ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Η/Υ
σε Python

Διάλεξη 6: Λεξικά και Σύνολα

Μάρκος Ζάμπογλου, mzampoglou@aegean.gr

Πριν αρχίσουμε: η συνάρτηση `zip()`

- Η συνάρτηση **`zip()`** δέχεται ως είσοδο έναν αριθμό **`n`** από iterators
- Επιστρέφει ένα αντικείμενο τύπου **`zip`**
  - Το οποίο περιέχει ομαδοποιημένα σε tuples όλα τα πρώτα αντικείμενα των iterators μαζί, έπειτα τα δεύτερα αντικείμενα μαζί κλπ.
  - Άρα σε κάθε θέση περιέχει **`n`** αντικείμενα

```
a = [1, -2, 4, 8, 3]
b = [True, False, True, True, False]
c = ("a", "z", "f", "d", "i")
z = zip(a,b,c)
print(type(z))
<class 'zip'>
print(list(z))
[(1, True, 'a'), (-2, False, 'z'), (4, True, 'f'), (8, True, 'd'),
 (3, False, 'i')]
```

Χρήση της `zip`

- Η βασική λειτουργία της κλάσης **`zip`** είναι πως επιτρέπει παράλληλη σάρωση αντικειμένων

Π.χ.:

```
a = [1, -2, 4, 8, 3]
b = [True, False, True, True, False]
c = ("a", "z", "f", "d", "i")

for t in zip(a, b, c):
  if t[1]==True:
    print(str(t[0])+t[2])
```

```
1a
4f
8d
```

Λεξικά (Dictionaries)

- Μια **iterable** και **mutable** δομή της Python
- Αποτελείται από **ζεύγη κλειδιών:τιμών** (***key:value pairs***)
- Κλάση: **`dict`**

```
my_dict = {'Name':'Markos',
           "Surname":"Zampoglou",
           "Weight":82, "Height":1.94,
           "Interests":["Machine Learning",
                        "Python"]}
print(my_dict['Name'])
Markos
print(type(my_dict))
<class 'dict'>
```

Indexing όχι με τη θέση αλλά με το κλειδί!

Δημιουργία και γέμισμα

Άδειο λεξικό:

```
car = {}
e = dict()
```

Γεμισμένο λεξικό:

```
person = {"a" : 2,
          (3,1) : False,
          True : [3,1,3]
          2: "Hello"}
```

Με τη χρήση των `dict()` και `zip()`

```
a=["a","b","c","d"]
b=[3,4,5,6,7]
z=dict(zip(a,b))
```

Το πρώτο αντικείμενο δίνει τα κλειδιά και το δεύτερο τις τιμές

- Οι τιμές μπορεί να είναι **οποιουδήποτε** τύπου, τα κλειδιά είναι **οποιουδήποτε immutable*** τύπου (sting, number, boolean, tuple)
- Απαγορεύονται διπλές εμφανίσεις του ίδιου κλειδιού

Προσθήκη νέου ζεύγους κλειδιού/τιμής σε λεξικό

```
car["Number Of Doors"] = 4
```

- Αν το κλειδί δεν υπάρχει, δημιουργείται και του ανατίθεται η τιμή
- Αν υπάρχει, αντικαθίσταται η τιμή

🤓 *στην πραγματικότητα δεν απαιτείται να είναι απλώς *immutable* αλλά και *hashable* -αλλά δε θα μπούμε σε αυτό το βάθος

Δημιουργία λεξικού από iterables

- Η **`dict()`** μπορεί να δημιουργήσει dictionaries από οποιονδήποτε συνδυασμό iterable από iterables
- π.χ., όλες οι παρακάτω γραμμές δημιουργούν το ίδιο λεξικό: {'b': 4, 'c': 6, 'd': 9}:

```
dict([('b', 4), ('c', 6), ('d', 9)]) #Λίστα από tuples
dict([['b', 4], ['c', 6], ['d', 9]]) #Λίστα από λίστες
dict((('b', 4), ('c', 6), ('d', 9))) #Tuple από tuples
dict((['b', 4], ['c', 6], ['d', 9])) #Tuple από λίστες
```

