Ελληνικό Μεσογειακό Πανεπιστήμιο
Σχολή Μηχανικών / Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

HMY01K06
Επιστημονικός Προγραμματισμός με Python

Διάλεξη Έκτη
Δομημένοι Τύποι Δεδομένων (συνέχεια)
Σύνολα, Λεξικά
Διαχείριση Σφαλμάτων

Φθινόπωρο 2025

Ο Τύπος δεδομένων set (σύνολο)

Δεν διαφέρει πολύ από την έννοια του συνόλου στα Μαθηματικά

Τι είναι: Συλλογή από *μοναδικά, αμετάβλητα* στοιχεία *χωρίς διάταξη*

- *Μοναδικά*: δεν επιτρέπονται επαναλήψεις του ίδιου στοιχείου

  ενώ πχ οι λίστες, οι πλειάδες και οι συμβολοσειρές μπορούν να περιέχουν ίδια στοιχεία

- *Αμετάβλητα*: πχ αριθμοί, συμβολοσειρές, πλειάδες …

  ένα σύνολο δεν μπορεί να περιέχει σαν στοιχεία του λίστες, λεξικά ή άλλα σύνολα

  μπορεί όμως να περιέχει μια ειδική κατηγορία συνόλων τα λεγόμενα ***παγιωμένα σύνολα (frozensets)*** που θα δούμε παρακάτω

- *Χωρίς διάταξη*: δεν υπάρχει πρώτο, δεύτερο … τελευταίο στοιχείο

  όλα τα στοιχεία 'χύμα στο ίδιο τσουβάλι'

Παρόλο που ένα σύνολο δεν μπορεί να περιέχει μεταβλητά αντικείμενα, το ίδιο το *σύνολο* είναι *μεταβλητός τύπος δεδομένων*

Που *(και γιατί)* χρησιμοποιούνται τα σύνολα

Χρησιμοποιούνται για να αποθηκεύσουμε *αμετάβλητα* δεδομένα όταν *δεν μας ενδιαφέρει η διάταξή τους*, ούτε υπάρχουν *επαναλήψεις*

Υλοποιούνται στην Python με τέτοιο τρόπο, ώστε η αναζήτηση των στοιχείων τους να γίνεται *πάρα πολύ γρήγορα* σε σχέση με τις ακολουθιακές δομές *(λίστες κλπ.)*

Παρόλο που τόσο οι *λίστες* όσο και οι *πλειάδες* υπερτερούν σε *ευελιξία*, *(δηλαδή μας παρέχουν περισσότερες δυνατότητες χειρισμού)* το βασικό *πλεονέκτημα των συνόλων* είναι η *ταχύτητα*

Πρακτικά χρησιμοποιούνται για:

- τον *έλεγχο της ιδιότητας μέλους* (`x in S`)πολύ *πιο γρήγορα από άλλες δομές*, ιδιαίτερα όταν το πλήθος των στοιχείων είναι *μεγάλο*
- την *εξάλειψη πολλαπλών αντιγράφων στοιχείων* από μια λίστα ή άλλη ακολουθιακή δομή
- τον *υπολογισμό μαθηματικών συναρτήσεων* που ορίζονται πάνω σε σύνολα όπως η *τομή*, η *ένωση*, η *διαφορά* και η *συμμετρική διαφορά*

Πως δημιουργούμε ένα σύνολο

- ***Πρώτος Τρόπος:*** με αναγραφή των στοιχείων του μέσα σε άγκιστρα { }

```
fruits = { 'apple', 'banana', 'cherry' }
x = {1, 2, 3}
y = {'μπλε', 'κόκκινο', 127, (12.4, 'a')}
```

Προσοχή!
z=set() ορίζει ένα κενό **σύνολο**
όμως z={} ορίζει ένα κενό **λεξικό**

- ***Δεύτερος Τρόπος:*** με την συνάρτηση **set()**

Η συνάρτηση set(*<απαριθμήσιμη ακολουθία αντικειμένων>*)δημιουργεί ένα σύνολο με στοιχεία τους όρους της απαριθμήσιμης ακολουθίας που δίνεται σαν όρισμα

```
x = set([ 1, 2, 3, 'go'])   #Δημιουργία συνόλου από λίστα (ή πλειάδα)
print(x)  ⇨  {3, 'go' 1, 2}

x = set(range(2,5)) #Δημιουργία συνόλου από την γεννητορική  συνάρτηση range
print(x)  ⇨  {4, 2, 3}

x = set('test') #Δημιουργία συνόλου από συμβολοσειρά
print(x)  ⇨  {'e', 't', 's'}  ← Παρατηρείστε ότι το 't' εμφανίζεται μόνο μια φορά
```

