']]

>>> colors.pop(1)
'green'
>>> colors
['red','blue','cyan','magenta','yellow', 'black']
```

Μπορούμε να ελέγξουμε αν ένα στοιχείο υπάρχει σε λίστα:

```
>>> 'white' in colors
False

>>> if 'white' not in colors: colors.append('white')
```

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)
```

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]
```

Άλλες μέθοδοι

λιστα.index(στοιχειο). Επιστρέφει το δείκτη της πρώτης παρουσίας του *στοιχειο* στη *λιστα*.

λιστα.count(στοιχειο). Επιστρέφει τον αριθμό φορών που εμφανίζεται το *στοιχειο* στη *λιστα*.

λιστα.reverse(). Αναστρέφει τη σειρά των στοιχείων στη *λιστα*.

λιστα.sort(). Ταξινομεί τα στοιχεία της *λιστα* σε αύξουσα σειρά.

λιστα.insert(i,στοιχειο). Το στοιχείο τοποθετείται στη θέση i της λιστα.

Παράδειγμα. Ας δημιουργήσουμε μία λίστα, μετά θα αντιστρέψουμε τη σειρά των στοιχείων της.

```
>>> L = ['a','b','c','d','e','f']
>>> L.reverse()
>>> L
['f', 'e', 'd', 'c', 'b', 'a']
```

Παρατήρηση. Δείτε ότι η εντολή `L.reverse` μεταλλάσσει την ίδια τη λίστα (και δεν δημιουργεί μία καινούρια για να βάλει το αποτέλεσμα της μεθόδου).

Παράδειγμα. Ας δημιουργήσουμε μία λίστα, μετά θα ανακατέψουμε τα στοιχεία της με τυχαίο τρόπο και τέλος θα εντοπίσουμε ένα συγκεκριμένο στοιχείο της.

```
import random
L = ['a','b','c','d','e','f']
random.shuffle(L)
print(L.index('a'))
```

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

```
def sumList(L):
    summ = 0
    for num in L:
        summ += num
    return summ

L = [1,2,3,4,5,6,8,9]
print(sumList(L))
```

Παράδειγμα. Ας ελέγξουμε αν μία λίστα είναι παλινδρομική (δηλαδή, αν η σειρά των στοιχείων της είναι η ίδια είτε διαβάζεται από αριστερά προς τα δεξιά είτε αντίστροφα).

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

print(flag)
```

Μία πιο συμπαγής συνάρτηση, η οποία χρησιμοποιεί αναδρομική κλήση, είναι η ακόλουθη.

```
def isPalindrome(L):
    if len(L) <= 1:
        return True
    return L[0] == L[-1] and isPalindrome(L[1:-1])
```

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

Ειδικότερα είδη λιστών

Παρατήρηση. Μία λίστα μπορεί να περιέχει στοιχεία διαφορετικών τύπων:

```
>>> exampleList = ['father',30,'son',3]
```

Παρατήρηση. Μία λίστα μπορεί να περιέχει λίστες:

```
>>> exampleList = [[1,2,3],['a','b','c']]
```

Παράδειγμα.

Ας βάλουμε σε μία λίστα τις άρτιες μεταθέσεις τριών αριθμών.

```
>>> permute = []
>>> permute.append([1,2,3])
>>> permute.append([2,3,1])
>>> permute.append([3,1,2])
>>> permute
[[1, 2, 3], [2, 3, 1], [3, 1, 2]]
```

Παράδειγμα.

Θα δούμε την πρόσβαση στα στοιχεία μίας λίστας `L1` η οποία είναι στοιχείο άλλης λίστας `L`. Αυτό μπορεί να γίνει με δύο τρόπους: είτε μέσω της αρχικής λίστας `L1` είτε μέσω του αντίστοιχου στοιχείου της μεγαλύτερης λίστας `L`.

```
>>> L1 = ['a','b','c']
>>> L2 = [1,2,3]
>>> L = [L1,L2]
>>> L
[['a', 'b', 'c'], [1, 2, 3]]
L1.append('d')
>>> L
[['a', 'b', 'c', 'd'], [1, 2, 3]]
>>> L[0].append('e')
>>> L
[['a', 'b', 'c', 'd', 'e'], [1, 2, 3]]
>>> L1
['a', 'b', 'c', 'd', 'e']
```

Έλγχος τύπου μεταβητής (`isinstance()`)

Μπορούμε να ελέγξουμε αν μία μεταβλητή `a` είναι τύπου `list` με την εντολή `isinstance(a,list)` η οποία θα δώσει αποτέλεσμα `True/False`.

