

Χρήσιμες εντολές για διαχείριση πινάκων

Δημιουργία 2-D πίνακα με τη χρήση της μεθόδου append

Μέχρι τώρα είδαμε διάφορους τρόπους ορισμού/δημιουργίας πινάκων με τη χρήση numpy μεθόδων ή δημιουργίας άδειας λίστας την οποία και γεμίζουμε ανάλογα.

Θα δούμε πως μπορούμε να διαχειριστούμε ανάλογες περιπτώσεις απ' ευθείας με τη δημιουργία πινάκων

```
row = int(input("Enter the number of rows:"))
column = int(input("Enter the number of columns:"))
matrix = [] # Initialize empty matrix

print("Enter the entries row wise:")
for i in range(row):          # A outer for loop for row entries
    a = []
    for j in range(column):    # A inner for loop for column entries
        a.append(int(input()))
    matrix.append(a)

# For printing the matrix
for i in range(row):
    for j in range(column):
        print(matrix[i][j], end = " ")
    print()
```

Εύρεση αριθμού γραμμών και στηλών ενός 2-D πίνακα

Η συνάρτηση **len(listname)** επιστρέφει το μέγεθος (size) μιας λίστας.

Αν η list είναι 2 διαστάσεων, η len() επιστρέφει τον αριθμό των γραμμών της list, που είναι η κύρια διάσταση μιας list.

Αν θέλουμε τον αριθμό των στηλών της list, θα πρέπει να δώσουμε ως όρισμα στη συνάρτηση len, μια γραμμή της list για να μας επιτρέψει το πλήθος των στοιχείων της, δηλαδή τον αριθμό των στηλών

```
an_array= [[1,2], [3,4]]
Rows = len(an_array)      # Number of rows
Cols = len(an_array[0])   # an_array[0] is the first row
Total_length = Rows * Cols
print("Total length is: ", Total_length)
```

Αν η list ήταν ένας numpy array, θα μπορούσαμε να χρησιμοποιήσουμε την μέθοδο **np.size(listname,axis)** για να βρούμε τον αριθμό των γραμμών και στηλών του.

Αν το 2^ο όρισμα δεν υπάρχει, η μέθοδος επιστρέφει το πλήθος των στοιχείων της list.

Αν axis = 0, επιστρέφεται ο αριθμός των γραμμών

Αν axis = 1, επιστρέφεται ο αριθμός των στηλών

```
an_array= [[1,2], [3,4]]
Rows = np.size(an_array,0)    # Number of rows
Cols = np.size(an_array,1)    # Number of columns
Total_length = np.size(an_array) # Rows * Cols
```

List Comprehension

Έστω ότι θέλουμε να εξαγάγουμε κάθε χαρακτήρα από ένα αντικείμενο τύπου string και να τον εισάξουμε σε μια λίστα

Ένας τρόπος είναι να χρησιμοποιήσουμε ένα *for loop*

```
LetList = []  
word = 'banana'  
for letter in word:  
    LetList.append(letter)  
print(LetList)
```

Output:

`['b','a','n','a','n','a']`

Η Python προσφέρει έναν πιο γρήγορο τρόπο για να κάνουμε την ίδια διαδικασία. Η μέθοδος ονομάζεται **list comprehension** και παρέχει τον ορισμό και δημιουργία μιας λίστας στηριζόμενη σε προϋπάρχουσα λίστα

```
word = 'banana'  
LetList = [ letter for letter in word]  
print(LetList)
```

Στο παραπάνω παράδειγμα, δημιουργείται μια νέα λίστα που ανατίθεται στη μεταβλητή **LetList** και περιέχει τα στοιχεία της **word** που είναι η string μεταβλητή για τη λίστα/ακολουθία 'banana'

List Comprehension

Η σύνταξη της δομής list comprehension είναι η ακόλουθη:

```
[ expression for item in list ]
```

```
[ expression for item in list ]
```

```
[ letter for letter in 'banana' ]
```



Στο προηγούμενο παράδειγμα είχαμε :

Παρόλο η σταθερά 'banana' είναι τύπου string και όχι list, η δομή ερμηνεύει την ακολουθία ως list. Το ίδιο συμβαίνει και με τα tuples.

Η δομή της list comprehension μπορεί να συνδυαστεί με μια άλλη δομή list comprehension, ή μια συνθήκη.

