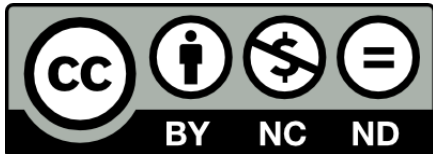


# Εισαγωγή στην **Python**

**7**





### *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
- Πλειάδα (Tuple)

# 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'
```

```
>>> 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**



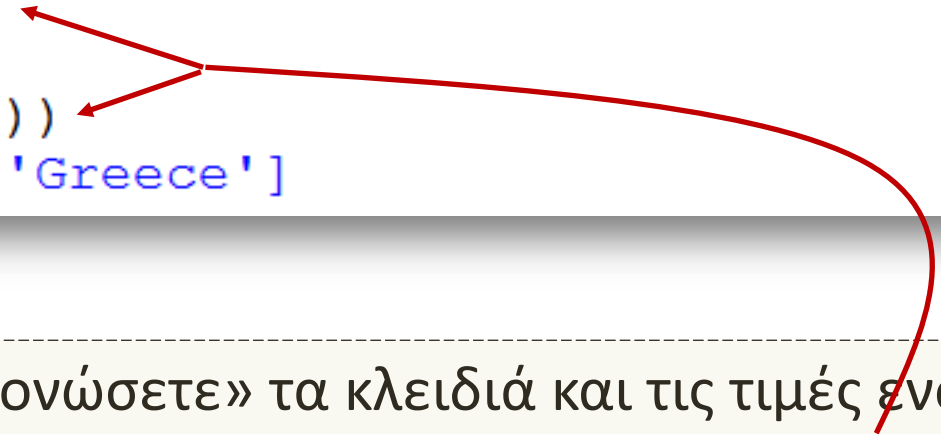
# Dictionary

## Ζεύγη

key:value

-3

```
>>> 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 τρία 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

## (γ) Λίστα με στοιχεία **λεξικά**

```
import pprint
country={}
country['GR'] = {'Name':'Greece',
                 'Capital':'Athens',
                 'Pop':11}
country['IT'] = {'Name':'Italy',
                 'Capital':'Rome',
                 'Pop':61}
country['ES'] = {'Name':'Spain',
                 'Capital':'Madrid',
                 'Pop':46}

# Construct a list with dictionary items
Europe=[country[k] for k in country]
pprint.pprint(Europe)

# Indexing:[Integer][Key], for example:
print(Europe[0]['Capital'])

# Print country name and capital city
for c in Europe:
    print('Country: ',c['Name'],'\tCapital: ',c['Capital'])
```

# 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 = {}`

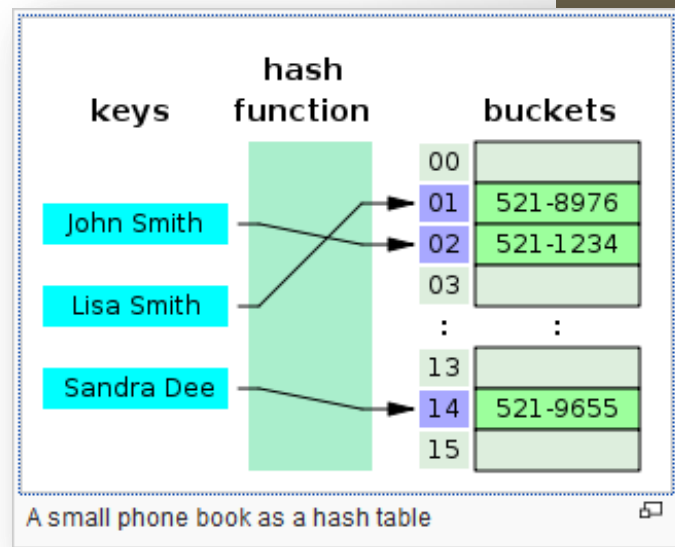
`D[99] = 'spam'`

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

# Dictionary

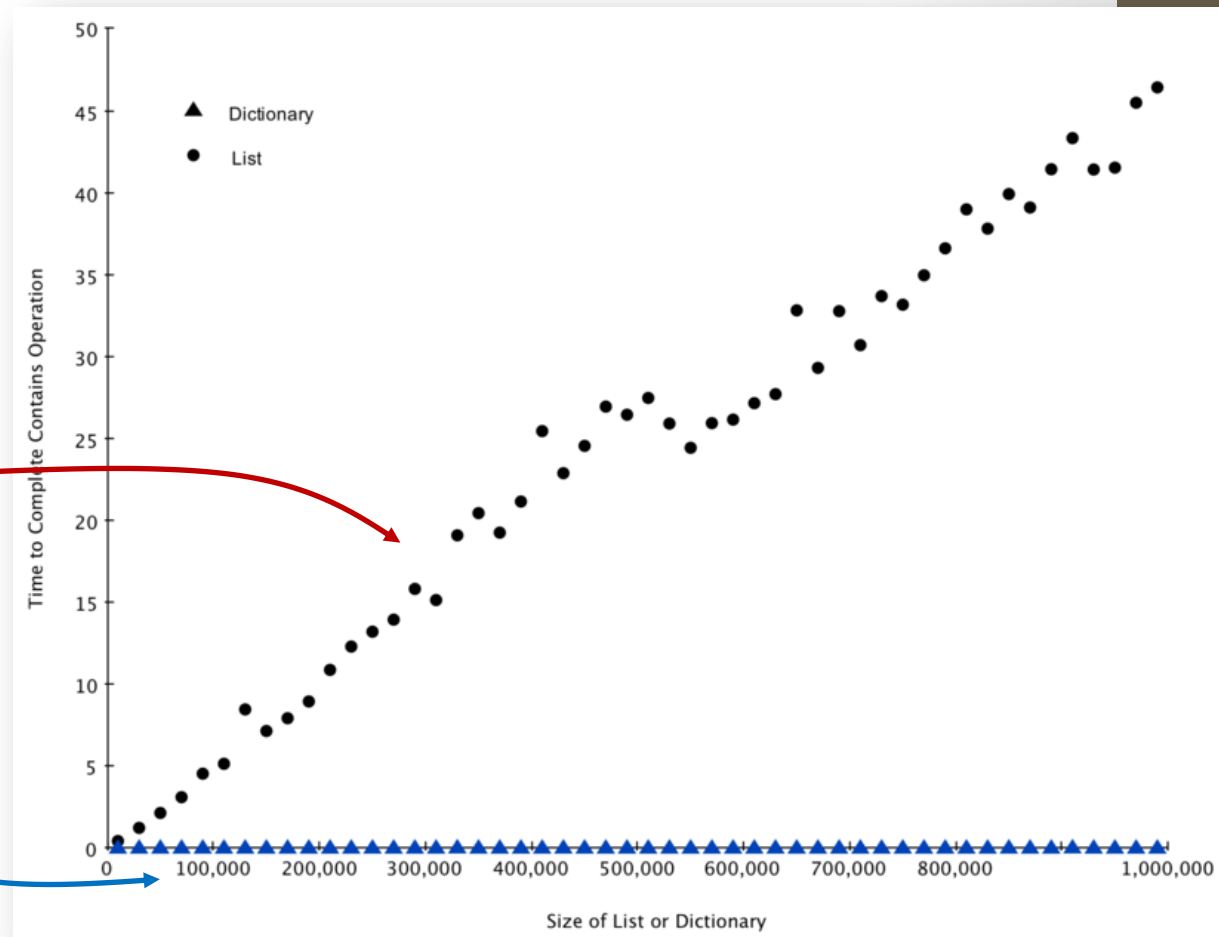
## Πίνακας κερματισμού

- 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 dictionary 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 πλειάδας-tuple)
- Ο απαριθμητής it είναι δομή πλειάδας (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)
```

