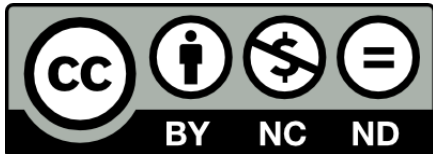


Εισαγωγή στην **Python**

6.2





Copyright

Το παρόν εκπαιδευτικό υλικό προσφέρεται ελεύθερα υπό τους όρους της άδειας Creative Commons:

- *Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Όχι Παράγωγα Έργα 3.0.*

Για να δείτε ένα αντίγραφο της άδειας αυτής επισκεφτείτε τον ιστότοπο

<https://creativecommons.org/licenses/by-nc-nd/3.0/gr/>

Στ. Δημητριάδης, 2015

Περιεχόμενα

- Λεξικό: τα Βασικά χαρακτηριστικά
- Δημιουργία Λεξικού
- Συναρτήσεις & Μέθοδοι Λεξικού

- Ομάδα (Tuple)
- Σύνολο (Set)

Dictionary

Λεξικό



Λεξικό

τα **βασικά** χαρακτηριστικά



Dictionary

```
>>> di = {'GR': 'Greece', 'IT': 'Italy', 'SP': 'Spain'}
```

Key : value

- **Key-value sequence:** Ακολουθία κλειδί:τιμή
 - Ένα λεξικό είναι μια ακολουθιακή δομή από ζεύγη κλειδί:τιμή
 - Γράφεται μέσα σε άγκιστρα { }
- **Mapping:** Δομή αντιστοίχισης
 - Κάθε κλειδί αντιστοιχείται σε μία τιμή και κάθε τιμή προσδιορίζεται με βάση ένα κλειδί (key) και ΟΧΙ αριθμητικό δείκτη
 - Πχ. 'GR' είναι το κλειδί (key) που προσδιορίζει την τιμή 'Greece'
- **Αναφορά σε τιμή:** Η αναφορά σε κάποια τιμή λεξικού γίνεται χρησιμοποιώντας το κλειδί περικλειόμενο σε **αγκύλες** (προσοχή: όχι άγκιστρα)

```
>>> di = {'GR': 'Greece', 'IT': 'Italy', 'SP': 'Spain'}
>>> di['GR']
'Greece'
```

Dictionary

Άλλες ιδιότητες

```
>>> di = { 'GR': 'Greece', 'IT': 'Italy', 'SP': 'Spain' }
```

Key: value

- **Un-ordered:** Αταξινόμητη

- Το Λεξικό είναι μια αταξινόμητη (unsorted) συλλογή δεδομένων – Δεν ταξινομείται

- **Heterogeneous:** Ανομοιογενής

- Μπορεί να περιλαμβάνει δεδομένα διαφορετικού τύπου

- **Mutable:** Μεταλλάξιμη

- Είναι **μεταλλάξιμη** δομή (mutable): δηλ. τα δεδομένα της μπορούν να μεταβληθούν στη θέση μνήμης ('in place') χωρίς να δημιουργηθεί νέα δομή λεξικού

- **Sequential:** Ακολουθιακή

- Ένα λεξικό είναι ακολουθιακή δομή και μπορεί να χρησιμοποιηθεί ως 'iterable' σε μια δομή επανάληψης for



Dictionary

Ζεύγη

key:value

-1

```
>>> di = {'GR': 'Greece', 'IT': 'Italy', 'SP': 'Spain'}
>>> di['GR']
'Greece'
>>> di['IT']
'Italy'
```

- Περιγραφή του dictionary 'di'
- Εμφάνιση της τιμής με κλειδί 'GR'
- Εμφάνιση της τιμής με κλειδί 'IT'

- Ένα dictionary **δεν** έχει συγκεκριμένη σειρά/τάξη των δεδομένων του
- Μπορούν να εμφανίζονται με διαφορετική σειρά

```
>>> di = {'GR': 'Greece', 'IT': 'Italy', 'SP': 'Spain'}
>>> di
{'GR': 'Greece', 'IT': 'Italy', 'SP': 'Spain'}
```

```
>>> di
{'SP': 'Spain', 'IT': 'Italy', 'GR': 'Greece'}
```



Dictionary

Ζεύγη

key:value

-2

- Τα κλειδιά είναι **μοναδικά** σε ένα λεξικό – δεν επαναλαμβάνονται
- Αν δηλωθεί νέο ζεύγος με **ίδιο κλειδί** τότε κρατά μόνον το **τελευταίο**