List Comprehension – με συνθήκη

Χρησιμοποιώντας μια συνθήκη στη θέση της expression στη list comprehension, μπορούμε να μεταβάλλουμε μια υπάρχουσα list ή tuple

```
AList = [ x for x in range(20) if x%4 == 0 ]  
print(AList)
```

Output:

[4, 8, 12, 16]

Η λίστα AList γεμίζει με τα στοιχεία της λίστας [0,19] τα οποία διαιρούνται με το 4.

List Comprehension – με φωλιασμένη συνθήκη

Μπορεί στη θέση της συνθήκης να υπάρχει φωλιασμένη συνθήκη κάνοντας την συνθήκη αρκετά πιο πολύπλοκη για την επιλογή από την προϋπάρχουσα list

```
AList = [x for x in range(100) if x%2 == 0 if x%5 == 0]  
print(AList)
```

Output:

[0,
10,
20,
30,
40,
50,
60,
70,
80,
90]

Η list comprehension στην περίπτωση αυτή επιλέγει ως στοιχεία της list AList αυτά που ικανοποιούν τις συνθήκες:

- x διαιρείται με το 2
- x διαιρείται με το 5

☐ Αν ικανοποιούνται και οι 2 συνθήκες, το στοιχείο x φυλάσσεται στην list AList

List Comprehension – με if ... else... συνθήκη

Θα μπορούσε η συνθήκη υπόθεσης να είναι περισσότερο πολύπλοκη:

```
AList = ['Even' if x%2 == 0 else 'Odd' for x in range(10)]  
print(AList)
```

Output:

['Even',
 'Odd',
 'Even',
 'Odd',
 'Even',
 'Odd',
 'Even',
 'Odd',
 'Even',
 'Odd']

Η list comprehension στην περίπτωση αυτή επιλέγει ως στοιχεία της list AList :

- 'Even' αν το x διαιρείται με το 2
- 'Odd' αν το x δεν διαιρείται με το 2

List Comprehension – με φωλιασμένο loop

Σε ασκήσεις του προηγούμενου εργαστηρίου υπάρχει στις προτεινόμενες λύσεις η χρήση ενός φωλιασμένου loop

```
AList = [[random.randint(0,100) for y in range(3)] for x in range(4)]  
print(AList)
```

Στην περίπτωση αυτή δημιουργείται μια list με 3 στήλες και 4 γραμμές

Ο τρόπος που δουλεύει το φωλιασμένο loop στην περίπτωση αυτή, είναι να πάρει το x μια τιμή από το εύρος [0,4) (αυτή θα είναι η γραμμή της AList) και κατόπιν να επιλέξει τιμές του y στο διάστημα [0,3) (οι στήλες της γραμμής).

Η τιμή κάθε στοιχείου επιλέγεται τυχαία στο κλειστό διάστημα [0,100] με τη randint

Συνάρτηση **zip()**

Η συνάρτηση `zip()` παίρνει αντικείμενα τύπου ακολουθίας (`list`, `tuples`, `sets`, `strings`, `dict`) και δημιουργεί μια ακολουθία με `tuples` τα στοιχεία των οποίων σε κάθε θέση στην ακολουθία, είναι τα στοιχεία των επιμέρους ακολουθιών στη θέση αυτή

Το `tuple` που επιστρέφει έχει μέγεθος ίσο με το μικρότερο μέγεθος του συνόλου των ακολουθιών που χρησιμοποιούνται

Η σύνταξη που ακολουθείται είναι: `zip(*akolouthies)` όπου `*akolouthies` = τυχαίος αριθμός ακολουθιών

Η συνάρτηση επιστρέφει μια ακολουθία από `tuples`

- Αν δώσουμε άδεια ακολουθία, τότε η συνάρτηση `zip` επιστρέφει μια κενή ακολουθία
- Αν δώσουμε μια ακολουθία, τότε επιστρέφει μια ακολουθία από `tuples` όπου κάθε `tuple` περιέχει ένα μόνο στοιχείο
- Αν δώσουμε περισσότερα από μια ακολουθίες τότε επιστρέφει μια ακολουθία από `tuples` όπου κάθε `tuple` περιέχει στοιχεία από όλες τις επιμέρους ακολουθίες