Τι θα εμφανίσει;

```
my_dict = {'Name':'Markos', "Surname":"Zampoglou",
        "Weight":82, "Height":1.94,
        "Interests":["Machine Learning",
                     "Python"]}
my_dict2 = my_dict
my_dict2['Weight'] = 84
print(my_dict['Weight'])
```

```
84
```

Γιατί;

- Γιατί η εντολή `my_dict2['Weight'] = 84` τροποποιεί μόνο μέρος του αντικειμένου `my_dict2` χωρίς να αλλάξει το συνολικό reference, οπότε το αντικείμενο `my_dict` συνεχίζει να δείχνει στο ίδιο (τροποποιημένο) λεξικό
  - *Η μερική τροποποίηση επιτρέπεται επειδή τα dicts είναι mutable*

Γιατί λεξικά;

- Τα λεξικά προσφέρουν γρήγορη αναζήτηση τιμών με βάση το κλειδί
  - ...και αργή αναζήτηση κλειδιών με βάση την τιμή
- Προσφέρουν επίσης γρήγορη προσθήκη-αφαίρεση ζευγών
- Προγραμματιστικά, μπορούμε να τις αντιμετωπίζουμε σαν λίστες με ονοματισμένες αντί για αριθμημένες θέσεις
  - Ανοίγουν πολύ βολικές δυνατότητες

Λεξικό αντί δομών επιλογής

Λύση με δομή επιλογής:

```
vehtype = input("Δώσε τον τύπο του οχήματος: ")
if vehtype=='Δίκυκλο':
  poso=30
elif vehtype=='ΙΧ':
  poso=50
elif vehtype=='Ταξί': #κλπ, όσες επιλογές χρειαζόμαστε
  poso = 60
print(f"Η χρέωση ΚΤΕΟ είναι {poso} ευρώ.")
```

Λύση με λεξικό:

```
posa = {'Δίκυκλο':30, 'ΙΧ':50, 'Ταξί':60} #κλπ, όσες επιλογές χρειαζόμαστε
vehtype = input("Δώσε τον τύπο του οχήματος: ")
print(f"Η χρέωση ΚΤΕΟ είναι {posa[vehtype]} ευρώ.")
```

Παράδειγμα 6.1

Να γραφεί κώδικας που θα δέχεται έναν αριθμό από το 1 ως το 12 και θα επιστρέφει τον αντίστοιχο μήνα. Η λύση να υλοποιηθεί με λεξικό και όχι **`if...elif...else`**

Παράδειγμα 6.1

Να γραφεί κώδικας που θα δέχεται έναν αριθμό από το 1 ως το 12 και θα επιστρέφει τον αντίστοιχο μήνα. Η λύση να υλοποιηθεί με λεξικό και όχι **if...elif...else**

```
months = {1:"Ιανουάριος",
   2:"Φεβρουάριος", 3:"Μάρτιος",
   4:"Απρίλιος", 5:"Μάιος",
   6:"Ιούνιος", 7:"Ιούλιος",
   8:"Αύγουστος", 9:"Σεπτέμβριος",
   10:"Οκτώβριος", 11:"Νοέμβριος",
   12:"Δεκέμβριος"}
num = int(input("Δώσε αριθμό μήνα: "))
print(months[num])
```

Λειτουργίες λεξικού

- Διαφορές και ομοιότητες με λίστες:
  - ✓ Indexing (*με το key, όχι με τη θέση*)
  - ✓ **`in`** ως λογική πράξη (π.χ. σε **`if`** ή **`while`**): *αναζήτηση στα keys, όχι στα values*
  - ✓ **`in`** για επανάληψη με **`for`**: *αναζήτηση στα keys, όχι στα values*
  - ✓ Συναρτήσεις (**`len()`**, **`min()`**, **`max()`**, **`sorted()`**): *υπό προϋποθέσεις*
  - ✓ Μεθόδους που επιστρέφουν τιμές
  - ✓ Απευθείας τροποποίηση τιμής: `a['speed']=4.2`
  - ✓ Μεθόδους που εφαρμόζονται in-place (**`dict.update()`**)
  - ✓ **`del`** για συγκεκριμένα στοιχεία: `del d[3]`
    - ✓ και φυσικά συνολικό `del a`
  - ✗ slicing (*αφού δεν υπάρχει αρίθμηση και σειρά*)
  - ✗ Πρόσθεση/συνένωση (**`+`**), πολλαπλασιασμός (**`*`**)