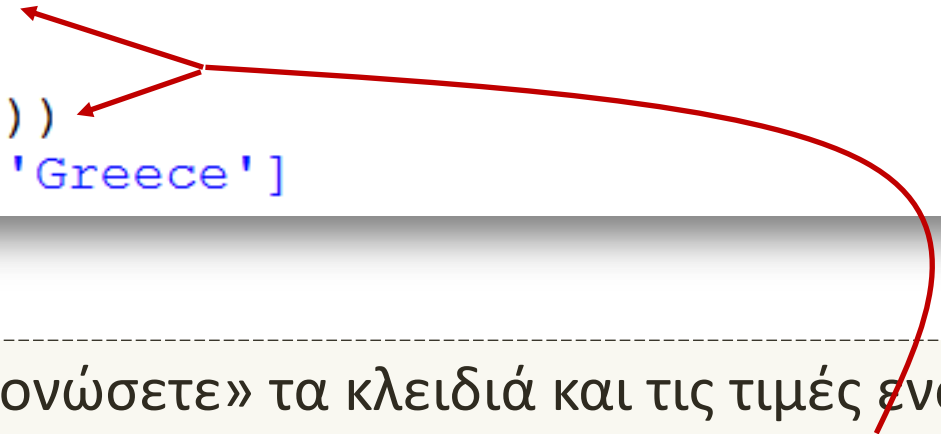
```
>>> di = {'GR':'Greece', 'GR':'greece', 'IT':'Italy', 'SP':'Spain'}  
>>> di  
{'SP': 'Spain', 'IT': 'Italy', 'GR': 'greece'}
```

```
>>> di = {'GR':'Greece', 'gr':'Greece', 'IT':'Italy', 'SP':'Spain'}  
>>> di  
{'SP': 'Spain', 'gr': 'Greece', 'IT': 'Italy', 'GR': 'Greece'}
```

- Αντίθετα οι **τιμές** μπορεί να επαναλαμβάνονται
- Μπορείτε να έχετε πολλά ζεύγη με **ίδια τιμή**



```
>>> di = {'GR':'Greece', 'IT':'Italy', 'SP':'Spain'}
>>> di.keys()
dict_keys(['SP', 'IT', 'GR'])
>>> list(di.keys())
['SP', 'IT', 'GR']
>>> list(di.values())
['Spain', 'Italy', 'Greece']
```



- Μπορείτε να «απομονώσετε» τα κλειδιά και τις τιμές ενός λεξικού σε ιδιαίτερες λίστες καλώντας τις μεθόδους **keys()** & **values()** αντίστοιχα...
- ...και δημιουργώντας λίστα με τη μέθοδο **list()**

Dictionary

Τύπος κλειδιών & τιμών

```
>>> di = {1:'Greece', 2:'Italy', 3:'Spain'}
>>> di[1]
'Greece'
>>> di[3]
'Spain'
```

- Τα κλειδιά μπορεί να είναι **ακέραιοι**
- Γενικά ως κλειδί σε ένα λεξικό μπορεί να χρησιμοποιηθεί κάθε τύπος που είναι **αμετάλλακτος (immutable)**, όπως: αλφαριθμητικά, ακέραιοι, ομάδες
- Αντίθετα οι τιμές μπορεί να είναι **οποιοδήποτε** τύπου

```
>>> D = {'spam': 'SPAM', (7,8):15, 100:[1,2,3]}
>>> D
{100: [1, 2, 3], (7, 8): 15, 'spam': 'SPAM'}
```



Δημιουργία Λεξικού



Dictionary

α) Δημιουργία με ανάθεση κλειδιού:τιμής (key:value)

```
di = {}  
di['GR'] = 'Greece'  
di['IT'] = 'Italy'  
di['SP'] = 'Spain'  
  
print(di)
```

- Αρχικοποίηση (κενό dict)
- Δημιουργία του dictionary 'di' με ανάθεση τιμής (δημιουργία ζεύγους key:value)
- Εμφάνιση

```
>>>  
{'GR': 'Greece', 'IT': 'Italy', 'SP': 'Spain'}  
>>>
```

Dictionary

β) Δημιουργία με τη **dict()** -1/2

```
lista = [('GR', 'Greece'), ('IT', 'Italy'), ('SP', 'Spain')]
di = dict(lista)

print(di)
```

- Μια **λίστα δυάδων** (list of tuples) μπορεί να μετατραπεί σε λεξικό με χρήση της συνάρτησης **dict()** όπως φαίνεται στο παράδειγμα

```
>>>
{'IT': 'Italy', 'SP': 'Spain', 'GR': 'Greece'}
>>>
```

Dictionary

β) Δημιουργία με τη **dict()** -2/2

```
di = dict(GR='Greece', IT='Italy', SP='Spain')  
print(di)
```

- Η **dict()** μπορεί να πάρει ως όρισμα και μια σειρά από απλές εντολές ανάθεσης ώστε να δημιουργήσει το αντίστοιχο λεξικό
- Παρατηρήστε ότι στην περίπτωση αυτή τα κλειδιά εμφανίζονται ως απλά ονόματα μεταβλητών (όχι σαν αλφαριθμητικά μέσα σε εισαγωγικά)

Dictionary γ) Δημιουργία με ‘περιγραφή’

(dictionary comprehension)

```
key_list = ['GR', 'IT', 'SP']  
val_list = ['Greece', 'Italy', 'Spain']  
  
di = {key_list[x]:val_list[x] for x in range(3)}  
  
print(di)
```

- Το λεξικό μπορεί να δημιουργηθεί και με ‘περιγραφή’ (comprehension) κατ’ αναλογία με την περιγραφή λίστας
- Στο παράδειγμα ‘περιγράφεται’ η αντιστοιχία **κλειδιού:τιμής** με βάση τις λίστες key_list & val_list