Παράδειγμα. [Πηγή: Σημειώσεις Μ. Πλεξουσάκη] Έστω L μια λίστα τα στοιχεία της οποίας μπορεί να είναι με τη σειρά τους λίστες. Για παράδειγμα, θα μπορούσαμε να έχουμε `L = [1, 2, ['one', 3], [[4]], 'Maria']`. Γράψτε μια συνάρτηση η οποία κατασκευάζει μια «μονοδιάστατη» λίστα, αφαιρεί δηλαδή τυχόν εσωτερικές λίστες.

```
def flatten(L):
```

```
    M = []
    for item in L:
        if isinstance(item, list):
            M.extend(flatten(item))
        else:
            M.append(item)
    return M
```

Ακολουθίες

Ερώτηση. Δώστε παραδείγματα ακολουθιών τα οποία έχουμε δει μέχρι τώρα. [Απάντηση: συμβολοσειρές, πλειάδες, λίστες.]

Ας θεωρήσουμε ότι στο όνομα μεταβλητής `seq` έχουμε είτε μία συμβολοσειρά, είτε μία πλειάδα, είτε μία λίστα. Τότε μπορούμε να χρησιμοποιήσουμε τις ακόλουθες εντολές.

- `seq[i]`: i-οστό στοιχείο ακολουθίας.
- `len(seq)`: μήκος ακολουθίας.
- `seq1 + seq 2`: συνένωση ακολουθιών.
- `n*seq`: επαναλαμβανόμενη ακολουθία.
- `seq[αρχη:τελος:βημα]`: τεμαχισμός ακολουθίας.
- `στοιχειο in seq`: boolean.
- `στοιχειο not in seq`: boolean.
- `for μεταβλητη in seq`: διατρέχει τα στοιχεία ακολουθίας.

Επίσης, έχουμε τις ακόλουθες συναρτήσεις για ακολουθίες

- `len(ακολουθια)`: ο αριθμός των στοιχείων ακολουθίας.
- `min(ακολουθια)`: το ελάχιστο στοιχείο ακολουθίας.
- `max(ακολουθια)`: το μέγιστο στοιχείο ακολουθίας.

Σε μία ακολουθία μπορούμε να εξασκήσουμε ορισμένες λειτουργίες:

- `ακολουθια.index(στοιχειο)`: η θέση της πρώτης εμφάνισης του στοιχείου στην ακολουθία.
- `ακολουθια.index.count(στοιχειο)`: αριθμός εμφανίσεων του στοιχείου στην ακολουθία.

Συμβολοσειρές και μέθοδοι Αν s είναι μία συμβολοσειρά, έχουμε τις μεθόδους:

- `s.index(s1)`: θέση του s1 στην s.
- `s.lower(), s.upper()`: μετατρέπει όλους τους χαρακτήρες σε πεζούς ή κεφαλαίους.
- `s.replace(old,new)`:
- `s.rstrip()`: αφαιρεί τους κενούς χαρακτήρες από το τέλος της s.
- `s.split(d)`: χωρίζει την s όπου υπάρχει το d και παράγει μία λίστα με το αποτέλεσμα.

Παράδειγμα. Ας διαβάσουμε σειρά αριθμών από το πληκτρολόγιο.

```
>>> line = input("Give numbers separated by comma: ")
Give numbers separated by comma: 1.2,  2.3,  3.14, 6
>>> strList = line.split(',')
>>> strList
['1.2', '  2.3', '  3.14', ' 6']
```

```
>>> numList = []
>>> for e in strList:
        numList.append(float(e))

>>> numList
[1.2, 2.3, 3.14, 6.0]
```

επίσης

```
numList = [float(e) for e in strList]
```

επίσης

```
numList = map(float,strList)
list(numList)
```

Ερώτηση. Πώς θα τροποποιούσατε το παραπάνω πρόγραμμα ώστε να διαβάζει μία σειρά αριθμών οι οποίοι διαχωρίζονται μόνο με κενά μεταξύ τους;

Δείτε επίσης και τις μεθόδους:

- `s.capitalize()`
- `s.center(πλατος),` `s.ljust(πλατος[,χαρακτήρας]),` `s.rjust(πλατος[,χαρακτήρας])`
- `s.endwith(σειμβολοσειρα), s.startwith(συμβολοσειρα)`
- `s.find(σειμβολοσειρα), s.rfind(σειμβολοσειρα)`
- `s.join(ακολουθία)`
- `s.partition([χαρακτήρας])`

Επίσης, τις

- `s.isalpha()`
- `s.isdigit()`
- `s.isnumeric()`

Μορφοποίηση εξόδου

Η εκτύπωση αποτελεσμάτων μπορεί να γίνει σε διαφορετικές μορφές:

```
for i in range(10):
        print('{:5d}'.format(i))
```

Εντός της εντολής `print` έχουμε διαμορφώσει μία συμβολοσειρά η οποία έχει περιέχει τους ακεραίους *i*. Στην παραπάνω εντολή ο κάθε ακέραιος βρίσκεται μέσα σε ένα πεδίο πλάτους 5 χαρακτήρων.