Η εντολή in σε λεξικά

```
my_dict={"price":15, "size":"M", "in stock": True}
```

Ως λογική πράξη, αναζητά στα keys:

```
print("price" in my_dict)
True
print("color" in my_dict)
False
```

Ως iteration, επιστρέφει τις τιμές των keys:

```
for a in my_dict:
  print(a)
price
size
in stock
```

Σάρωση λεξικού

```
for key in my_dict:
  print(f'key: {key},\t value:
   {my_dict[key]}')
```

```
key: price,         value: 15
key: size,          value: M
key: in stock,      value: True
```

Η **for** μου φέρνει ένα-ένα όλα τα κλειδιά, τα οποία μπορώ να χρησιμοποιήσω για να πάρω (ή να τροποποιήσω) τις αντίστοιχες τιμές.

Συναρτήσεις για λεξικά

Οι βασικές συναρτήσεις εκτελούνται στην πραγματικότητα πάνω στο σύνολο των κλειδιών:

- **`len(X)`**: επιστρέφει το πλήθος των κλειδιών
  - δηλαδή των στοιχείων
- **`sum(X)`**: επιστρέφει το άθροισμα των κλειδιών
  - αν είναι όλα αριθμήτικά
- **`min(X)`**, **`max(X)`**: επιστρέφει το μεγαλύτερο ή το μικρότερο από τα κλειδιά
  - αν ανήκουν όλα σε συγκρίσιμους τύπους
- **`sorted(X)`**: επιστρέφει μια λίστα που περιέχει τα κλειδιά ταξινομημένα με αύξουσα σειρά
  - αν ανήκουν όλα σε συγκρίσιμους τύπους

Μέθοδοι λεξικών 1/3

Μέθοδοι που επιστρέφουν τα περιεχόμενα του λεξικού:

```
# εξαγωγή κλειδιών/τιμών
print(my_dict.keys())
dict_keys(['price', 'size', 'in stock'])
print(my_dict.values())
dict_values([15, 'M', True])
prin(my_dict.items())
dict_items([('price', 15), ('size', 'M'),
            ('in stock', True)])
```

Μπορούμε να εξάγουμε τα περιεχόμενά τους π.χ. με τη `list()`

Η καθεμιά επιστρέφει διαφορετικό τύπο αντικειμένου!

Τιμές

Τιμές

Tuples με ζεύγη τιμών

Μέθοδοι λεξικών 2/3

- **`dict.copy()`**: Δημιουργεί ένα αντίγραφο του λεξικού στη μνήμη και το επιστρέφει, για να μπορούμε να κάνουμε αλλαγές χωρίς να επηρεάζουμε το πρωτότυπο

```
b=my_dict.copy()
```

- **`dict.update(X)`**: Δέχεται ως παράμετρο ένα iterable **`X`** που περιέχει iterables με ζεύγη τιμών, και τα περνάει ως νέα κλειδιά:τιμές στο λεξικό. Αν κάποιο κλειδί υπάρχει ήδη αντικαθιστά την τιμή, ενώ αν δεν υπάρχει το δημιουργεί

```
a={'value1':1, 'value2':2}
a.update([['value2',2.5],['value3',3]] )
print(a)
{'value1': 1, 'value2': 2.5, 'value3': 3}
```

Αντί για iterable με iterables, μπορούμε να δώσουμε κατευθείαν ένα λεξικό:

```
a.update({'value1':-1.5, 'newValue4':4})
print(a)
{'value1': -1.5, 'value2': 2.5, 'value3': 3, 'newValue4': 4}
```