Λίστα & Λεξικό: Συνδυασμοί



Συνδυάζοντας **Λίστα** και **Λεξικό**

- Συνδυάζοντας λίστα και λεξικό μπορείτε να δημιουργήσετε σύνθετες και εξαιρετικά ευέλικτες δομές για κάθε είδος υπολογιστικού προβλήματος που αντιμετωπίζετε
- Ακολουθούν δύο χαρακτηριστικά παραδείγματα

(α) Λεξικό με **λίστα** ως τιμή

```
# Σχήμα DICT {Code:[CountryName, Capital, Pop]}
country = {}
country['GR'] = ['Greece', 'Athens', 11]
country['IT'] = ['Italia', 'Rome', 60]
country['SP'] = ['Spain', 'Madrid', 50]

for c in country:
    print(c, country[c])

while True:
    epil = input("Κωδικός Χώρας ('q' to Quit): ")
    if epil != 'q':
        print(country[epil])
    else:
        break
```

- Οι τιμές του λεξικού country είναι λίστες με τα στοιχεία της κάθε χώρας

(β) Λεξικό με **λεξικό** ως τιμή

```
# DICT {Code:{CountryName:, Capital:, Pop:}}
country = {}
country['GR'] = {'Name':'Greece',
                 'Capital':'Athens',
                 'Pop':'11 milions'}
country['IT'] = {'Name':'Italy',
                 'Capital':'Rome',
                 'Pop':'60 milions'}
country['SP'] = {'Name':'Spain',
                 'Capital':'Madrid',
                 'Pop':'50 milions'}

for c in country:
    print(c, country[c])

while True:
    epil = input("Κωδικός Χώρας ('q' to Quit): ")
    if epil != 'q':
        print(country[epil]['Name'])
        key2 = input("Search 'Capital / Pop': ")
        print(country[epil][key2])
    else:
        break
```

- Οι τιμές του λεξικού country είναι **λεξικά** με τα στοιχεία της κάθε χώρας
- Προσέξτε τον τρόπο με τον οποίο γίνεται αναφορά στα στοιχεία αυτών των λεξικών:
 - (α) με το αλφαριθμητικό κλειδί
 - (β) με **μεταβλητή** τύπου αλφαριθμητικού

Λίστα & Λεξικό: Σύγκριση

Υλοποίηση Λεξικού: **Hash table**



- Δύο ευέλικτες δομές της Python
- **Λίστα**: **δεικτοδοτημένη** δομή (indexed) που αντιστοιχεί τιμές σε θέσεις. Μπορεί να είναι ταξινομημένη (sorted)
- **Λεξικό**: δομή **αντιστοίχισης** (mapping) (ακολουθιακή μεν αλλά αταξιινόμητη) που αντιστοιχεί κλειδιά σε τιμές (μέσω μιας συνάρτησης κατακερματισμού – hash table). Δεν ταξινομείται (un-sorted)
- Ένα λεξικό συχνά είναι δομή με **περισσότερο νόημα και λογική** για τον άνθρωπο-χρήστη

Παράδειγμα

```
>>> L = ['Greece', 11, ['Athens', 'Salonica']]
>>> L[0]
'Greece'
>>> L[1]
11
>>> L[2][1]
'Salonica'
```

```
>>> D = {'Country':'Greece', 'Population':11, 'Cities':['Athens', 'Salonica']}
>>> D['Country']
'Greece'
>>> D['Population']
11
>>> D['Cities']
['Athens', 'Salonica']
```

- Η ίδια «εγγραφή» υλοποιημένη με λίστα (επάνω) και λεξικό (κάτω)
- Η διαχείριση των δεδομένων στο λεξικό με τη χρήση κλειδιών μπορεί να είναι περισσότερο κατανοητή

- Στην πράξη τα Λεξικά προτιμούνται στις εξής περιπτώσεις:

1. Δομές δεδομένων με **ετικέτα**, κατάλληλες για **γρήγορη αναζήτηση** με μνημονικά κλειδιά (ονόματα) (σε αντίθεση με την πιο αργή διαδικασία γραμμικής αναζήτησης σε λίστα)

2. **Αραιές** (sparse) συλλογές δεδομένων σε "αυθαίρετες" θέσεις

Πχ. $D = \{ \}$

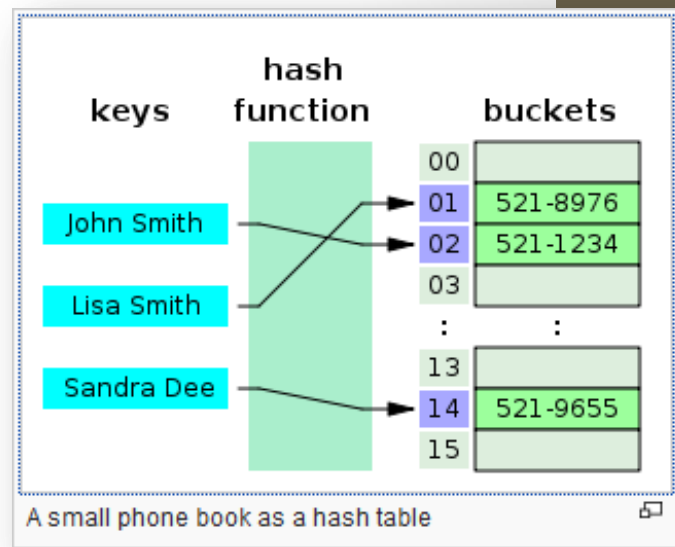
$D[99] = \text{'spam'}$

Πχ. $\text{table} = \{1975: \text{'Holy Grail'},$
 $1979: \text{'Life of Brian'},$
 $1983: \text{'The Meaning of Life'}\}$