Πως διατρέχουμε ή εμφανίζουμε όλα τα στοιχεία ενός συνόλου

Εφόσον δεν υπάρχουν δείκτες, δεν γνωρίζουμε την ακριβή θέση κάθε στοιχείου

Με την εντολή **for** μπορούμε να *διατρέξουμε* όλα τα στοιχεία ενός συνόλου είτε για να τα *χρησιμοποιήσουμε* στους υπολογισμούς μας, είτε απλά για να τα *εμφανίσουμε*

```
fruits = {'apple', 'banana', 'cherry'}
for x in  fruits :
    print(x, end=' ')
'banana' 'cherry' 'apple'
```

Αν θέλουμε μόνο να εμφανίσουμε τα στοιχεία ενός συνόλου, αυτό μπορεί να γίνει και με σκέτο `print`

```
print(fruits)  ⇨ {'banana', 'cherry', 'apple'}
print(fruits)  ⇨ {'cherry', 'banana', 'apple'}
```

Επειδή στα σύνολα δεν υπάρχει διάταξη, η *σειρά εμφάνισης των στοιχείων* τους είναι κάθε φορά ***τυχαία***

δεν υπάρχει πρώτο, δεύτερο ,… τελευταίο στοιχείο *(όλα χύμα στο ίδιο τσουβάλι)*

Πράξεις και μέθοδοι πάνω στα σύνολα

Πρόσβαση στοιχείων συνόλου *(το είδαμε και προηγουμένως)*

Δεν υπάρχουν δείκτες, δεν γνωρίζουμε την ακριβή θέση κάθε στοιχείου.

Όμως μπορούμε να απαριθμήσουμε όλα τα στοιχεία ενός συνόλου με το `for`

```
fruits = { 'apple', 'banana', 'cherry' }

  for x in  fruits :
    print(x, end=' ')
'banana' 'cherry' 'apple'
```

Υπενθυμίζεται ότι η πράξη `in` (*δηλαδή ο έλεγχος αν ένα στοιχείο ανήκει ή όχι σε μια ομάδα αντικειμένων*) γίνεται *πολύ πιο γρήγορα* και *αποδοτικά* στα σύνολα απ' ότι σε άλλους τύπους της Python (λίστες, πλειάδες, συμβολοσειρές)

Έλεγχος αν ένα στοιχείο ανήκει στο σύνολο

```
fruits = { 'apple', 'banana', 'cherry' }
print('banana' in fruits ) ⇨ True
print('orange' in fruits ) ⇨ False
```

Πράξεις και μέθοδοι πάνω στα σύνολα

Αλλαγή στοιχείου ενός συνόλου

Από την στιγμή που το σύνολο δημιουργείται *δεν μπορούμε να αλλάξουμε ένα στοιχείο του*
όμως μπορούμε να *προσθέσουμε* ή να *αφαιρέσουμε* στοιχεία του

Θυμηθείτε: τα σύνολα περιέχουν μεν αμετάβλητα αντικείμενα, όμως τα ίδια είναι μεταβλητός τύπος δεδομένων

Προσθήκη νέων στοιχείων σε σύνολο

Αν θέλουμε να προσθέσουμε *ένα και μόνο στοιχείο*, χρησιμοποιούμε τη μέθοδο .add()

```
fruits = { 'apple', 'banana', 'cherry' }
fruits.add('orange')
print(fruits) ⇨ {'banana', 'cherry', 'orange' 'apple'}
```

Αν θέλουμε να προσθέσουμε *περισσότερα από ένα στοιχεία*, χρησιμοποιούμε τη μέθοδο .update()

```
fruits = { 'apple', 'banana', 'cherry' }
fruits.update( ['orange', 'mango', 'grapes'] )
print(fruits) ⇨ {'banana', 'grapes', 'cherry', 'orange' 'apple', 'mango'}
```

Τα στοιχεία μέσα σε έναν απαριθμήσιμο Τ.Δ. (λίστα, πλειάδα… κλπ)

Πράξεις και μέθοδοι πάνω στα σύνολα

Εύρεση μεγέθους *(πληθικού αριθμού)* συνόλου

```
fruits = { 'apple', 'banana', 'cherry' }
print(len(fruits))    ⇨  3
```