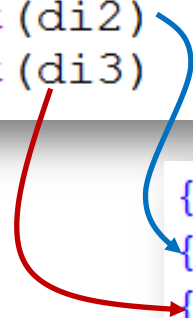
```
{'SP': 'Espana', 'GR': 'Greece', 'FR': 'France'}  
{'SP': 'Espana', 'FR': 'France'}
```

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

# Tuple

# Πλειάδα





# Tuple (Πλειάδα) Τι είναι

- Μια **πλειάδα** (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 πλειάδας στη μνήμη
  - `T = (1, 2, 3)`
  - `T[2] = 4` *# Error!*
  - `T = T[:2] + (4,)` *# OK: (1, 2, 4)*
- Μια πλειάδα χρησιμοποιεί παρενθέσεις αλλά γίνεται αναφορά στις τιμές της με αγκύλες όπως και η λίστα. Πχ.:
  - `T = (0, 'Spam', 2.5)`
  - `print(T[1])`
- Μια πλειάδα είναι:
- **Δεικτοδοτημένη** (indexed) , πχ. `T[1]`
- **Αμετάλλακτη** (immutable)
- **Ανομοιογενής** (heterogeneous), πχ. `T = (0, 'Spam', 2.5)`
- **Ταξινομημένη** (ordered)

# Tuple

## Ιδιότητες

- `()`      Κενή πλειάδα
- `T = (5,)`      Πλειάδα με ένα στοιχείο
- `T = (5, 'Spam', 8.3, 12)`      Πλειάδα με τέσσερα στοιχεία
- `T = 5, 'Spam', 8.3, 12`      Παρόμοια (δηλ. δηλώνεται και χωρίς `()`)
- `T = ('Bob', ('dev', 'mgr'))`      Φωλιασμένες πλειάδες
- `T = tuple('spam')`      Η **tuple()** μετατρέπει το αλφαριθμητικό σε πλειάδα
- `T[i]`    `T[i][j]`      Δεικτοδότηση
- `T[i:j]`    `len(T)`      Τομή, Μήκος πλειάδας
- `T1 + T2`      Συνένωση (Concatenate)
- `T * 3`      Πολ/σμός με ακέραιο (repeat)
- `for x in T: print(x)`      Επαναληπτική δομή,
- `'spam' in T`      'Ανήκειν' (membership)
- `[x ** 2 for x in T]`      Περιγραφή Πλειάδας (Comprehension, συμπερίληψη)

# 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}