Dictionary

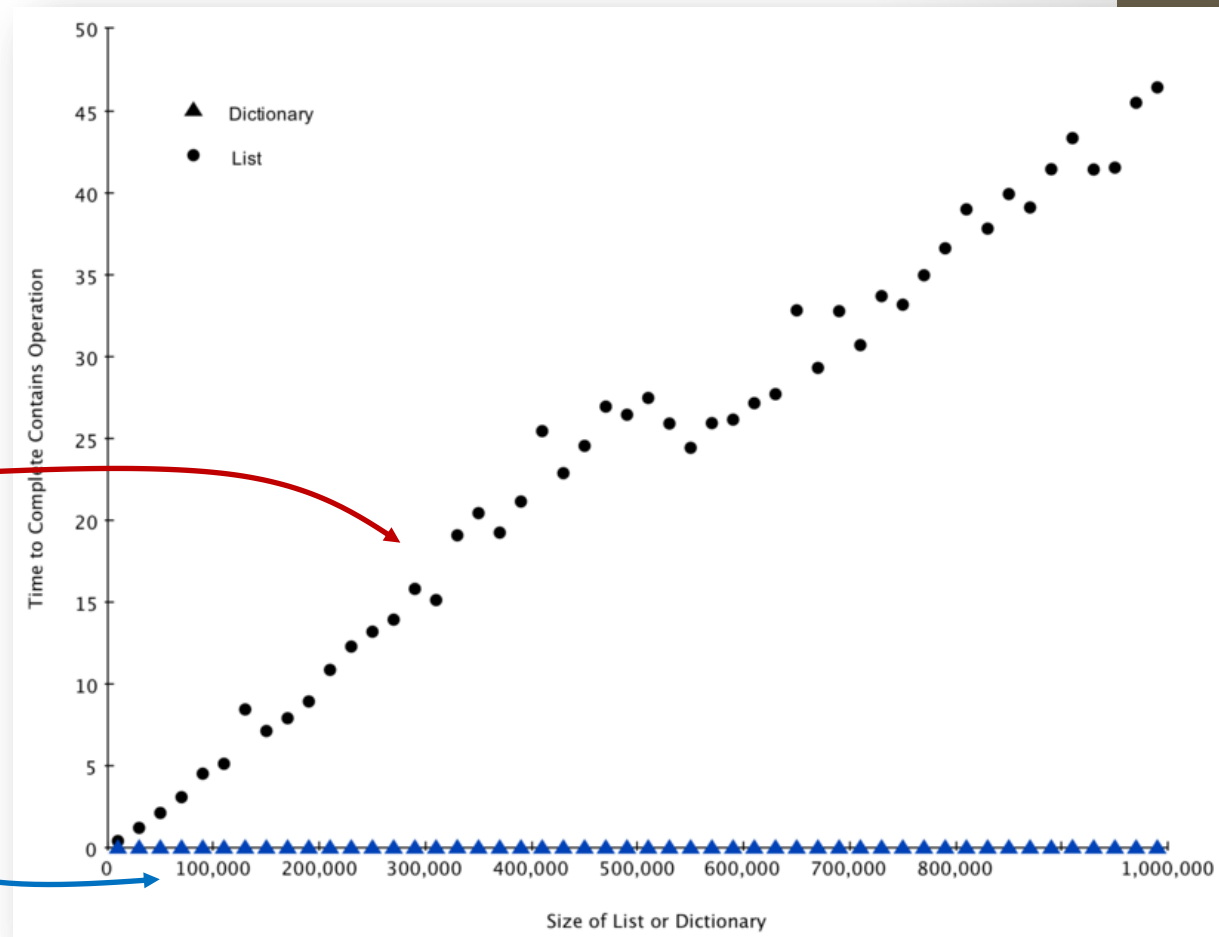
Πίνακας κερματισμού

- Τεχνικά ένα λεξικό υλοποιείται μέσω ενός **πίνακα κερματισμού** (hash table)
- Ένας πίνακας κατακερματισμού χρησιμοποιεί μια **συνάρτηση κερματισμού** (hash function) ώστε να αντιστοιχίσει ένα σύνολο κλειδιών (keys) σε ένα πίνακα τιμών (values ή buckets)
- Το πλεονέκτημα αυτής της τεχνικής είναι ότι ο προσδιορισμός ενός δεδομένου για διαχείρισή του (πχ. εισαγωγή, αναζήτηση και διαγραφή) γίνεται σε **σταθερό χρόνο**, δηλαδή είναι της τάξης **$O(1)$**
- Αντίθετα σε μια δεικτοδοτημένη δομή όπως πχ. η λίστα, ο προσδιορισμός ενός δεδομένου γίνεται με χρόνο ανάλογο προς το πλήθος n των στοιχείων της λίστας, δηλ. είναι της τάξης **$O(n)$**