Παράδειγμα. Το παρακάρω πρόγραμμα τυπώνει δύο ακεραίους σε κάθε γραμμή σε πεδία πλάτους 5 και 7 χαρακτήρων αντίστοιχα. Μεταξύ τους αφήνονται δύο κενοί χαρακτήρες.

```
for i in range(10):
        print('{:5d}  {:7d}'.format(i,i**2))
```

Δείτε επίσης το αποτέλεσμα του προγράμματος:

```
for i in range(10):
        print('{:5d} - {:7d}'.format(i,i**2))
```

Για αριθμούς κινητής υποδιαστολής έχουμε τους κωδικούς μορφοποίησης *f* και *e* (για τον επιστημονικό συμβολισμό). Για παράδειγμα `{:8.5f}` δίνει πλάτος 8 με 5 δεκαδικά ψηφία.

Πέρασμα λίστας σε συνάρτηση ως όρισμα.(παράμετρο)

```
List1 = [1,4,7,8,9,14,33]

def sum_list (length,list) :

    sum_list = 0

    for ct in range (0,length):

        sum_list = sum_list + list[ct]

    return sum_list

print (sum_list(2, list1))
```

Αρχεία

Τα δεδομένα τα οποία χρειάζεται ένα πρόγραμμα μπορούν να διαβαστούν από ένα αρχείο. Επίσης τα αποτελέσματα τα οποία παράγονται μπορούν να καταγραφούν (να εκτυπωθούν) σε αρχείο. Ένα αρχείο μπορεί να συνδεθεί με το πρόγραμμά μας μέσω της εντολής `open`.

```
f = open('filename.dat','w')
```

Η παραπάνω εντολή δημιουργεί αρχείο με όνομα `filename.dat` και το συνδέει με το πρόγραμμά μας με σκοπό να γραφούν ('w' - write) στο αρχείο νέα αποτελέσματα. Η σύνδεση του αρχείου γίνεται μέσω ενός νέου αντικειμένου της python στο οποίο δώσαμε το όνομα *f*. Από εδώ και κάτω μπορούμε να επεμβαίνουμε στο αρχείο (με το όνομα `filename.dat`) χρησιμοποιώντας τη μεταβλητή *f*.

Παράδειγμα. Γράψτε μία σειρά ακεραίων καθώς και το τετράγωνό τους και την τετραγωνική τους ρίζα σε ένα αρχείο.

```
import math
f = open('test.dat','w')
```

```
for i in range(10):
    s = '{:4d}  {:5d}   {:5.3f}\n'.format(i,i**2,math.sqrt(i))
    f.write(s)
f.close()
```

Ακολουθήσαμε τα εξής βήματα:

- Συνδέουμε το αρχείο (`open`)
- Δημιουργούμε μία συμβολοσειρά *s* η οποία περιέχει τη μορφή την οποία θέλουμε να εκτυπώσουμε.
- Η μέθοδος `write` γράφει στο αρχείο τη συμβολοσειρά *s*.
- Η μέθοδος `close` τερματίζει τη σύνδεση του αρχείου με το πρόγραμμα (επίσης φέρνει το αρχείο στην τελική του μορφή).

Θα δούμε πώς μπορούμε να γράψουμε δεδομένα (αποτελέσματα) σε ένα αρχείο.

Παράδειγμα. (μέθοδος `read`) Διαβάστε το περιεχόμενο ενός αρχείου και εκτυπώστε το στην οθόνη.

```
f = open('test.dat','r')
s = f.read()
f.close()
print(s)
```

Ακολουθήσαμε τα εξής βήματα:

- Συνδέουμε το αρχείο (`open`) δηλώνοντας ότι θα διαβάσουμε από αυτό (`'r'` - read).
- Διαβάζουμε όλο το αρχείο (`f.read()`) και το περιεχόμενο πηγαίνει στη μεταβλητή *s*
- Η μέθοδος `close` τερματίζει τη σύνδεση του αρχείου με το πρόγραμμα.
- Τέλος γράφουμε το περιεχόμενο του αρχείου στην οθόνη.

Ένα αρχείο το οποίο είναι πιθανόν πολύ μεγάλο είναι οποσδήποτε προτιμότερο να διαβαστεί κατά τμήματα. Η μέθοδος `readline()` διαβάζει μία γραμμή από αρχείο.