Αφαίρεση ενός συγκεκριμένου στοιχείου από ένα σύνολο

Υπάρχουν δυο διαφορετικές μέθοδοι: `.remove()` και `.discard()`

```
fruits = { 'apple', 'banana', 'cherry' }
fruits.remove('banana')
print(fruits)    ⇨  {'cherry', 'apple'}
```

Αν το στοιχείο που θέλουμε να αφαιρέσουμε δεν υπάρχει, τότε το `remove` *επιστρέφει μήνυμα λάθους*

```
fruits = { 'apple', 'banana', 'cherry' }
fruits.discard('banana')
print(fruits)    ⇨  {'apple','cherry'}
```

Αν το στοιχείο που θέλουμε να αφαιρέσουμε δεν υπάρχει, τότε το `discard` ***ΔΕΝ*** *επιστρέφει μήνυμα λάθους*

Πράξεις και μέθοδοι πάνω στα σύνολα

Διαγραφή όλων των στοιχείων ενός συνόλου

Η μέθοδος .clear() *διαγράφει όλα τα στοιχεία* ενός συνόλου *(το μετατρέπει σε κενό σύνολο)*

```
fruits = {'apple', 'banana', 'cherry'}
fruits.clear()
print(fruits) ⇨ set()
```

Έτσι υποδηλώνεται το κενό σύνολο. Θυμηθείτε: {} υποδηλώνει κενό λεξικό

Διαγραφή του ίδιου του συνόλου

Η λέξη κλειδί del *διαγράφει τελείως* το σύνολο

```
fruits = {'apple', 'banana', 'cherry'}
del fruits
print(fruits) ⇨  NameError: name 'fruits' is not defined
```

Πράξεις και μέθοδοι πάνω στα σύνολα

Έλεγχος αν ένα σύνολο αποτελεί υποσύνολο ή υπερσύνολο κάποιου άλλου

Χρησιμοποιούμε αντίστοιχα τις μεθόδους .issubset() .issuperset()

```
set1 = {3, 5, 2, 8, 6}
set2 = {6, 5, 3}
set3 = {1,2,3,4,5,6,7,8}
print(set2.issubset(set1))    ⇨ True  (εναλλακτικός τρόπος γραφής: set1 <= set2)
print(set3.issuperset(set1))  ⇨ True  (εναλλακτικός τρόπος γραφής: set3 >= set1)
```

Έλεγχος αν δυο σύνολα είναι ξένα μεταξύ τους *(δεν περιέχουν κοινά στοιχεία)*

Χρησιμοποιούμε τη μέθοδο .isdisjoint()

```
set1 = {3, 5, 2, 8, 6}
set2 = {6, 5, 3}
print(set1.isdisjoint(set2))  ⇨ False
```

Πράξεις και μέθοδοι πάνω στα σύνολα

Ένωση δυο συνόλων (κοινά και μη κοινά στοιχεία από μια φορά το καθένα)

Χρησιμοποιούμε την μέθοδο .union()

```
set1 = {'a', 'b', 'c'}
set2 = {1, 2, 3, 'a'}
set3 = set1.union(set2)   (εναλλακτικός τρόπος γραφής: set1 | set2)
print(set3)
{'c', 'b', 3, 1, 'a', 2}
```

Εναλλακτικά μπορεί να χρησιμοποιηθεί και η μέθοδος .update() που είδαμε νωρίτερα

```
set1 = {'a', 'b', 'c'}
set2 = {1, 2, 3, 'a'}
set1.update(set2)
print(set1)
{2, 3, 1, 'a', 'b', 'c'}
```

Πράξεις και μέθοδοι πάνω στα σύνολα

Τομή δυο συνόλων (κοινά στοιχεία)

Χρησιμοποιούμε την μέθοδο .intersection()

```
set1 = {3, 5, 2, 8}
set2 = {6, 2, 3}

set3 = set1.intersection(set2) (εναλλακτικός τρόπος γραφής: set1 & set2 )

print(set3)

{3, 2}
```

Πράξεις και μέθοδοι πάνω στα σύνολα

Διαφορά δυο συνόλων Χρησιμοποιούμε την μέθοδο `.difference()`

`A.difference(B)` : όλα τα στοιχεία του A που δεν ανήκουν στο B

```
set1 = {3, 5, 2, 8}
set2 = {6, 2, 3}
print(set1.difference(set2))
{8, 5}
```

```
set1 = {3, 5, 2, 8}
set2 = {6, 2, 3}
print(set2.difference(set1))
    {6}
```