Σύγκριση χρόνου αναζήτησης με τον τελεστή 'in'

- Καθώς αυξάνει το πλήθος στοιχείων (οριζόντιος άξονας) **αυξάνει γραμμικά** ο χρόνος αναζήτησης στη **λίστα** ($O(n)$) ενώ **μένει σταθερός** στο **λεξικό** ($O(1)$)



Πηγή: [Problem Solving with Algorithms and Data Structures](#)

Παράδειγμα: Χρόνος αναζήτησης σε Λίστα & Λεξικό

```
import time

lista = [i for i in range(1000001)]
n_key = ['n'+str(i) for i in lista]

di = {n_key[i]:lista[i] for i in lista}

n = int(input('Number: '))
n_key = 'n'+str(n)

start = time.time()
print(lista[n])
stop = time.time()
print('List time = ', stop-start)

start = time.time()
print(di[n_key])
stop = time.time()
print('Dict time = ', stop-start)
```

- Τρέχοντας τον κώδικα μπορείτε να συγκρίνετε τους χρόνους που χρειάζεται για να εντοπιστεί κάποιο στοιχείο σε Λίστα και σε Λεξικό

Συναρτήσεις & Μέθοδοι Λεξικού



keys() & values()

```
>>> di = {'GR': 'Greece', 'IT': 'Italy', 'SP': 'Spain'}
>>>
>>> di.keys()
dict_keys(['GR', 'IT', 'SP'])
>>>
>>> list(di.keys())
['IT', 'GR', 'SP']
>>>
>>> sorted(di.keys())
['GR', 'IT', 'SP']
```

- Η **di.keys()** επιστρέφει τα κλειδιά (όχι ταξινομημένα)
- Η **list(di.keys)** επιστρέφει λίστα κλειδιών (όχι ταξινομημένα)
- Η **sorted(di.keys())** επιστρέφει ταξινομημένη λίστα κλειδιών
- Αντίστοιχα μπορείτε να χειριστείτε τις **τιμές (values)** του λεξικού καλώντας τη μέθοδο **values()**

get()

```
>>> di = {'GR': 'Greece', 'SP': 'Spain', 'IT': 'Italy'}
>>> di.get('GR')
'Greece'
>>> di.get('USA', 'No such key')
'No such key'
```

- Η **get()** επιστρέφει την τιμή (value) ενός κλειδιού (key)
- Αν δεν βρεθεί το κλειδί επιστρέφει 'None' ή κάποιο αλφαριθμητικό που μπορούμε να προκαθορίσουμε (όπως στο παράδειγμα το 'No such key')

```
>>> di['USA']
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    di['USA']
KeyError: 'USA'
```

- Το πλεονέκτημα χρήσης της **get()** είναι ότι δεν προκαλεί σφάλμα KeyError αν το συγκεκριμένο κλειδί δεν υπάρχει

in

```
di = {'GR': 'Greece', 'SP': 'Spain', 'IT': 'Italy'}

if 'FR' in di:
    print('Χώρα: ', di['FR'])
else:
    print('Ανύπαρκτη Χώρα')
```

- Ο τελεστής **in** ελέγχει αν περιλαμβάνεται το κλειδί 'FR' στα κλειδιά του directory di
- Επιστρέφει true / false ανάλογα με το αποτέλεσμα της αναζήτησης

len()

```
>>> di = {'GR': 'Greece', 'SP': 'Spain', 'IT': 'Italy'}  
>>> len(di)  
3
```

- Η **len()** επιστρέφει το **πλήθος των ζευγών key:value** που περιλαμβάνονται στο dictionary

items()

```
>>> di = {'GR':'Greece', 'SP':'Spain', 'IT':'Italy'}
>>> for it in di.items():
    print('Κωδικός: ', it[0], ' Χώρα: ', it[1])
```

```
Κωδικός:  GR  Χώρα:  Greece
Κωδικός:  IT  Χώρα:  Italy
Κωδικός:  SP  Χώρα:  Spain
```

- Η **items()** επιστρέφει λίστα ζευγών key:value (δομή ομάδας-tuple)
- Ο απαριθμητής it είναι δομή ομάδας (tuple) δύο στοιχείων
- Ο κώδικας θα μπορούσε να γραφεί και όπως παρακάτω:

```
for code, country in di.items():
    print('Κωδικός: ', code, ' Χώρα: ', country)
```

for loop & dictionary

```
>>> di = {'GR':'Greece', 'SP':'Spain', 'IT':'Italy'}  
>>> for code in di:  
    print(code)
```

```
IT  
SP  
GR
```

- Ένα λεξικό μπορεί να χρησιμοποιηθεί ως δομή απαρίθμησης σε ένα βρόχο for
- Στην περίπτωση αυτή προσέξτε ότι:
- (α) ο απαριθμητής παίρνει ως τιμές μόνον τα **κλειδιά** του λεξικού
- (β) τα κλειδιά **δεν επιστρέφονται με κάποια καθορισμένη σειρά** – επομένως μην βασίσετε τη λογική του βρόχου επανάληψης σε κάποια υποτιθέμενη σειρά των κλειδιών στο λεξικό

del

```
>>> di = {'GR': 'Greece', 'SP': 'Spain', 'IT': 'Italy'}
>>> del di['IT']
>>> di
{'GR': 'Greece', 'SP': 'Spain'}
```

