Εφαρμογές Η/Υ

3ο Μάθημα

- Σύνθετοι τύποι αντικειμένων
  - Πλειάδες
  - Σύνολα
  - Λεξικά
- Μορφοποιημένη Έξοδος (οθόνη, αρχείο)
- Είσοδος δεδομένων από αρχείο

Ν. Λαγαρός, Θ. Στάμος, Χ. Φραγκουδάκης

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Πλειάδες

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> u = t, (1, 2, 3, 4, 5) # Οι πλειάδες μπορεί να είναι εμφωλευμένες
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> t[0] = 88888 # Οι πλειάδες δεν μεταλλάσσονται
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> v = ('a', [3, 2, 1]) # Μπορεί να περιέχουν αντικείμενα που μεταλλάσσονται
>>> v
('a', [3, 2, 1])
```

- Μια σειρά αντικειμένων που χωρίζονται με κόμματα
- Πρόσβαση στα στοιχεία με δεικτοδότηση (όπως στις λίστες)
- Δεν μεταλλάσσονται με την ανάθεση στα στοιχεία τους (immutable)
- Ενδεχομένως περιέχουν αντικείμενα που μεταλλάσσονται με την ανάθεση

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Πλειάδες

```
>>> alist = [1, 2, 3]
>>> atuple = 1, 2, 3, alist
>>> atuple
(1, 2, 3, [1, 2, 3])
>>> atuple[0] = 'one'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> atuple[3] = 'alist like [1, 2, 3]'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> atuple[3][0] = 'one'
>>> atuple
(1, 2, 3, ['one', 2, 3])
```

- Οι πλειάδες δεν μεταλλάσσονται με την ανάθεση στα στοιχεία τους
- Όμως το `atuple[3]` είναι μια λίστα που μεταλλάσσεται με την ανάθεση
- `atuple[3][0] = 'one'` αναθέτει στο πρώτο στοιχείο της λίστας `[1, 2, 3]`

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Πλειάδες

```
>>> empty = ()                  # Κενή πλειάδα
>>> singleton = 'hello',        # Πλειάδα με ένα στοιχείο (παρατηρήστε το κόμμα)
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)
>>> def basic_operations(x, y):          # Μια συνάρτηση που επιστρέφει πλειάδα
...     return x+y, x-y, x*y, x/y                    # Tuple packing
>>> sum, dif, mul, div = basic_operations(10, 11)   # Tuple unpacking
>>> sum, dif, mul, div
(21, -1, 110, 0.9090909090909091)
```

- Κενή πλειάδα, πλειάδα με ένα μόνο στοιχείο
- Μια συνάρτηση μπορεί να επιστρέψει πολλές τιμές σε μια πλειάδα
- Συσκευασία αντικειμένων σε πακέτο (tuple packing)
- Αποσυσκευασία αντικειμένων από το πακέτο (tuple unpacking)
- Η πολλαπλή ανάθεση είναι ένας συνδυασμός του tuple packing/unpacking

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Σύνολα

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # Δεν υπάρχουν διπλά αντικείμενα
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # γρήγορος έλεγχος εγκλεισμού στο σύνολο
True
>>> 'crabgrass' in basket
False

>>> a = set('abracadabra')             # Το σύνολο γραμμάτων του 'abracadabra'
>>> b = set('alacazam')                # Το σύνολο γραμμάτων του 'alacazam'
>>> a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # Διαφορά συνόλων
{'r', 'd', 'b'}
>>> a | b                              # Ένωση συνόλων
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # Τομή συνόλων
{'a', 'c'}
>>> a ^ b                              # Συμμετρική διαφορά συνόλων
{'r', 'd', 'b', 'm', 'z', 'l'}

>>> # Περιγραφές συνόλων όπως και στις λίστες
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
{'r', 'd'}
```

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Σύνολα

```
>>> from math import sqrt
>>> n = 100
>>> sqrt_n = int(sqrt(n))
>>> no_primes = {j for i in range(2, sqrt_n) for j in range(i*2, n, i)}
>>> no_primes
{4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30, 32,
33, 34, 35, 36, 38, 39, 40, 42, 44, 45, 46, 48, 49, 50, 51, 52, 54, 55, 56, 57,
58, 60, 62, 63, 64, 65, 66, 68, 69, 70, 72, 74, 75, 76, 77, 78, 80, 81, 82, 84,
85, 86, 87, 88, 90, 91, 92, 93, 94, 95, 96, 98, 99}
>>> primes = {i for i in range(n) if i not in no_primes}
>>> print(primes)
{0, 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97}
```