Την συνάρτηση την έχουμε χρησιμοποιήσει στην περίπτωση της μεθόδου `np.savetxt` για να γράψουμε 2 `listes` ως μία: (π.χ. `Lab03`, άσκηση 8)

```
Xinp = np.arange(-20,21,1)
YYinp = XXinp**2 + 2*XXinp + 1
np.savetxt("lab03_prob08_np.out",list(zip(XXinp,YYinp)),fmt='%3d %6d')
```


Η συνάρτηση zip() – παραδείγματα

```
AList =[1, 2, 3]
NList =[‘one’, ‘two’, ‘three’]
result = zip(AList,NList) # epistrofi mias akoloythias
result_list = list(result) # Metatropi tis akoloythias se list
print(result_list)
```

Το πρόγραμμα θα επιστρέψει: [(1,‘one’), (2,‘two’), (3,‘three’)]

```
AList =[1, 2, 3]
NList =[‘one’, ‘two’]
Plist =[‘ONE’, ‘TWO’, ‘THREE’, ‘FOUR’]
result = zip(AList,NList,Plist)
result_list = list(result)
print(result_list)
```

Το πρόγραμμα θα επιστρέψει: [(1,‘one’, ‘ONE’), (2,‘two’, ‘TWO’)]

```
AList =[1, 2, 3]
NList =[‘o’, ‘t’, ‘z’]
result = zip(AList,Nlist)
result_list = list(result)
print(result_list)
Num,Let=zip(*result_list) # kanei unzip ti lista
print(‘num=’,Numb,‘\nLet = ‘,Let)
```

Το πρόγραμμα επιστρέφει:

[(1,‘o’), (2,‘t’), (3,‘z’)]

num = (1, 2, 3)

Let = (‘o’, ‘t’, ‘z’)

Ανώνυμες συναρτήσεις - συναρτήσεις **lambda**

Συναρτήσεις που ορίζονται χωρίς όνομα, αποτελούν τις λεγόμενες ανώνυμες συναρτήσεις ή συναρτήσεις **lambda**.


Οι κανονικές συναρτήσεις ορίζονται χρησιμοποιώντας τη λέξη κλειδί – **def** – οι ανώνυμες συναρτήσεις ορίζονται με τη λέξη κλειδί **lambda**

Η σύνταξη για μια **lambda** συνάρτηση είναι η ακόλουθη:

lambda ορίσματα: **expression**

Οι συναρτήσεις **lambda** μπορεί να έχουν οποιοδήποτε αριθμό ορισμάτων, **αλλά μόνο μια** **expression** η οποία υπολογίζεται και επιστρέφεται από τη συνάρτηση

ορίσματα: **expression**



```
double = lambda x: x*2
print(double(5))
```

Στο παραπάνω παράδειγμα, η συνάρτηση δεν έχει όνομα, ωστόσο επιστρέφει ένα αντικείμενο τύπου συνάρτησης, το οποίο ανατίθεται στο `double`. Χρησιμοποιώντας την `double` μπορούμε να την καλέσουμε ως συνηθισμένη συνάρτηση.

Η εντολή ανάθεσης: `double = lambda x: x*2` είναι ίδια με

```
def double(x):
    return x*2
```

Χρήση lambda συναρτήσεων

Lambda συναρτήσεις χρησιμοποιούνται σε περιπτώσεις που χρειαζόμαστε μια ανώνυμη συνάρτηση για μικρό χρονικό διάστημα.

Συνήθως χρησιμοποιούνται σε συναρτήσεις υψηλότερης τάξης, οι οποίες χρησιμοποιούν απλούστερες συναρτήσεις ως όρισμά τους

Χρησιμοποιούνται στις συναρτήσεις **filter()** και **map()** της python

➤ Η συνάρτηση **filter()** έχει ως ορίσματα μια συνάρτηση και μια list

- ❑ Η συνάρτηση καλείται για κάθε στοιχείο της list και επιστρέφει μια νέα list με στοιχεία αυτά για τα οποία η συνάρτηση έχει σαν αποτέλεσμα True

```
AList = [1, 5, 4, 6, 8, 11, 3, 12]
newList = list(filter(lambda x: (x%2==0), AList))
print(newList)
```

Output:
[4, 6, 8, 12]

➤ Η συνάρτηση **zip()** έχει ως ορίσματα μια συνάρτηση και μια list

- ❑ Η συνάρτηση καλείται για κάθε στοιχείο της list και επιστρέφει μια νέα list με στοιχεία αυτά τα οποία υπολογίζει η συνάρτηση για κάθε στοιχείο της λίστας

```
AList = [1, 5, 4, 6, 8, 11, 3, 12]
newList = list(map(lambda x: x*2, AList))
print(newList)
```