- Ένα ζεύγος key:value μπορεί να διαγραφεί από το λεξικό με τη χρήση της εντολής **del()** όπως φαίνεται στο παράδειγμα

update()

```
di1 = {'GR': 'Greece', 'SP': 'Spain'}  
di2 = {'SP': 'Espana', 'FR': 'France'}  
  
di1.update(di2)  
print(di1)  
print(di2)
```

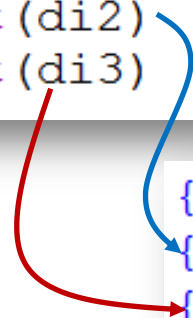
```
{'SP': 'Espana', 'GR': 'Greece', 'FR': 'France'}  
{'SP': 'Espana', 'FR': 'France'}
```

- Η **update()** "ενημερώνει" το 1^ο λεξικό με βάση τις τιμές ενός 2^{ου}
- Για **ίδια** κλειδιά → Αλλάζουν οι τιμές στο 1^ο Λεξικό
- Για **διαφορετικά** κλειδιά → Προστίθενται τα ζεύγη *Κλειδί:Τιμή* στο 1^ο λεξικό

copy()

```
di1 = {'GR': 'Greece', 'SP': 'Spain'}  
di2 = {'SP': 'Espana', 'FR': 'France'}
```

```
di1.update(di2)  
di3 = di2.copy()  
di3['SP'] = 'Spain'  
print(di1)  
print(di2)  
print(di3)
```



```
{'SP': 'Espana', 'FR': 'France', 'GR': 'Greece'}  
{'SP': 'Espana', 'FR': 'France'}  
{'SP': 'Spain', 'FR': 'France'}
```

- Η **copy()** δημιουργεί ένα **νέο αντίγραφο** ενός λεξικού
- Δηλ. δημιουργείται ένα **νέο αντίγραφο στη μνήμη**, άρα κάθε αλλαγή τιμής στο νέο λεξικό **δεν** επηρεάζει το αρχικό
- Στο παράδειγμα η τιμή του κλειδιού 'SP' αλλάζει στο λεξικό di3 αλλά όχι στο αρχικό di2

Δείτε παραδείγματα για τα Λεξικά:

- <http://www.dotnetperls.com/dictionary-python>



Ασκήσεις

1. Πολλά παιχνίδια έχουν πίνακα επιδόσεων παικτών (**'Hall of Fame'**) όπου εμφανίζονται τα ονόματα των παικτών και το σκορ που έχουν πετύχει. Να γραφεί κώδικας που να ζητά το **όνομα του παίκτη** από το πληκτρολόγιο και υπολογίζει με **τυχαίο τρόπο ένα σκορ** (πχ. έναν ακέραιο μεταξύ 500 και 1000). Στη συνέχεια ο κώδικας καταχωρεί το **όνομα παίκτη ως κλειδί και το σκορ ως τιμή** σε ένα λεξικό. Η εισαγωγή δεδομένων συνεχίζεται μέχρι ο χρήστης να δώσει από το πληκτρολόγιο 'q'. Τότε το πρόγραμμα παρουσιάζει το λεξικό με τη μορφή ενός ζεύγους σε κάθε γραμμή και σταματά.
2. Στον προηγούμενο κώδικα γράψτε συνάρτηση `dic_order` η οποία να δέχεται ως όρισμα το λεξικό και να εμφανίζει τα ζεύγη του λεξικού με **αλφαβητική σειρά κλειδιών** (όνομα παίκτη).

Ασκήσεις

3. Γράψτε πρόγραμμα το οποίο να δέχεται από το πληκτρολόγιο: (α) τον κωδικό χώρας(string) , (β) την ονομασία της (string) και (γ) τον πληθυσμό της (int). Στη συνέχεια να καταχωρεί τα στοιχεία αυτά σε λεξικό ως εξής: κλειδί τοποθετεί τον κωδικό χώρας και ως τιμή μια λίστα δύο στοιχείων με πρώτο την ονομασία και δεύτερο τον πληθυσμό της χώρας. Η εισαγωγή δεδομένων σταματά όταν δοθεί ο χαρακτήρας 'q'. Τότε, το πρόγραμμα να εμφανίζει τα στοιχεία του λεξικού σε σειρές ως εξής: στην πρώτη θέση ο κωδικός της χώρας και στη συνέχεια η ονομασία και ο πληθυσμός της. Χρησιμοποιήστε print με μορφοποίηση (μέθοδος format) για τη στοίχιση των δεδομένων.