- Κόσκινο του Ερατοσθένη: εύρεση όλων των πρώτων αριθμών έως το n
  - Όλα τα πολλαπλάσια των αριθμών δεν είναι πρώτοι αριθμοί
  - Σχηματίζουμε το σύνολο των μη πρώτων αριθμών
  - Πρώτοι αριθμοί είναι όλοι οι αριθμοί έως το n που δεν είναι μη πρώτοι

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> list(tel.keys())      # ισοδύναμα [key for key in tel.keys()]
['irv', 'guido', 'jack']
>>> sorted(tel.keys())
['guido', 'irv', 'jack']
>>> 'guido' in tel        # έλεγχος εγκλεισμού για το κλειδί 'guido'
True
>>> 'jack' not in tel
False
```

- Τα λεξικά είναι συλλογές αντικειμένων σε "ονοματισμένες" θέσεις (κλειδιά)
- Τα κλειδιά είναι μοναδικά και δεν μπορεί να είναι τύποι που μεταλλάσσονται

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> city = {"New York City":8175133, "Los Angeles": 3792621, "Washington":
632323, "Chicago": 2695598, "Toronto":2615060, "Montreal":11854442, "Ottawa":
883391, "Boston":62600}
>>> city["New York City"]
8175133
>>> city["Toronto"]
2615060
>>> city["Boston"]
62600
```

```
>>> city["Detroit"]  # Ένα κλειδί πρέπει να έχει οριστεί για να χρησιμοποιηθεί
Traceback (most recent call last):
  File "<stdin>", line 1, in
KeyError: 'Detroit'
>>> city # Δεν υπάρχει διάταξη στα περιεχόμενα του λεξικού
{'Toronto': 2615060, 'Ottawa': 883391, 'Los Angeles': 3792621, 'Chicago':
2695598, 'New York City': 8175133, 'Boston': 62600, 'Washington': 632323,
'Montreal': 11854442}
```

- Ένα κλειδί πρέπει να έχει οριστεί για να χρησιμοποιηθεί
- Στα λεξικά δεν υπάρχει διάταξη σε αντίθεση με τις πλειάδες και τις λίστες

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> food = {"ham" : "yes", "egg" : "yes", "spam" : "no" }
>>> food
{'egg': 'yes', 'ham': 'yes', 'spam': 'no'}
>>> food["spam"] = "yes"
>>> food
{'egg': 'yes', 'ham': 'yes', 'spam': 'yes'}
```

- Ενώ τα κλειδιά είναι διαφορετικά οι τιμές μπορεί να επαναλαμβάνονται

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> # Ένα αγγλογερμανικό λεξικό
... en_de = {"red" : "rot", "green" : "grün", "blue" : "blau", "yellow":"gelb"}
>>> # Ένα γερμανογαλλικό λεξικό
... de_fr = {"rot" : "rouge", "grün" : "vert", "blau" : "bleu", "gelb":"jaune"}
>>> print("The German word for red is:", en_de["red"])
The German word for red is: rot  # Αγγλικά -> Γερμανικά
>>> print("The French word for the German word rot is:", de_fr["rot"])
The French word for the German word rot is: rouge # Γερμανικά -> Γαλλικά
>>> print("The French word for red is:", de_fr[en_de["red"]])
The French word for red is: rouge # Αγγλικά -> Γαλλικά μέσω των Γερμανικών
```