Output:
[2, 10, 8, 12, 16, 22, 6, 24]

Συναρτήσεις **lambda**

Οι συναρτήσεις `lambda` μπορούν να χρησιμοποιηθούν για τον ορισμό και δημιουργία μιας λίστας και επομένως μπορούν να συγκριθούν με τη δομή της `list comprehension`

```
LetList = list(map(lambda x: x, 'banana'))  
print(LetList)
```

Output:

`['b','a','n','a','n','a']`

Η συνάρτηση print ()

Όπως έχουμε χρησιμοποιήσει μέχρι τώρα, η συνάρτηση **print()** χρησιμοποιείται για να τυπώσουμε κάποιο αντικείμενο είτε στην οθόνη ή σε κάποιο εξωτερική διάταξη.

Η πλήρης σύνταξη της συνάρτησης είναι:

```
print(object(s), sep=separator, end=end, file=filename, flush=flush)
```

- **object(s)** – τα αντικείμενα που θα εκτυπωθούν
- **sep** – ο τρόπος διαχωρισμού των αντικειμένων – by default είναι " "
- **end** – προσδιορίζει ποιο αντικείμενο θα πρέπει τουλάχιστον να εκτυπωθεί
- **filename** – όνομα αρχείου για εκτύπωση που είναι διαθέσιμο για εγγραφή
- **flush** – καθαρισμός του output stream. Αυτό είναι False by default.

Συνήθως στο τέλος κάθε εκτέλεσης της εντολής print υπάρχει end='\\n', αλλαγή γραμμής.

Έχουμε συναντήσει αρκετές περιπτώσεις στις οποίες θα θέλαμε να διατηρηθεί η ίδια γραμμή μετά την εκτύπωση ενός αντικειμένου.

Για να διατηρήσουμε εκτύπωση στην ίδια γραμμή, χρησιμοποιούμε το print με την επιλογή end=""

```
print('b=',a,end=' ')
```

Η συνάρτηση print()

Έστω ότι θέλουμε να τυπώσουμε μια λίστα με 6 αριθμούς σε μορφή 2 γραμμών με 3 στήλες

Α' Τρόπος

```
A = [ 1, 2, 3, 4, 5, 6]
for ii in range(0,6,3):
    print(A[ii+0],A[ii+1],A[ii+2])
```

Β' Τρόπος

```
A = [ 1, 2, 3, 4, 5, 6]
for ii in range(0,6,3):
    for jj in range(3):
        print(A[ii+jj],end=' ') # Diatiroume tin idia grammi sto jj loop
    print()                    # Allagi grammis meta to telos tou jj loop
```

Dictionaries

Η έννοια της συλλογής - collection

Μια συλλογή αποτελεί ένα καλό είδος αντικειμένου γιατί μπορούμε να κάνουμε ανάθεση περισσότερες από μία τιμές και να τις χρησιμοποιήσουμε ή να τις μεταφέρουμε σε διάφορα σημεία ενός προγράμματος

Δηλαδή έχουμε διάφορες τιμές σε μια μόνο μεταβλητή

Αυτό επιτυγχάνεται έχοντας περισσότερες θέσεις στην μεταβλητή που χρησιμοποιούμε και έχοντας τρόπους να βρούμε τις θέσεις αυτές μέσα στη μεταβλητή

Στον αντίποδα, οι περισσότερες μεταβλητές που χρησιμοποιούμε περιέχουν μια και μόνο τιμή.

Κάθε φορά που κάνουμε ανάθεση μιας νέας τιμής στην μεταβλητή, η προηγούμενη τιμή διαγράφεται

```
>>> x = 2  
>>> x = 4  
>>> print(x)  
4
```


Δύο είδη συλλογών

Lists: Μια γραμμική συλλογή τιμών που παραμένει σε τάξη

Dictionary: Μια «σακκούλα» με τιμές αλλά η κάθε μια με τη δική της ετικέτα



χαρτομάντηλα

διαβατήριο

καραμέλες

χρήματα

Dictionaries στην Python

Dictionary: αποτελεί την πλέον αποτελεσματική και χρήσιμη μέθοδο για δημιουργία μιας συλλογής δεδομένων