Tuple

Ομάδα (Πλειάδα)



Tuple (Ομάδα)

Τι είναι

- Μια ομάδα (tuple) είναι μια **ακολουθιακή δομή όπως ακριβώς η λίστα** αλλά με μία σημαντική διαφορά: **δεν είναι μεταλλάξιμη**
- Άρα, **δεν αλλάζει τιμές** παρά μόνον αν δημιουργήσετε ένα νέο αντίγραφο της ομάδας στη μνήμη
 - `T = (1, 2, 3)`
 - `T[2] = 4` *# Error!*
 - `T = T[:2] + (4,)` *# OK: (1, 2, 4)*
- Μια ομάδα χρησιμοποιεί παρενθέσεις αλλά γίνεται αναφορά στις τιμές της με αγκύλες όπως και η λίστα. Πχ.:
 - `T = (0, 'Spam', 2.5)`
 - `print(T[1])`
- Μια ομάδα είναι:
- **Δεικτοδοτημένη** (indexed) , πχ. `T[1]`
- **Αμετάλλακτη** (immutable)
- **Ανομοιογενής** (heterogeneous), πχ. `T = (0, 'Spam', 2.5)`
- **Ταξινομημένη** (ordered)

Tuple

Ιδιότητες

- `()` Κενή ομάδα
- `T = (5,)` Ομάδα με ένα στοιχείο
- `T = (5, 'Spam', 8.3, 12)` Ομάδα με τέσσερα στοιχεία
- `T = 5, 'Spam', 8.3, 12` Παρόμοια (δηλ. δηλώνεται και χωρίς `()`)
- `T = ('Bob', ('dev', 'mgr'))` Φωλιασμένες ομάδες
- `T = tuple('spam')` Η **tuple()** μετατρέπει το αλφαριθμητικό σε ομάδα
- `T[i]` `T[i][j]` Index, index of index
- `T[i:j]` `len(T)` Slice, length
- `T1 + T2` Concatenate, repeat
- `T * 3`
- `for x in T: print(x)` Iteration, membership
- `'spam' in T`
- `[x ** 2 for x in T]` Comprehension

Tuple

Χρήση

- «**Σταθερότητα**»: ως λίστες με «σταθερές» τιμές (immutability)
 - Πχ. περιγραφή των χρωμάτων -μοντέλο RGB- ως τριάδα τιμών
 - BLACK = (0, 0, 0)
 - WHITE = (255, 255, 255)
- «**Κλειδιά**»: ως κλειδιά σε λεξικά (δεν μπορούν να χρησιμοποιηθούν οι λίστες καθώς είναι μεταλλάξιμες)
 - Πχ. Matrix = {(2, 3, 4): 88, (7, 8, 9): 99}

Set

Σύνολο



Set (Σύνολο)

Τι είναι

- Ένα σύνολο (set) είναι –όπως και ένα Λεξικό- μια μη **ταξινομημένη συλλογή μοναδιαίων και αμετάλλακτων αντικειμένων** – κάτι σαν ένα λεξικό μόνον με κλειδιά (χωρίς τιμές).
 - Άρα δεν είναι ούτε δομή ακολουθίας (όπως η λίστα) ούτε δομή αντιστοίχισης (όπως το λεξικό)
- Ένα σύνολο **ακολουθεί τη λογική και τις ιδιότητες των μαθηματικών συνόλων**
 - Πχ. εξ ορισμού ένα στοιχείο εμφανίζεται άπαξ στο σύνολο ανεξάρτητα του πόσες φορές θα προστεθεί σε αυτό.

Set

Ιδιότητες

--1

- Για να δημιουργήσετε ένα set περάστε ως όρισμα μια ακολουθία (ή άλλη απαριθμήσιμη δομή) στη συνάρτηση **set()**

```
>>> a = set('xy159')
>>> b = set('xyzk1250')
>>> a
{'9', 'x', '5', 'y', '1'}
>>> b
{'k', '2', 'x', 'z', '5', 'y', '0', '1'}
```

```
>>> b - a #Διαφορά
{'k', '2', 'z', '0'}
>>>
>>> a | b #Ένωση
{'9', 'k', '2', 'x', 'z', '5', 'y', '0', '1'}
>>>
>>> a & b #Τομή
{'x', '5', 'y', '1'}
>>>
>>> a ^ b #Συμμετρική διαφορά (XOR)
{'9', 'k', '2', 'z', '0'}
>>>
>>> a > b, b > a #Υπερσύνολο, υποσύνολο
(False, False)
```

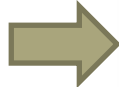
- **Προσθήκη** στοιχείων σε σύνολο: με τη μέθοδο **add()**

```
>>> c = set()      # Ορισμός κενού συνόλου
>>> c.add(15)      # Προσθήκη στοιχείου
>>> c
{15}
```

- **Αφαίρεση** στοιχείων από σύνολο: με τη μέθοδο **remove()**

```
>>> c = set([10, 15])
>>> c
{10, 15}
>>> c.remove(10)
>>> c
{15}
```


- Membership

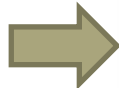


```
>>> a  
{'9', 'x', '5', 'y', '1'}
```

```
>>>
```

```
>>> 'x' in a
```

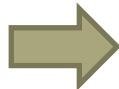
```
True
```



```
>>> x in a
```

```
False
```

- Length



```
>>>
```

```
>>> len(a)
```

```
5
```

- Iteration



```
>>> for i in a: print(i)
```

```
9
```

```
x
```

```
5
```

```
y
```

```
1
```