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> # Ένα αγγλογερμανικό λεξικό
... en_de = {"red" : "rot", "green" : "grün", "blue" : "blau", "yellow":"gelb"}
>>> # Ένα γερμανογαλλικό λεξικό
... de_fr = {"rot" : "rouge", "grün" : "vert", "blau" : "bleu", "gelb":"jaune"}
>>> print("The German word for red is:", en_de["red"])
The German word for red is: rot  # Αγγλικά -> Γερμανικά
>>> print("The French word for the German word rot is:", de_fr["rot"])
The French word for the German word rot is: rouge # Γερμανικά -> Γαλλικά
>>> print("The French word for red is:", de_fr[en_de["red"]])
The French word for red is: rouge # Αγγλικά -> Γαλλικά μέσω των Γερμανικών
```

Ανέκδοτο των γλωσσολόγων για τους αυτόματους μεταφραστές:

> Δώθηκε στον αυτόματο μεταφραστή η έκφραση:
>
> The spirit is strong but the flesh is weak
>
> Ζητήθηκε η μετάφραση στα Ρωσικά και από τα Ρωσικά πίσω στα Αγγλικά:
>
> The vodka is good but the meat is rotten

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> morse = {"A" : ".-", "B" : "-...", "C" : "-.-.", "D" : "-..", "E" : ".",
"F" : "..-.", "G" : "--.", "H" : "....", "I" : "..", "J" : ".---", "K" : "-.-",
"L": ".-..", "M" : "--", "N" : "-.", "O" : "---", "P" : ".--.", "Q" : "--.-",
"R": ".-.", "S" : "...", "T" : "-", "U" : "..-", "V" : "...-", "W" : ".--",
"X": "-..-", "Y" : "-.--", "Z" : "--..", "0" : "-----", "1" : ".----",
"2" : "..---", "3" : "...--", "4" : "....-", "5" : ".....", "6" : "-....",
"7" : "-- ...", "8" : "---..", "9" : "----.", "." : ".-.-.-", "," : "--..--"}
>>> message = 'ATTACK.TOMMOROW.AT.500.HOURS'
>>> for letter in message:
...     print(morse[letter], end=" ")
...
.- - - .- -.-. -.- .-.-.- - --- -- -- --- .-. --- .-- .-.-.- .- - .-.-.- .....
----- ----- .-.-.- .... --- ..- .-. ...
>>> len(morse)
38
>>> "a" in morse
False
>>> "A" in morse
True
>>> "a" not in morse
True
```

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> dishes = ["pizza", "sauerkraut", "paella", "hamburger"]
>>> countries = ["Italy", "Germany", "Spain", "USA"]
>>> country_specialities = list(zip(countries, dishes))
>>> print(country_specialities)
[('Italy', 'pizza'), ('Germany', 'sauerkraut'), ('Spain', 'paella'),
('USA', 'hamburger')]
>>> country_specialities_dict = dict(country_specialities)
>>> print(country_specialities_dict)
{'USA': 'hamburger', 'Germany': 'sauerkraut', 'Italy': 'pizza',
'Spain': 'paella'}
```

- Μετατροπή λιστών σε λεξικά
- Η συνάρτηση `zip()` λειτουργεί όπως το φερμουάρ: δημιουργεί πλειάδες με πρώτο στοιχείο από την πρώτη και δεύτερο από τη δεύτερη λίστα
- `list(zip(countries, dishes))` δημιουργεί μια λίστα πλειάδων
- `dict(country_specialities)` δημιουργεί λεξικό από τη λίστα πλειάδων:
  - Κλειδιά είναι τα πρώτα στοιχεία της πλειάδας
  - Τιμές είναι τα δεύτερα στοιχεία της πλειάδας

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> knowledge = {"Frank": {"Perl"}, "Monica": {"C", "C++"}}
>>> knowledge2 = {"Guido":{"Python"}, "Frank":{"Perl", "Python"}}
>>> knowledge.update(knowledge2)
>>> knowledge
{'Frank': {'Python', 'Perl'}, 'Monica': {'C++', 'C'}, 'Guido': {'Python'}}
>>> knowledge.clear()
>>> knowledge
{}
```

- Η μέθοδος `update()` ενημερώνει ένα λεξικό από ένα άλλο λεξικό
- Η μέθοδος `clear()` διαγράφει όλα τα περιεχόμενα (κλειδιά και τιμές) του λεξικού

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for key, value in knights.items(): # Διατρέχει τιμές και κλειδιά
...     print(key, value)
gallahad the pure
robin the brave
>>> for key in knights.keys():         # Διατρέχει τα κλειδιά
...     print(key)
gallahad
robin
>>> for value in knights.values():     # Διατρέχει τις τιμές
...     print(value)
the pure
the brave
```

- Μέθοδοι που διατρέχουν το λεξικό
- Μέθοδος `items()`
- Μέθοδος `keys()`
- Μέθοδος `values()`

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> adict = {"list":"Liste", "dictionary":"Wörterbuch", "function":"Funktion"}
>>> alist = [("list","Liste"), ("dictionary","Wörterbuch"),
("function","Funktion")]

>>> def lookup(alist, value):
...     for atuple in alist:
...         if atuple[0] == value:
...             return atuple[1]
...
>>> lookup(alist, 'function')
'Funktion'
>>> adict['function']
'Funktion'
```