Dictionary: επιτρέπουν γρήγορους υπολογισμούς και πράξεις με βάση δεδομένων

Dictionary: συναντώνται και σε άλλες γλώσσες προγραμματισμού

Στις λίστες, δημιουργείται ένας κατάλογος με τη θέση της κάθε τιμής στη λίστα. Η πρόσβαση γίνεται με βάση τον κατάλογο αυτό

Τα dictionaries προσφέρουν περιεχόμενο χωρίς να είναι ταξινομημένο

Για να δημιουργήσουμε κατάλογο για τιμές που υπάρχουν στο dictionary χρησιμοποιούμε ετικέτες αναφοράς:

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

Σύγκριση λίστας και dictionary

Τα dictionaries είναι όπως οι λίστες, με τη διαφορά ότι χρησιμοποιούν ετικέτες (**keys**) για την εύρεση των τιμών που είναι αποθηκευμένες σε αυτά αντί των θέσεων που χρησιμοποιούνται στις λίστες.

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```

Σύγκριση list και dictionary

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

List		
Key	Value	
[0]	21	lst
[1]	183	

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```

Dictionary		
Key	Value	
['course']	182	ddd
['age']	21	

Σύνταξη (σταθερές) των dictionaries και πρόσβαση

Η σύνταξη ενός αντικειμένου τύπου dictionary, χρησιμοποιεί **{ }** μέσα στα οποία περικλείονται μια σειρά από **ζεύγη keys**(ετικέτες) : **τιμές**

Τα keys είναι μοναδικά σε ένα dictionary ενώ οι τιμές τους δεν είναι μοναδικές

Οι τιμές των keys σε ένα dictionary μπορεί να είναι οποιουδήποτε τύπου αλλά τα keys πρέπει να είναι *immutable* τύπου, (strings, αριθμοί ή tuples)

Μπορούμε να κατασκευάσουμε ένα άδειο dictionary χρησιμοποιώντας απλά { }

Για να αποκτήσουμε πρόσβαση σε στοιχεία του dictionary θα χρειαστεί να δώσουμε το όνομα του dictionary ακολουθούμενο από [] και το όνομα του key που ενδιαφέρει

```
>>> dict = { 'fotis' : 1 , 'giannis' : 42, 'maria': 100}
>>> print(dict)
{'maria': 100, 'fotis': 1, 'giannis': 42}
>>> ooo = {}
>>> print(ooo)
{}
>>> print("dict['fotis']:", dict['fotis'])
dict['fotis']: 1
```

Πρόσβαση σε πληροφορία του dictionary

Κάτι ιδιαίτερα χρήσιμο με τα dictionaries είναι να μετρήσουμε πόσο συχνά εμφανίζεται κάποια τιμή

```
>>> ccc = dict()
>>> ccc['key1'] = 1
>>> ccc['key2'] = 1
>>> print(ccc)
{'key2': 1, 'key1': 1}
>>> ccc['key1'] = ccc['key1'] + 1
>>> print(ccc)
{'key2': 1, 'key1': 2}
```

Οδηγούμαστε σε λάθος αν προσπαθήσουμε να αποκτήσουμε πρόσβαση σε key που δεν υπάρχει στο dictionary.

Χρησιμοποιώντας τον τελεστή **in** μπορούμε να ελέγξουμε αν υπάρχει κάποιο key στο dictionary.

```
>>> ccc = dict()
>>> print(ccc['key3'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'key3'
>>> 'csev' in ccc
False
```

Διαχείριση νέων keys

Όταν συναντούμε ένα νέο key στο dictionary, θα πρέπει να προσθέσουμε το key στον κατάλογο των keys του dictionary και αν έχει εμφανιστεί πολλές φορές τότε αρχίζουμε να μετρούμε τις περιπτώσεις που εμφανίζεται

```
counts = dict()
names = ['key1', 'key2', 'key1', 'key3', 'key2']
for name in names :
    if name not in counts:
        counts[name] = 1
    else :
        counts[name] = counts[name] + 1
print(counts)
```

Η μέθοδος **Get** για τα dictionaries

Το γεγονός ότι ελέγχουμε αν κάποιο key προϋπάρχει σε ένα dictionary και αν δεν βρεθεί να κάνουμε ανάθεση μιας συγκεκριμένης τιμής αναφοράς, είναι μια πολύ συνηθισμένη διαδικασία. Για τον σκοπό αυτό υπάρχει η μέθοδος **get()** που κάνει την διερεύνηση αυτή και αναθέτει μια τιμή για keys που δεν βρέθηκαν με τέτοιο τρόπο ώστε το πρόγραμμα να μην σταματά αν δεν βρεθεί το key.

```
dict={'Name': 'Zena', 'Age':7}
print("Value : %s" %dict.get('Age'))
Print("Value : %s"% dict.get('Education','Never') )
```

```
Value : 7
Value : Never
```

Η μέθοδος `get()` απλουστεύει ιδιαίτερα το μέτρημα των keys στο dictionary.