Παράδειγμα. (μέθοδος `readline`) Διαβάστε το περιεχόμενο ενός αρχείου κατά γραμμές και εκτυπώστε τες στην οθόνη.

```
f = open('test.dat','r')
while True:
    s = f.readline()
    if s == '': break
    print(s,end='')    # είτε: print(s[:-1]), print(s.rstrip())
f.close()
```

Ακολουθήσαμε τα εξής βήματα:

- Συνδέουμε το αρχείο (`open`) δηλώνοντας ότι θα διαβάσουμε από αυτό.
- Αρχίζουμε μία ανακύκλωση (`while`) διαβάζοντας διαδοχικές γραμμές με τη μέθοδο `readline()` και το περιεχόμενο πηγαίνει στη μεταβλητή *s*

- Μετά την τελευταία γραμμή του αρχείου η `readline` θα δώσει μία κενή συμβολοσειρά. Αυτό το εκμεταλλευόμαστε για να λήξει η ανακύκλωση.
- Τυπώνουμε τη γραμμή την οποία διαβάσαμε. Όμως, κάθε γραμμή που διαβάζουμε τελειώνει με τον χαρακτήρα `\n`. Για να αποφύγουμε την αλλαγή γραμμής σε κάθε εκτύπωση του *s* δίνουμε στην εντολή `print` ως τελευταίο χαρακτήρα τον κενό.
- Τερματίζουμε τη σύνδεση του αρχείου με το πρόγραμμα.

Ερώτηση. Τι θα συμβεί με το παραπάνω πρόγραμμα αν το αρχείο περιέχει σε κάποιο σημείο μία κενή γραμμή;

Παράδειγμα. (Ανακύκλωση στο αντικείμενο του αρχείου) Υπάρχει μία ακόμα μέθοδος για να διαβάσουμε ένα αρχείο χρησιμοποιώντας απευθείας το αντικείμενο του αρχείου. Μπορούμε να διαβάσουμε το περιεχόμενο ενός αρχείου κατά γραμμές.

```
f = open('test.dat','r')
for line in f:
    print(line,end='')
f.close()
```

Ακολουθήσαμε τα εξής βήματα:

- Συνδέουμε το αρχείο (`open`).
- Κάνουμε μία ανακύκλωση (`for`) στις διαδοχικές γραμμές του αρχείου διατρέχοντας απευθείας το αντικείμενο του αρχείου *f*.
- Τυπώνουμε τη γραμμή την οποία διαβάσαμε αποφεύγοντας την αλλαγή γραμμής.
- Τερματίζουμε τη σύνδεση του αρχείου με το πρόγραμμα.

Ερώτηση. (Μέθοδος `readlines`) Δείτε πώς μπορείτε να διαβάσετε το περιεχόμενο αρχείου με τη μέθοδο `readlines` (όπως στον ακόλουθο κώδικα).

```
f = open('test.dat','r')
f.readlines()
```

Προσάρτηση δεδομένων. Μπορούμε να ανοίξουμε ένα αρχείο για προσάρτηση (append) δεδομένων:

```
f = open('test.dat','a')
```

Ανάγνωση και αποθήκευση. Μπορούμε να ανοίξουμε ένα αρχείο για ανάγνωση και για αποθήκευση δεδομένων:

```
f = open('test.dat','r+')
```

Πρόβλημα. Έστω το αρχείο το οποίο δημιουργείται με τον κώδικα:

```
```

```
import math
f = open('test.dat','w')
for i in range(10):
    s = '{:4d},  {:5d},   {:5.3f}\n'.format(i,i**2,math.sqrt(i))
    f.write(s)
f.close()
```

Διαβάστε το αρχείο και εκχωρήστε σε μεταβλητές κάθε έναν από τους τρεις αριθμούς σε μία γραμμή του αρχείου. μετά ελέγξτε αν πραγματικά το 2ος αριθμός είναι το τετράγωνο του πρώτου και ο τρίτος είναι η τετραγωνική του ρίζα.

```
import math
f = open('test.dat','r')
for line in f:
    line = line.split(',')
    i = line[0].strip(); i = int(i)
    i2 = line[1].strip(); i2 = int(i2)
    isq = line[2].strip(); isq = float(isq)
    if i2 != i**2:
        print('line {:3d}:  {:4d} != {:3d}**2'.format(i,i2,i))
    elif abs(isq-math.sqrt(i))>1e-3:
        print('line {:3d}:  {:f} != sqrt({:3d})'.format(i,isq,i))
    else:
        print(i,i2,isq)
f.close()
```