- Πολυπλοκότητα αναζήτησης
- `adict` και `alist` περιέχουν την ίδια πληροφορία
- Η αναζήτηση στην `alist` ενδεχομένως να εξαντλήσει όλα τα ζεύγη ($\mathrm{O}(n)$)
- Η αναζήτηση στο `adict` επιστρέφει άμεσα το αποτέλεσμα ($\mathrm{O}(1)$)

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

Πέτρα-Ψαλίδι-Χαρτί.

```
playera = input("πέτρα, ψαλίδι ή χαρτί; (παίκτης Α) ")
playerb = input("πέτρα, ψαλίδι ή χαρτί; (παίκτης Β) ")

if playera == playerb:
        print("Ισοπαλία!")
elif playera == "πέτρα":
    if playerb == "ψαλίδι":
        print("Κερδίζει ο παίκτης Α!")
    else:
        print("Κερδίζει ο παίκτης Β!")
elif playera == "ψαλίδι":
    if playerb == "χαρτί":
        print("Κερδίζει ο παίκτης Α!")
    else:
        print("Κερδίζει ο παίκτης Β!")
elif playera == "χαρτί":
    if playerb == "πέτρα":
        print("Κερδίζει ο παίκτης Α!")
    else:
        print("Κερδίζει ο παίκτης Β!")
```

Υπάρχει πιο κομψή λύση με χρήση λεξικού

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
wins = {
    'πέτρα': 'ψαλίδι',
    'ψαλίδι': 'χαρτί',
    'χαρτί': 'πέτρα'
}

playera = input("πέτρα, ψαλίδι ή χαρτί; (παίκτης Α) ")
playerb = input("πέτρα, ψαλίδι ή χαρτί; (παίκτης Β) ")

if playera == playerb:
        print("Ισοπαλία!")
elif wins[playera] == playerb:
    print("Κερδίζει ο παίκτης Α!")
else:
    print("Κερδίζει ο παίκτης Β!")
```

Με τη χρήση του λεξικού `wins` εξετάζουμε άμεσα αν η επιλογή του πρώτου παίκτη κερδίζει την επιλογή του δεύτερου, αλλιώς κερδίζει ο δεύτερος.

Εισαγωγή στην Python

Σύνθετοι τύποι αντικειμένων: Λεξικά

```
>>> a = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> b = 'nopqrstuvwxyzabcdefghijklmNOPQRSTUVWXYZABCDEFGHIJKLM'
>>> rot13e = dict(list(zip(a,b))) # Δημιουργία λεξικού για την κωδικοποίηση
>>> rot13d = dict(list(zip(b,a))) # Δημιουργία λεξικού για την αποκωδικοποίηση
>>> encrypt = {ord(k):ord(v) for k,v in rot13e.items()}
>>> decrypt = {ord(k):ord(v) for k,v in rot13d.items()}
>>> 'Caesar cipher? I much prefer Caesar salad!'.translate(encrypt)
'Pnrfne pvcure? V zhpu cersre Pnrfne fnynq!'
>>> 'Pnrfne pvcure? V zhpu cersre Pnrfne fnynq!'.translate(decrypt)
'Caesar cipher? I much prefer Caesar salad!'
```

- Στοιχειώδης κρυπτογραφία. Κώδικας του Ιούλιου Καίσαρα (Caesar cipher)
- Λέγεται ότι έτσι επικοινωνούσε ο Ιούλιος Καίσαρας με τους στρατηγούς του
- Αντικαθιστούσε κάθε γράμμα "κυλώντας το κυκλικά" 13 θεσεις
- `ord(char)` είναι ο αύξοντας αριθμός του `char` στον πίνακα χαρακτήρων
- Η μέθοδος `translate(d)` "μεταφράζει" ανάλογα με το λεξικό `d` που αντιστοιχίζει αύξοντες αριθμούς χαρακτήρων σε άλλους αύξοντες αριθμούς

Εισαγωγή στην Python

Μορφοποιημένη Έξοδος

```
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad')
'abracadabra'
```

- Τα `{` και `}` υποδεικνύουν θέσεις που θα δεχτούν τιμές (placeholders)
- Η μέθοδος `format` λαμβάνει σαν όρισμα μια πλειάδα
  - Αν δεν υπάρχουν νούμερα εντός των `{ }` τα στοιχεία της πλειάδας αντιστοιχίζονται 1-1 με τις θέσεις
  - Αν υπάρχουν νούμερα εντός των `{ }` αναφέρονται στους δείκτες θέσεων της πλειάδας και μπορεί να έχουν οποιαδήποτε σειρά
  - Οι θέσεις που δέχονται τιμές μπορεί να επαναλαμβάνουν τους δείκτες θέσεων της πλειάδας

Εισαγωγή στην Python

Μορφοποιημένη Έξοδος