εναλλακτικός τρόπος γραφής διαφοράς συνόλων : `set1 - set2` *αντί για* `set1.difference(set2)`

Αν το αποτέλεσμα της πράξης θέλουμε να αποθηκευτεί *πίσω στο αρχικό σύνολο* και όχι σε προσδιοριστή, τότε μπορούμε να χρησιμοποιήσουμε εναλλακτικά την μέθοδο `.difference_update()`

```
set1 = {3, 5, 2, 8}
set2 = {6, 2, 3}
set1.difference_update(set2)
print(set1) ⇨   {8, 5}
```

```
set1 = {3, 5, 2, 8}
set2 = {6, 2, 3}
set2.difference_update(set1)
print(set2) ⇨   {6}
```

Πράξεις και μέθοδοι πάνω στα σύνολα

Συμμετρική διαφορά δυο συνόλων Χρησιμοποιούμε την μέθοδο `.symmetric_difference()`

`A.symmetric_difference(B)` : όλα τα μη κοινά στοιχεία των A και B

```
set1 = {3, 5, 2, 8}
set2 = {6, 2, 3}
set3 = set1.symmetric_difference(set2)  (εναλλακτικός τρόπος γραφής: set1 ^ set2 )
print(set3) ⇨  {8, 6, 5}
```

Αν το αποτέλεσμα της πράξης θέλουμε να αποθηκευτεί *πίσω στο αρχικό σύνολο* και όχι σε νέο προσδιοριστή τότε μπορούμε να χρησιμοποιήσουμε εναλλακτικά την μέθοδο `.symmetric_difference_update()`

```
set1 = {3, 5, 2, 8}
set2 = {6, 2, 3}
set1.symmetric_difference_update(set2)
print(set1) ⇨  {5, 6, 8}
```

Σύνοψη πράξεων και μεθόδων πάνω στα σύνολα

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item If the item to remove does not exist, NO error is raised
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element If the item to remove does not exist, an error is raised.
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Παγιωμένα -ή παγωμένα- σύνολα (frozen sets)

Σύνολα που αφού δημιουργηθούν, *δεν μπορούν πια να τροποποιηθούν* προσθέτοντας ή αφαιρώντας άλλα στοιχεία *(είναι αμετάβλητα αντικείμενα)*

Τα παγιωμένα σύνολα δημιουργούνται με την συνάρτηση `frozenset()`

```
x = frozenset( { 1, 2, 3} )  #Μετατροπή κανονικού συνόλου σε frozenset
print(x)  ⇨ frozenset ( { 3, 1, 2} )

x = frozenset( [ 1, 2, 3] )  # Μετατροπή λίστας σε frozenset
print(x)  ⇨ frozenset ( { 1, 3, 2} )

x = frozenset( range(0:5) ) #Δημιουργία frozenset από range
print(x)  ⇨ frozenset ( {4, 1, 3, 0, 2}  )

x = frozenset( 'Hello' ) # Δημιουργία frozenset από συμβολοσειρά
print(x)  ⇨ frozenset ( { 'e', 'H', 'o', 'l'} )
```

Καθώς τα παγιωμένα σύνολα είναι *αμετάβλητα αντικείμενα (immutables)*, μπορούν να αποτελέσουν *στοιχεία άλλων συνόλων (ή κλειδιά λεξικών όπως θα δούμε παρακάτω)*

Ο Τύπος δεδομένων dictionary (λεξικό)

Διατεταγμένες[(*)] συλλογές *μοναδικών* ζευγών στοιχείων της μορφής *(κλειδί : τιμή)* όπου το πρώτο στοιχείο κάθε ζεύγους *(δηλαδή το κλειδί)* είναι *αμετάβλητο*

Σ' ένα λεξικό αποθηκεύονται *απεικονίσεις*, δηλαδή ζεύγη στοιχείων όπου το πρώτο που ονομάζεται *κλειδί (key)* αντιστοιχίζεται στο δεύτερο που ονομάζεται *τιμή (value)*

```
mycar = { 'brand':'VW',  'model':'Golf', 'year':2025 }
          κλειδί  τιμή     κλειδί  τιμή    κλειδί  τιμή
```

- Τα *κλειδιά* ενός λεξικού αποτελούνται μόνο από *αμετάβλητα* στοιχεία
- Τα *κλειδιά* ενός λεξικού είναι *μοναδικά*
- Οι *τιμές* ενός λεξικού μπορεί να είναι *οποιοδήποτε αντικείμενο της Python και δεν είναι απαραίτητα μοναδικές (πολλά κλειδιά με ίδια τιμή είναι Ok)*