Μπορούμε να δώσουμε μηδενική τιμή όταν δεν βρεθεί το key και στη συνέχεια να προσθέτουμε 1 :

```
counts = dict()
names = ['key1', 'key2', 'key1', 'key3', 'key2']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print(counts)
```

Default τιμή



Output

```
{'key1':2, 'key3':1, 'key2':2}
```


Μέτρημα λέξεων σε κείμενο

Συγκεκριμένα, σήμερα τίθενται επιπλέον σε εφαρμογή τα ακόλουθα:

1. Επαναλειτουργία των δημοτικών σχολείων, Γ' Λυκείου και τεχνικών σχολών στην ιδιωτική και δημόσια εκπαίδευση.
2. Επαναλειτουργία επιχειρήσεων λιανικού εμπορίου, συμπεριλαμβανομένων των πολυκαταστημάτων και εμπορικών κέντρων.
3. Επαναλειτουργία πρακτορείων τυχερών παιγνίων, μόνο για σκοπούς συμπλήρωσης του δελτίου, χωρίς τη χρήση τραπεζοκαθισμάτων.
4. Επαναλειτουργία μουσείων και άλλων αρχαιολογικών χώρων.
5. Η Θεία Λειτουργία στους χώρους θρησκευτικής λατρείας τελείται με την παρουσία πιστών με μέγιστο αριθμό προσώπων μέχρι 50 άτομα, τηρουμένων των υγειονομικών πρωτοκόλλων.
6. Αύξηση αριθμού ατόμων για επισκέψεις σε οικίες, σε τέσσερα (4) άτομα.

Επίσης, σύμφωνα με το νέο διάταγμα που εξέδωσε ο Υπουργός Υγείας, από σήμερα ξεκινά και η σταδιακή και σε φάσεις επανεκκίνηση του τομέα του αθλητισμού, καθώς και το άνοιγμα των θεάτρων και των κινηματογράφων.

Σημειώνεται ότι στο νέο διάταγμα περιλαμβάνεται και ο υποχρεωτικός αυτοπεριορισμός 72 ωρών για όσους φτάνουν στην Κύπρο από το εξωτερικό.

Μέτρηση με βάση κάποιο συγκεκριμένο σχήμα

Η μέθοδος που μπορούμε να ακολουθήσουμε για να μετρήσουμε λέξεις σε ένα κείμενο, είναι να χωρίσουμε κάθε γραμμή του κειμένου (`split`) σε λέξεις και κατόπιν να έχουμε ένα loop ως προς τις λέξεις και να χρησιμοποιήσουμε ένα dictionary για να παρακολουθήσουμε πόσες φορές εμφανίζεται κάθε λέξη ανεξάρτητα από τις άλλες.

```
counts = dict()
line = input("Enter a line of text:")

words = line.split()
print('Words:', words)

for word in words:
    counts[word] = counts.get(word,0) + 1
print('Counts', counts)
```

Dictionaries και προσδιορισμένα loops

Παρόλο που τα dictionaries δεν έχουν τα δεδομένα τους αποθηκευμένα σε τάξη, μπορούμε να χρησιμοποιήσουμε for loops για να εξετάσουμε τα στοιχεία του.

Αυτό το οποίο εκτελεί είναι το πέρασμα όλων των keys και η ανάγνωση των τιμών τους.

```
>>> counts = { 'fotis' : 1 , 'giannis' : 42, 'maria': 100}
>>> for key in counts:
...     print(key, counts[key])
...
maria 100
fotis 1
giannis 42
>>>
```

Εξαγωγή λίστας των keys και τιμών

Από ένα dictionary μπορούμε να εξαγάγουμε μια λίστα με **keys** του ή τιμές (**values**) του ή και ζεύγη των δύο.