Μέθοδοι λεξικών 3/3

- **dict.pop(key)**: Επιστρέφει την τιμή που αντιστοιχεί στο κλειδί key, και διαγράφει το ζεύγος κλειδί:τιμή από το λεξικό. Αν το κλειδί δεν υπάρχει προκαλεί Error

```
a={"one":1, "two":2}
print(a.pop("one"))
1
print(a)
{'two': 2}
```

  - **dict.pop(key, value)**: Το ίδιο, όμως αν το κλειδί δεν υπάρχει επιστρέφει την τιμή **value** και δε διαγράφει τίποτα

- **dict.popitem()**: Επιστρέφει ένα ζεύγος τιμή:κλειδί από το λεξικό με τη μορφή ενός tuple δύο τιμών, και το διαγράφει από το λεξικό. Το αντικείμενο το επιλέγει η Python.

```
a={"one":1, "two":2, "three":3}
print(a.popitem())
('three', 3)
print(a)
{'one': 1, 'two': 2}
```

Παράδειγμα 6.2

Δίνονται δύο λεξικά που περιέχουν τις πωλήσεις δυο υποκαταστημάτων για κάθε χρονιά που έχουν λειτουργήσει ως τώρα. Να δημιουργηθεί ένα τρίτο λεξικό που θα περιέχει τις αθροιστικές πωλήσεις της επιχείρησης για όλες τις χρονιές.

Παράδειγμα 6.2

Δίνονται δύο λεξικά που περιέχουν τις πωλήσεις δυο υποκαταστημάτων για κάθε χρονιά που έχουν λειτουργήσει ως τώρα. Να δημιουργηθεί ένα τρίτο λεξικό που θα περιέχει τις αθροιστικές πωλήσεις της επιχείρησης για όλες τις χρονιές.

```
sales1 = {…}
sales2 = {…}

tot_sales = sales1.copy()
for y in sales2:
  if y not in tot_sales:
    tot_sales[y]=sales2[y]
  else:
    totl_sales[y]+=sales2[y]
print(tot_sales)
```

Παράδειγμα 6.3

Να γραφεί κώδικας που να δέχεται ένα string και θα δημιουργεί ένα λεξικό που θα καταγράφει το πλήθος των εμφανίσεων κάθε χαρακτήρα.

Παράδειγμα 6.3

Να γραφεί κώδικας που να δέχεται ένα string και θα δημιουργεί ένα λεξικό που θα καταγράφει το πλήθος των εμφανίσεων κάθε χαρακτήρα.

```
st = '…'
counts={}
for ch in st:
  if ch not in counts:
    counts[ch]=1
  else:
    counts[ch]+=1
print(counts)
```

Dictionary comprehension

- Όπως και οι λίστες, έτσι και τα λεξικά είναι mutable
  - Αυτό σημαίνει ότι μπορώ να δημιουργήσω λεξικά με comprehension ("*σύνθεση*")
- Σύνταξη:
  `{key:value for a in iterable if condition}`
- Προφανώς το **`key`** και το **`value`** πρέπει να προκύπτουν από το **`a`** με κάποιο τρόπο, ώστε να παίρνουν τιμή σε κάθε επανάληψη
  - Συχνά το **`a`** είναι σύνθετο αντικείμενο
- Αντίστοιχα και το **`condition`** πρέπει να είναι μια συνθήκη που περιλαμβάνει το **`a`** (ή μέρος του)

Dictionary comprehension

Από μία λίστα:

```
a=[3,5,6,2,3]
keyvalues = {str(a.index(item)):item for item in a}
{'0': 3, '1': 5, '2': 6, '3': 2}
```

Από δύο λίστες με **zip()**:

```
a=['a', 'b', 'c', 'd', 'f']
b=[2, 7, 5, 1, 3]
d3 = {i:j for (i,j) in zip(a,b) if j>2}
{'b': 7, 'c': 5, 'f': 3}
```

Παράδειγμα 6.4

Δίνεται μια λίστα `L`
που περιέχει λέξεις.
Να γραφεί κώδικας
που επιστρέφει ένα
λεξικό που θα περιέχει
ως κλειδιά όσες από
τις λέξεις αυτές
ξεκινάνε με κεφαλαίο
γράμμα, και ως τιμές
το μήκος τους.