Ένα *λεξικό* είναι στην ουσία ένα ***σύνολο*** *από μοναδικά κλειδιά* σε κάθε ένα από τα οποία έχουμε αντιστοιχίσει και μια *τιμή*

(*) Από την Python 3.7 και μετα, υπάρχει συγκεκριμένη διάταξη των στοιχείων ενός λεξικού, που προκύπτει από την σειρά εισαγωγής τους

Ο Τύπος δεδομένων dictionary (λεξικό)

Μπορούμε να προσπελάσουμε οποιοδήποτε στοιχείο ενός λεξικού χρησιμοποιώντας ως δείκτη το *κλειδί* αυτού του στοιχείου

Στην περίπτωση αυτή μας επιστρέφεται η *τιμή* που αντιστοιχεί σε αυτό το κλειδί

```
>>> mycar = {'brand':'VW', 'model':'Golf', 'year':2025}
>>> print(mycar['model'])
'Golf'
```

Δίνουμε το κλειδί 'model'
Μας επιστρέφεται η τιμή 'Golf'

Καθώς τα λεξικά είναι μεταβλητά αντικείμενα, μπορούμε να *αλλάζουμε τις τιμές των κλειδιών τους*, καθώς και να *προσθέτουμε/διαγράφουμε στοιχεία*

```
>>> mycar['model'] = 'Polo'
>>> print(mycar['model'])
'Polo'
```

```
>>> mycar['color'] = 'Blue'
>>> print(mycar['color'])
'Blue'
```

Νέο κλειδί 'color'

Τρόποι δημιουργίας λεξικών

Με απ' ευθείας αναγραφή των στοιχείων του *(μπορεί να γίνει σε μια ή περισσότερες γραμμές)*

```
mycar = { 'brand':'VW',
 'model':'Golf',
 'year': 2025 }
print(mycar) ⇨ {'brand':'VW', 'model':'Golf', 'year':2025}
```

Με χρήση της μεθόδου `.fromkeys(s,v)` πάνω στο `dict` *(ή σε ένα υπάρχον λεξικό)*

`s`: μια *ακολουθία τιμών* που θα αποτελέσουν τα *κλειδιά* του λεξικού

`v`: [προαιρετική] τιμή στην οποία αρχικοποιούνται *όλα τα κλειδιά*

```
key_list = ['a', 'b', 'c']   (εκτός από λίστα θα μπορούσε να είναι πλειάδα ή σύνολο)
d = dict.fromkeys(key_list, 5)
print(d)       ⇨ {'a':5, 'b':5, 'c':5}
d1 = d.fromkeys(key_list)
print(d1)  ⇨ {'a':None, 'b':None, 'c':None}
```

Αν παραληφθεί το **v**, τότε όλα τα κλειδιά αρχικοποιούνται σε None

Προσθήκη νέων στοιχείων στο λεξικό

Απ' ευθείας

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025}
mycar['color'] = 'blue'
mycar['engine'] = 1600
{'brand':'VW', 'model':'Golf', 'year':2025, 'color':'blue', 'engine':1600}
```

Με χρήση της μεθόδου update()

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025}
mycar.update({'color':'blue','engine':1600})
mycar.update((('color','blue'),('engine',1600)))
{'brand':'VW', 'model':'Golf', 'year':2025, 'color':'blue', 'engine':1600}
```

Πλειάδα δύο στοιχείων, καθένα από οποία είναι μια πλειάδα

Εναλλακτικός τρόπος γραφής

Αν χρησιμοποιήσουμε ένα *κλειδί που ήδη υπάρχει*, τότε απλά *αλλάζει η τιμή αυτού του κλειδιού*

```
mycar.update({'engine':2000})
{'brand':'VW', 'model':'Golf', 'year':2025, 'color':'blue', 'engine':2000}
```

Πράξεις και μέθοδοι πάνω στα λεξικά

Εκτύπωση τιμών που αντιστοιχούν σε συγκεκριμένα κλειδιά

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025}
print(mycar['model']) ⇨ Golf
print(mycar['year'])  ⇨ 2025
```

Εναλλακτικός τρόπος με χρήση της μεθόδου `get()`

```
x = mycar.get('year')
print(x)  ⇨ 2025
```

Αλλαγή τιμής ενός κλειδιού

(Θυμηθείτε: οι τιμές των κλειδιών αλλάζουν, *τα ίδια τα κλειδιά **δεν** αλλάζουν)*