Set Αμετάλλακτα αντικείμενα

- Ένα set μπορεί να περιέχει μόνον **αμετάλλακτους** (*immutable*) τύπους αντικειμένων.
 - Έτσι, λίστες και λεξικά δεν μπορούν να περιλαμβάνονται σε ένα set, αλλά μια ομάδα (tuple) μπορεί.

```
>>> c
{10, 20}
>>> c.add([30, 40])
Traceback (most recent call last):
  File "<pyshell#110>", line 1, in <module>
    c.add([30, 40])
TypeError: unhashable type: 'list'
>>> c.add((30, 40))
>>> c
{10, 20, (30, 40)}
```

- Η προσπάθεια να προσθέσουμε τη λίστα [30, 40] στο set c οδηγεί σε σφάλμα “unhashable type: ‘list’ “
- Ενώ η προσθήκη της ομάδας (30, 40) γίνεται σωστά

Set Hashable & Unhashable τύπος

- Στην Python ένας τύπος δεδομένου χαρακτηρίζεται “**hashable**” αν κωδικοποιείται με μια κρυφή τιμή “κλειδί” (hash), με βάση την οποία συγκρίνονται τα αντικείμενα αυτού του τύπου για το αν έχουν ίδια τιμή ή διαφορετική
- Έτσι ένας “**hashable**” τύπος μπορεί να χρησιμοποιηθεί ως “μοναδικό κλειδί” σε σύνθετες δομές αντικειμένων, όπως πχ. σε ένα λεξικό
- Γενικά ένας **hashable** τύπος είναι ένας *immutable* τύπος, δηλ. ένας τύπος που δεν αλλάζει τιμή “in place”
 - Όπως **integers**, **string**, **tuple**
- ...ενώ οι μεταλλάξιμοι τύποι (πχ. λίστες, λεξικά, σύνολα) χαρακτηρίζονται ως “**unhashable**”.



Hashable & Unhashable

Παραδείγματα

```
>>> x=1
>>> y=1
>>> id(x);id(y)
493731240
493731240
>>> hash(x);hash(y)
1
1
>>> t1=(10, 20); t2=(10, 20)
>>> id(t1); id(t2)
44044696
44127536
>>> hash(t1); hash(t2)
-95038731
-95038731
>>> l = [1, 2, 3]
>>> hash(l)
Traceback (most recent call last):
  File "<pyshell#169>", line 1, in <module>
    hash(l)
TypeError: unhashable type: 'list'
```

- Δείτε ότι για τους Integers x , y και τα tuples $t1$ & $t2$ η **hash()** επιστρέφει το ίδιο κλειδί
- Οι tuples αναγνωρίζονται ως διαφορετικά αντικείμενα (κωδικός `id()`)
- Αντίθετα για τη λίστα l η κλήση της `hash()` δεν έχει νόημα και επιστρέφει Error

Set Frozenset: Immutable & Hashable

- Τα κλασικό set της Python είναι mutable και επομένως unhashable τύπος
- Αν χρειάζεστε μια “σταθερή” μορφή συνόλου (δηλ. Immutable και hashable) τότε χρησιμοποιήστε την frozenset() για να μετατρέψετε ένα set στη μορφή αυτή

```
>>> s = set([1,2,3,4])
```

```
>>> s
```

```
{1, 2, 3, 4}
```

```
>>> fs = frozenset(s)
```

```
>>> fs
```

```
frozenset({1, 2, 3, 4})
```

```
>>> s.add(10)
```

```
>>> fs.add(10)
```

```
Traceback (most recent call last):
```

```
File "<pyshell#202>", line 1, in <module>
```

```
    fs.add(10)
```

```
AttributeError: 'frozenset' object has no attribute 'add'
```

- Δεν μπορείτε να προσθέσετε νέα στοιχεία σε ένα frozenset

Set

Χρήση

- Μια ενδιαφέρουσα χρήση των sets είναι για το φιλτράρισμα των διπλών/πολλαπλών στοιχείων σε μια σύνθετη δομή, όπως παρακάτω:
- Μετατρέψτε τη δομή σε set
 - (οπότε τα διπλά/πολλαπλά στοιχεία εμφανίζονται μόνον μία φορά)
- Μετατρέψτε το set πάλι στον αρχικό τύπο

```
>>> L = [1, 2, 1, 3, 1, 4, 5, 2, 6, 3]
>>> L = list(set(L))
>>> L
[1, 2, 3, 4, 5, 6]
```

Σύνοψη Χαρακτηριστικά Τύπων

- **Lists**, **Dictionaries**, **Sets** : Μεταλλάξιμοι (mutable)
 - **Tuples**, **Frozensets**: Μη μεταλλάξιμοι (immutable)
-
- **Tuples**, **Frozensets**, (και *Integers*, *Strings*): hashable
 - **Lists**, **Dictionaries**, **Sets** : unhashable
-
- **Lists**, **Dictionaries**, **Tuples**: Περιέχουν κάθε τύπο δεδομένων
 - **Sets**: Περιέχουν κάθε μη μεταλλάξιμο (immutable) τύπο
 - **Lists**, **Dictionaries**, **Tuples**: μπορούν να δημιουργήσουν φωλιασμένες δομές
 - **Lists**, **Dictionaries**, **Sets**: Μεταβάλλονται δυναμικά (“in place”)