Παράδειγμα 6.4

Δίνεται μια λίστα **L** που περιέχει λέξεις. Να γραφεί κώδικας που επιστρέφει ένα λεξικό που θα περιέχει ως κλειδιά όσες από τις λέξεις αυτές ξεκινάνε με κεφαλαίο γράμμα, και ως τιμές το μήκος τους.

```
L=["βροντάει", "ο",
"Όλυμπος", "αστράφτει", "η",
"Γκιώνα", "μουγκρίζουν",
"τ'", "Άγραφα", "σειέται",
"η", "στεριά"]


D = {w:len(w) for w in L if
w[0].isupper()}
print(D)
```

Σύνολο

Μια mutable συλλογή από μοναδικά, αμετάβλητα* στοιχεία, χωρίς διάταξη

Δημιουργία με το σύμβολο **{}**, όπως τα λεξικά

```
set1 = {4,7.2,'Hello',True}
```

Δημιουργία από άλλο σύνθετο αντικείμενο με τη συνάρτηση **set()**

```
set2 = set([5,6,2,6])
set3 = set((5,6,2,6))
set4 = set(range(3,20,4))
```

🤪 *και πάλι, όχι απλώς *immutable* αλλά και *hashable* -δεν πειράζει.

Συλλογή χωρίς διάταξη

Το γεγονός ότι τα περιεχόμενα ενός set δεν έχουν διάταξη σημαίνει ότι δε μπορώ να χρησιμοποιήσω indexing και slicing για να τα πάρω.

Μπορώ όμως να χρησιμοποιήσω την **in**:

```
s={3,5,1,9}
```

- Ως λογική πράξη:

```
print(5 in s)
True
```

- Σε δομή επανάληψης:

```
for i in s:
  print(i, end=" ")
1 3 5 9
```

Τη σειρά που παίρνω τα στοιχεία την αποφασίζει η Python

Τροποποίηση set

- Τα σύνολα δεν έχουν indexing, είναι όμως **mutable**
  - Διαθέτουν μεθόδους in-place για την τροποποίησή τους

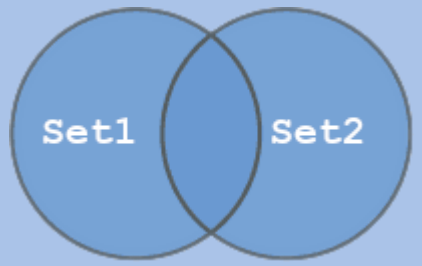
- **`set.add(x)`**: Προσθέτει το στοιχείο **`x`** στο σύνολο
- **`set.update(x)`**: Προσθέτει τα στοιχεία του iterable **`x`** στο σύνολο
  - **Προσοχή**: Η **`s.update((2,4))`** θα προσθέσει το **`2`** και το **`4`** στο **`s`**, ενώ η **`s.add((2,4))`** θα προσθέσει το **`(2,4)`** στο **`s`**.
- **`set.discard(x)`**: Αφαιρεί το στοιχείο **`x`** από το σύνολο, αν υπάρχει
- **`set.pop()`**: *Επιστρέφει* ένα στοιχείο (όποιο επιλέξει η Python) από το σύνολο και ταυτόχρονα το αφαιρεί