```
mycar['year'] = 2007

print(mycar['year']) ⇨ 2007
```

Πράξεις και μέθοδοι πάνω στα λεξικά

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025 }
```

Εκτύπωση όλων των ***κλειδιών*** ενός λεξικού

```
for x in mycar:
  print(x)
```

```
brand
model
year
```

Με το for διατρέχουμε τα *κλειδιά* ενός λεξικού

Εκτύπωση όλων των ***τιμών*** ενός λεξικού

```
for x in mycar:
  print(mycar[x])
```

```
VW
Golf
2025
```

```
for x in mycar:
  print(x, '=', mycar[x], end = '/')
```

```
brand = VW / model = Golf / year = 2025 /
```

Πράξεις και μέθοδοι πάνω στα λεξικά

Έλεγχος αν ένα κλειδί υπάρχει στο λεξικό

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025 }
if 'model' in mycar:
    print('Yes, 'model' is a key in this dictionary')
```

```
Yes, 'model' is a key in this dictionary
```

Μέγεθος *(πλήθος ζευγών)* λεξικού

```
print(len(mycar)) ⇨ 3
```

Πράξεις και μέθοδοι πάνω στα λεξικά

Εξαγωγή συγκεκριμένου στοιχείου λεξικού με τη μέθοδο *pop(<key>)*

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025 }
x = mycar.pop('model') #Εξαγωγή της τιμής του στοιχείου με το συγκεκριμένο κλειδί
print(x, "/", mycar) ⇨ Golf / {'brand':'VW', 'year':2025}
```

Με τη μέθοδο pop() επιστρέφεται *η τιμή του κλειδιού* – το ίδιο το *κλειδί χάνεται*

```
x = mycar.pop('engine') #αν δεν βρεθεί το κλειδί, εγείρεται σφάλμα
KeyError: 'engine'
```

Εναλλακτική μορφή: *pop(<key>,<default>)* #αν δεν βρεθεί το κλειδί, επιστρέφει την τιμή default

```
x = mycar.pop('engine', '???' )
print(x, "/", mycar) ⇨ ??? / {'brand':'VW', 'model':'Golf', 'year':2025}
```

Πράξεις και μέθοδοι πάνω στα λεξικά

Εξαγωγή ***τελευταίου*** στοιχείου ενός λεξικού με τη μέθοδο *popitem()*

Επιστρέφει σε μορφή ζεύγους *(key, value)* το *τελευταίο* στοιχείο του λεξικού

Η μέθοδος *δεν δέχεται παραμέτρους*

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025 }
x = mycar.popitem() #Εξαγωγή του τελευταίου στοιχείου
print(mycar) ⇨  {'brand':'VW', 'model':'Golf'}
print(x) ⇨ ('year', 2025)
```

Πράξεις και μέθοδοι πάνω στα λεξικά

Διαγραφή στοιχείων λεξικού με τη λέξη κλειδί `del`

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025 }
del mycar['model'] #Διαγραφή του στοιχείου με το συγκεκριμένο κλειδί
print(mycar) ⇨  { 'brand':'VW', 'year':2025 }
```

Διαγραφή ολόκληρου του λεξικού με τη λέξη κλειδί `del`

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025 }
del mycar #Διαγραφή ολόκληρου του λεξικού
print(mycar) ⇨ NameError: name 'mycar' is not defined
```

Πράξεις και μέθοδοι πάνω στα λεξικά

Διαγραφή περιεχομένου *(καθάρισμα)* ενός λεξικού με τη μέθοδο `clear`

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025 }
mycar.clear()
print(mycar) ⇨  {}
```

Θυμόμαστε `{}` : κενό λεξικό, ενώ `set()` : κενό σύνολο

Αντιγραφή λεξικού με τη συνάρτηση *`dict()`*

```
mycar = {'brand':'VW', 'model':'Golf', 'year':2025 }
newcar = dict(mycar)
print(newcar) ⇨ { 'brand':'VW', 'model':'Golf', 'year':2025}
```

Στην ουσία η dict() χρησιμοποιείται εδώ για να δημιουργήσει ένα νέο λεξικό από τα στοιχεία του παλιού