Το ζεύγος ονομάζεται **item** του dictionary

Χρησιμοποιούμε τις μεθόδους:

dictname.**keys()**

dictname.**values()**

dictname.**items()**

Μπορούμε να έχουμε ένα loop ως προς το ζεύγος **key-value** ενός dictionary **χρησιμοποιώντας 2 μεταβλητές επανάληψης**

Για κάθε επανάληψη, η πρώτη μεταβλητή αναφέρεται στο key και η δεύτερη στην αντίστοιχη τιμή του key

```
>>> dict = { 'fotis' : 1 , 'giannis' : 42, 'maria': 100}
>>> print(list(dict))
['fotis', 'giannis', 'maria']
>>> print(dict.keys())
['fotis', 'giannis', 'maria']
>>> print(dict.values())
[100, 1, 42]
>>> print(dict.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

```
dict= { 'fotis' : 1 , 'giannis' : 42, 'maria': 100}
for aaa, bbb in dict.items() :
    print(aaa, bbb)
```

	aaa	bbb
maria 100	[maria]	<div>100</div>
fotis 1	[fotis]	<div>1</div>
giannis 42	[giannis]	<div>42</div>

Παράδειγμα μέτρησης λέξεων από ένα file

```
filename = input('Enter file:')
Inpfile = open(filename)

counts = dict()
for line in inpfile:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

Πρόσθεση, τροποποίηση, διαγραφή στοιχείων dictionary

Ένα dictionary μπορεί να τροποποιηθεί προσθέτοντας ένα νέο στοιχείο, ή ένα ζεύγος key-τιμής (item), ή διαγράφοντας ένα προϋπάρχον στοιχείο.

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
dict['Age'] = 8 # update existing entry  
dict['School'] = "Private School" # Add new entry  
print("dict['Age']: ", dict['Age'])  
print("dict['School']: ", dict['School'])
```

Μπορούμε να διαγράψουμε προϋπάρχοντα στοιχεία ενός dictionary, ή να σβήσουμε όλο το περιεχόμενό του. Μπορούμε επίσης να διαγράψουμε το dictionary με μια εντολή

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
del dict['Name'] # remove entry with key 'Name'  
dict.clear() # remove all entries in dict  
del dict # delete entire dictionary  
print("dict['Age']: ", dict['Age'])  
print("dict['School']: ", dict['School'])
```

Ιδιότητες των keys ενός dictionary

Οι τιμές ενός dictionary μπορεί να είναι οτιδήποτε αντικείμενο της Python. Αυτό όμως δεν ισχύει για τα keys τα οποία θα πρέπει:

(α) Περισσότερες από μια entry για κάθε key δεν επιτρέπεται. Σε αντίθετη περίπτωση το τελευταίο στοιχείο είναι αυτό που καθορίζει το συγκεκριμένο key

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}  
Print("dict['Name']: ", dict['Name'])
```

Output

```
dict['Name']: Manni
```

(b) Τα keys είναι immutable και επομένως θα πρέπει να είναι τύπου string, tuple ή number

```
dict = {['Name']: 'Zara', 'Age': 7}  
print "dict['Name']: ", dict['Name']
```

Output

```
Traceback (most recent call last):  
File "test.py", line 3, in <module>  
dict = {['Name']: 'Zara', 'Age': 7};  
TypeError: unhashable type: 'list'
```

Συναρτήσεις και μέθοδοι για dictionaries

`cmp(dict1, dict2)` : σύγκριση στοιχείων μεταξύ δύο dictionaries

`len(dict)`: συνολικό μέγεθος του dictionary

`str(dict)`: δημιουργεί μια string αναπαράσταση του dictionary

`type(variable)`: επιστρέφει τον τύπο της μεταβλητής που εισάγεται

`Dict.clear()`: διαγράφει στοιχεία του dictionary Dict

`Dict.copy()`: επιστρέφει αντίγραφο του dictionary Dict

`Dict.fromkeys()`: δημιουργία ενός νέου dictionary από στοιχεία του Dict

`Dict.get(key, Default=0)`: επιστρέφει τη τιμή του key ή 0 αν δεν υπάρχει

`Dict.has_key(key)`: επιστρέφει True αν υπάρχει το key

`Dict.items()`: επιστρέφει μία list των (keys, values) του Dict

`Dict.keys()`: επιστρέφει μία list των (keys) του Dict

`Dict.values()`: επιστρέφει μία list των (τιμών) του Dict

`Dict.update(dict2)`: προσθέτει τα items του dict2 στο Dict