Πράξεις συνόλων

Ένωση

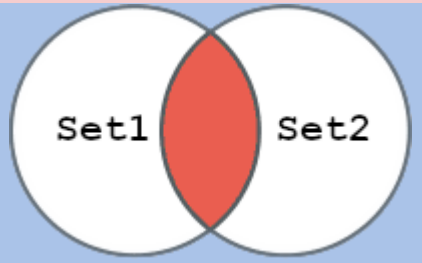
```
set1 | set2
set1.union(set2)
```

Οι πράξεις γίνονται είτε με τελεστή, είτε με μια μέθοδο συνόλων



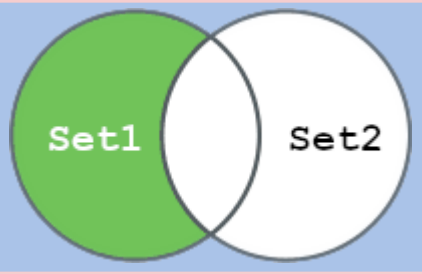
Τομή

```
set4 = set1 & set2
set4 = set1.intersection(set2)
```



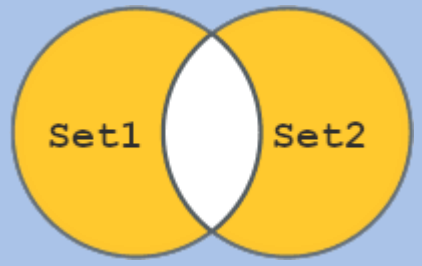
Αφαίρεση

```
set5 = set1 - set2
set5 = set1.difference(set2)
```



Συμμετρική αφαίρεση

```
set7 = set1 ^ set2
set7 = set1.symmetric_difference(set2)
```



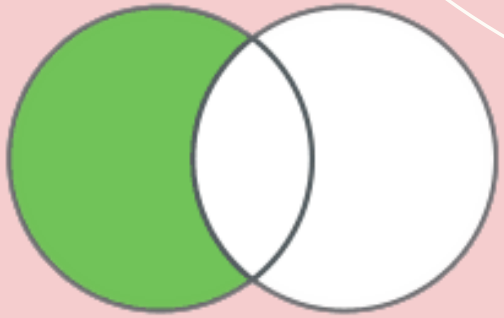
Frozen sets

- Immutable εκδοχή των sets
- Διαθέτουν όλες τις λειτουργίες των συνόλων εκτός από αυτές που το τροποποιούν (δηλαδή τις μεθόδους in-place)

```
set1_fr = frozenset({3, 6, 8})
set2_fr = frozenset({8,1})
print(set1_fr.intersection(set2_fr))
frozenset({8})
print(type(set1_fr))
<class 'frozenset'>
```

Γιατί σύνολα;

- Το ένα χρήσιμο χαρακτηριστικό των συνόλων είναι η μοναδικότητα των στοιχείων τους
  - Π.χ. μπορώ να επιλέξω τα μοναδικά στοιχεία μιας λίστας με το να τη μετατρέψω σε σύνολο
- Το άλλο χρήσιμο χαρακτηριστικό τους είναι η μεγάλη ταχύτητα αναζήτησης με την **`in`**
  - Αν θέλω να ελέγξω αν μια τιμή βρίσκεται σε μια συλλογή, η ιδανική δομή για αυτό το σκοπό είναι ένα **`set`**
- Η τρίτη χρήσιμη λειτουργία που μας παρέχει είναι οι πράξεις συνόλων



Παράδειγμα 6.4

Δίνεται μια λίστα `L`. Να γραφεί κώδικας που θα δημιουργεί μια νέα λίστα `L_u` που θα περιέχει μόνο τα μοναδικά στοιχεία της `L`

Παράδειγμα 6.4

Δίνεται μια λίστα **L**. Να γραφεί κώδικας που θα δημιουργεί μια νέα λίστα **L_u** που θα περιέχει μόνο τα μοναδικά στοιχεία της **L**

```
L=[3,5,12,1,3,11,12,5]

L_u=list(set(L))
print(L_u)
```

Παράδειγμα 6.5

Να λυθεί ξανά το παράδειγμα 6.4, αλλά με τη χρήση dictionary comprehension.

Παράδειγμα 6.5

Να λυθεί ξανά το παράδειγμα 6.3, αλλά με τη χρήση dictionary comprehension.

```
st = '…'
counts={}
counts={ch:st.count(ch) for ch in set(st)}
print(counts)
```