Πράξεις και μέθοδοι πάνω στα λεξικά

Δημιουργία σύνθετων λεξικών που αποτελούνται από λεξικά (nested dictionaries)

```
my_family = {
  'child1' : {
    'name' : 'George',
    'year' : 2004
  },
  'child2' : {
    'name' : 'Maria',
    'year' : 2007
  },
  'child3' : {
    'name' : 'John',
    'year' : 2011
  }
}
```

Η τιμή που αντιστοιχεί στο κλειδί `'child1'` είναι ένα λεξικό δύο στοιχείων

```
>>> my_family['child1']
{'name': 'George', 'year': 2004}

>>> my_family['child1']['year']
2004

>>> my_family['child3']['name']
'John'
```

```
>>> my_family
{'child_1': {'name': 'George', 'year': 2004}, 'child_2': {'name': 'Maria',
'year': 2007}, 'child_3': {'name': 'John', 'year': 2011}}
```

Πράξεις και μέθοδοι πάνω στα λεξικά

Εναλλακτικός τρόπος: δημιουργία τριών λεξικών, και στη συνέχεια δημιουργία ενός νέου λεξικού που τα περιέχει

```
first  = {
  'name' : 'George',
  'year' : 2004
}
 second  = {
  'name' : 'Maria',
  'year' : 2007
}
 third  = {
  'name' : 'John',
  'year' : 2011
}

my_family = {'child1':first, 'child2':second, 'child3':third}
print(my_family)
```

```
{'child1': {'name': 'George', 'year': 2004}, 'child2': {'name': 'Maria', 'year':
2007}, 'child3': {'name': 'John', 'year': 2011}}
```

Σύνοψη πράξεων και μεθόδων πάνω στα λεξικά

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Διαχείριση σφαλμάτων χρόνου εκτέλεσης στην Python

- Η Python υποστηρίζει εναν *μηχανισμό διαχείρισης σφαλμάτων* που συμβαίνουν κατά την διάρκεια της *εκτέλεσης* ενός προγράμματος (*run time errors*) ο οποίος ονομάζεται `try-except`
- Ο μηχανισμός αυτός μας δίνει την δυνατότητα να τα χειριστούμε αυτά τα σφάλματα εσωτερικά χωρίς να σταματήσει να εκτελείται το πρόγραμμα. *(χωρίς να 'χτυπήσει' το πρόγραμμα).* Αυτό γίνεται με την βοήθεια των λεγομένων *εξαιρέσεων (exceptions)*
- Κάθε φορά που συμβαίνει ένα *σφάλμα εκτέλεσης* η Python εγείρει μια *εξαίρεση (exception)* σχετική με το σφάλμα που συνέβη

  πχ. διαίρεση δια του μηδενός: η Python εγείρει την εξαίρεση `ZeroDivisionError`

  πχ. άγνωστος προσδιοριστής: η Python εγείρει την εξαίρεση `NameError`

- Αν δεν έχουμε υλοποιήσει στο πρόγραμμά μας τον μηχανισμό, `try-except` τότε σε περίπτωση σφάλματος εκτέλεσης το πρόγραμμα *σταματάει να εκτελείται* και μας εμφανίζει *(συνήθως με κόκκινα γράμματα)* το *όνομα της εξαίρεσης* και το *σημείο του προγράμματος* όπου προέκυψε *δείτε το παράδειγμα στην επόμενη διαφάνεια*

Διαχείριση σφαλμάτων χρόνου εκτέλεσης στην Python

```
x = int(input('Δώσε έναν αριθμό: '))
y = 10 / x
print(y)
```

```
Δώσε έναν αριθμό: 0
Traceback (most recent call last):
  File '<module1>', line 2, in <module>
ZeroDivisionError: division by zero
```

```
Δώσε έναν αριθμό: Καλημέρα
Traceback (most recent call last):
  File '<module1>', line 1, in <module>
ValueError: invalid literal for int() with base 10: 'ααα'
```

Run time errors
Εξαιρέσεις (exceptions)

Δομή διαχείρισης εξαιρέσεων της Python `try` / `except`

Γενική μορφή

```
try:
    μπλοκ_εντολών
except όνομα_εξαίρεσης_1:
    εντολές_χειρισμού_εξαίρεσης_1
except όνομα_εξαίρεσης_2:
    εντολές_χειρισμού_εξαίρεσης_2
....
else:
    εντολές_χειρισμού_λοιπών_εξαιρέσεων
finally:
    εντολές_"καθαρισμού"
```