```
>>> for x in range(1, 11):
...     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

- `{0:2d}`: το 1ο όρισμα είναι αριθμός και εμφανίζεται σε εύρος 2 χαρακτήρων
- `{1:2d}`: το 2ο όρισμα είναι αριθμός και εμφανίζεται σε εύρος 3 χαρακτήρων
- `{2:2d}`: το 3ο όρισμα είναι αριθμός και εμφανίζεται σε εύρος 4 χαρακτήρων

Εισαγωγή στην Python

Μορφοποιημένη Έξοδος

```
>>> '{:,}'.format(1234567890) # Χρησιμοποιεί το , για να χωρίζει τις χιλιάδες
'1,234,567,890'
>>> coord = (3, 5) # Ένα όνομα πλειάδας πρόκειται να περάσει όρισμα στη format
>>> 'X: {0[0]};  Y: {0[1]}'.format(coord)  # Πρόσβαση στις θέσεις του ονόματος
'X: 3;  Y: 5'
>>> '{:<30}'.format('left aligned')
'left aligned                  '   # Αριστερή στοίχιση σε εύρος 30 χαρακτήρων
>>> '{:>30}'.format('right aligned')
'                 right aligned'   # Δεξιά στοίχιση σε εύρος 30 χαρακτήρων
>>> '{:^30}'.format('centered')
'           centered           '   # Στοίχιση στο κέντρο σε εύρος 30 χαρακτήρων
>>> '{:*^30}'.format('centered')
'***********centered***********'   # Χρησιμοποιεί το '*' αντί του ' '
>>> points = 19
>>> total = 22                     # Εμφάνιση ποσοστού με δύο δεκαδικά ψηφία
>>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 86.36%'
>>> # Εμφάνιση του 42 σε διάφορες αριθμιτικές βάσεις:
... 'int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}'.format(42)
'int: 42;  hex: 2a;  oct: 52;  bin: 101010'
```

Εισαγωγή στην Python

Μορφοποιημένη έξοδος σε αρχείο

```
afile = open('output.txt', 'w')

for x in range(1,11):
    afile.write('{0:2d} {1:3d} {2:4d}\n'.format(x, x*x, x*x*x))

afile.close()
```

- `open(output,txt, 'w')` ανοίγει το αρχείο `output.txt` για έξοδο
- Η παράμετρος `'w'` σημαίνει ότι τα περιεχόμενα του `output.txt` θα αντικατασταθούν
- Η μέθοδος `write` "τυπώνει" στο αρχείο
- Το `\n` είναι ο χαρακτήρας "αλλαγή γραμμής"

Εισαγωγή στην Python

Μορφοποιημένη έξοδος σε αρχείο

```
afile = open('output.txt', 'a')

afile.write('{:^3} {:^6} {:^6}\n'.format('x', 'x**2', 'x**3'))
afile.write('=== ====== ======\n')

for x in range(1,11):
    afile.write('{0:3d} {1:6d} {2:6d}\n'.format(x, x*2, x**3))

afile.close()
```

- Η παράμετος `'a'` σημαίνει ότι η έξοδος στο αρχείο θα γίνει μετά την τελευταία γραμμή του (θα διατηρηθούν τα περιεχόμενά του)

Εισαγωγή στην Python

Είσοδος δεδομένων από αρχείο

```
Αρχείο Κόμβων (komvoi.txt)          Αρχείο Ράβδων (ravdoi.txt)
-------------                       -------------
K1 0.0 0.0                          PK1 K1 K2
K2 3.0 0.0                          PK2 K2 K3
K3 6.0 0.0                          PA1 K4 K5
A1 1.5 2.0                          PΔ1 K1 A1
A2 4.5 2.0                          PΔ2 A1 K2
                                    PΔ3 K2 A2
                                    PΔ4 A2 K3
```

```python
kdict={} # Αρχικοποίηση του λεξικού των κόμβων
for line in open('komvoi.txt'): # Άνοιγμα αρχείου για ανάγνωση
    alist = line.strip().split(' ') # Δημιουργεί λίστα με τα "κομμάτια" του line
    name = alist[0]
    coords = (float(alist[1]), float(alist[2]))
    kdict[name] = coords
```

Ερωτήσεις;