το τμήμα εντολών του προγράμματος στο οποίο επιδρά το `try` / `except`

εντολές που εκτελούνται αν εγερθεί η εξαίρεση_1

εντολές που εκτελούνται αν εγερθεί η εξαίρεση_2

εντολές που εκτελούνται αν εγερθεί κάποια εξαίρεση διαφορετική από όλες τις παραπάνω

εκτελούνται ***πάντα*** *(είτε εγερθεί εξαίρεση είτε όχι)* και συνήθως αφορούν το 'καθάρισμα' της δομής πριν τη συνέχεια του προγράμματος

Προαιρετικά

Η απλούστερη δυνατή μορφή του `try` / `except`

```
try:
    μπλοκ_εντολών
except:
    εντολές_χειρισμού_εξαιρέσεων
```

το τμήμα εντολών του προγράμματος στο οποίο επιδρά το `try` / `except`

εκτελούνται αν εγερθεί ***οποιαδήποτε*** εξαίρεση

Εναλλακτική μορφή

```
try:
    μπλοκ_εντολών
except Exception as e:
    print(e) # Σύντομη περιγραφή του σφάλματος (εξαίρεσης)
    υπόλοιπες_εντολές_χειρισμού_εξαιρέσεων  (προαιρετικά)
```

Λίστα εξαιρέσεων της Python *(standard exceptions)*

Exception Name & Description
Exception Base class for all exceptions (έχει ξεχωριστή χρήση από τις υπόλοιπες)
StopIteration Raised when the next() method of an iterator does not point to any object.
SystemExit Raised by the sys.exit() function.
StandardError Base class for all built-in exceptions except StopIteration and SystemExit.
ArithmeticError Base class for all errors that occur for numeric calculation.
OverflowError Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError Raised when a floating point calculation fails.
ZeroDivisionError Raised when division or modulo by zero takes place for all numeric types.
AssertionError Raised in case of failure of the Assert statement.
AttributeError Raised in case of failure of attribute reference or assignment.
EOFError Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError Raised when an import statement fails.
KeyboardInterrupt Raised when the user interrupts program execution, usually by pressing Ctrl+c.
LookupError Base class for all lookup errors.
IndexError Raised when an index is not found in a sequence.

Exception Name & Description
KeyError Raised when the specified key is not found in the dictionary.
NameError Raised when an identifier is not found in the local or global namespace.
UnboundLocalError Raised when trying to access a local variable in a function or method but no value has been assigned to it.
EnvironmentError Base class for all exceptions that occur outside the Python environment.
IOError Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
OSError Raised for operating system-related errors.
SyntaxError Raised when there is an error in Python syntax.
IndentationError Raised when indentation is not specified properly.
SystemError Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
SystemExit Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
TypeError Raised when an operation or function is attempted that is invalid for the specified data type.
ValueError Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError Raised when a generated error does not fall into any category.
NotImplementedError Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

Παραδείγματα διαχείρισης σφαλμάτων μέσω εξαιρέσεων

```
try:
    x = int(input('Δώσε έναν αριθμό: '))
    y = 10 / x
    print(y)
except:
    print('Εντοπίστηκε σφάλμα')
```

```
Δώσε έναν αριθμό: 0
Εντοπίστηκε σφάλμα
```

```
try:
    x = int(input('Δώσε έναν αριθμό: '))
    y = 10 / x
    print(y)
except ZeroDivisionError:
    print('Διαίρεση δια του μηδενός')
except:
    print('Άλλο σφάλμα ')
```

```
Δώσε έναν αριθμό: 0
Διαίρεση δια του μηδενός

Δώσε έναν αριθμό: abc
Άλλο σφάλμα
```

```
try:
    x = int(input('Δώσε έναν αριθμό: '))
    y = 10 / x
    print(y)
except ZeroDivisionError:
    print('Διαίρεση δια του μηδενός')
except Exception as e:
    print(e) # περιγραφή του σφάλματος
print('Το πρόγραμμα συνεχίζει να εκτελείται')
```

```
Δώσε έναν αριθμό: 0
Διαίρεση δια του μηδενός
Το πρόγραμμα συνεχίζει να εκτελείται

Δώσε έναν αριθμό: abc
invalid literal for int() with base 10: 'abc'
Το πρόγραμμα συνεχίζει να εκτελείται
```

Τέλος Διάλεξης

Ερωτήσεις;

Τμήματα αυτής της διάλεξης περιέχουν πληροφορίες από πηγές που είναι ελεύθερες στο διαδίκτυο όπως η Βικιπαίδεια και ανοιχτές σημειώσεις παρεμφερών διαλέξεων.