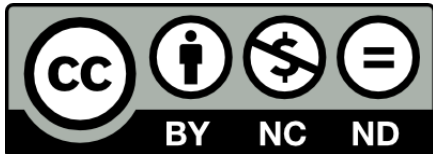


Εισαγωγή στην Python

Scripts for Scrabble

Πακέτο διαφανειών 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, 2021



python

Scripts for Scrabble

Τα περιεχόμενα αυτής της ενότητας σας δίνουν **λύσεις για τον κώδικα** που θα αναπτύξετε στην εργασία 'Scrabble'

Δεν είναι ύλη που εξετάζεται στο Test Python



Περιεχόμενα

- 1) Αρχείο λέξεων της Ελληνικής γλώσσας
 - Από το greek.txt στο greek7.txt
- 2) Δομή δεδομένων για τα γράμματα του παιχνιδιού
- 3) Λέξεις γλώσσας: Οργάνωση σε Λίστα ή Λεξικό;
- 4) Ενδιάμεση αποθήκευση: pickle vs json
- 5) itertools: βιβλιοθήκη για συνδυαστική ανάλυση
- 6) random: βιβλιοθήκη για διαχείριση ψευδοτυχαίων αριθμών
- 7) get: ασφαλής πρόσβαση σε δεδομένα dictionary
- 8) Επικοινωνία κλάσεων μεταξύ τους



1) Αρχείο λέξεων της Ελληνικής γλώσσας

Από το greek.txt στο greek7.txt

- Το λεξικό της γλώσσας που θα χρησιμοποιήσει το Scrabble (δηλ. όλες οι λέξεις που θα γίνονται αποδεκτές και έχουν μέχρι και 7 γράμματα) είναι διαθέσιμο στο αρχείο **greek7.txt** που σας δίνεται έτοιμο
 - Μπορείτε να το κατεβάσετε από τη σελίδα του μαθήματος, στην ενότητα «Scripts for Scrabble - 2022»)

=====

- Οι παρακάτω οδηγίες **ΔΕΝ** αποτελούν μέρος της εξεταστέας ύλης και δίνονται μόνον για εκπαιδευτικούς λόγους για όσους/ες θέλουν να μάθουν (ή να θυμηθούν) πώς χειριζόμαστε ένα αρχείο κειμένου στην Python και πώς μπορούμε να δημιουργήσουμε το greek7.txt από το αρχείο που περιλαμβάνει ΟΛΕΣ τις λέξεις της νέας ελληνικής γλώσσας
- Αν θέλετε να δημιουργήσετε μόνοι σας το αρχείο greek7.txt ακολουθείστε τα παρακάτω βήματα:
- Α) Κατεβάστε από τη σελίδα του μαθήματος το αρχείο **greek.txt** που περιλαμβάνει εκατοντάδες χιλιάδες λέξεις της ελληνικής γλώσσας (και με την κλίση τους) (θα το βρείτε επίσης στην ενότητα «Scripts for Scrabble - 2022»)
- Β) Γράψτε τον κώδικα που δημιουργεί το αρχείο **greek7.txt** που θα πρέπει να περιλαμβάνει μόνο εκείνες τις λέξεις του greek.txt που έχουν μέχρι και 7 γράμματα (σας δίνεται στην επόμενη διαφάνεια)



Κώδικας που δημιουργεί το αρχείο greek7.txt

```
# Δημιουργία greek7.txt
# encoding="utf-8"

with open('greek.txt', 'r', encoding="utf-8") as f:
    with open('greek7.txt', 'w') as f7:
        for line in f:
            if len(line) <= 8:
                f7.write(line)
```

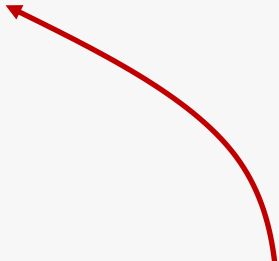
- Αξίζει να προσέξετε τα εξής:
- 1) Το αρχείο λειτουργεί ως «επαναληπτική δομή» (iterable στην εντολή for)
- Έτσι η ανάγνωση δεδομένων γίνεται με την τεχνική της 'ροής' (streaming) και όχι με διάβασμα ολόκληρου του αρχείου στη μνήμη.
- 2) **Παράμετρος 'encoding'**: για να διαβαστούν σωστά οι ελληνικοί χαρακτήρες θα πρέπει να δηλωθεί η κωδικοποίηση Unicode ('utf-8') στην εντολή open με τη χρήση της παραμέτρου 'encoding' όπως βλέπετε
- 3) **Έλεγχος '<=8'**: Ελέγχουμε αν το μήκος γραμμής είναι <=8 (και όχι 7) γιατί στο τέλος κάθε γραμμής υπάρχει χαρακτήρας αλλαγής γραμμής.



UNICODE & encoding

```
# Δημιουργία greek7.txt
# encoding="utf-8"

with open('greek.txt', 'r', encoding="utf-8") as f:
    with open('greek7.txt', 'w') as f7:
        for line in f:
            if len(line) <= 8:
                f7.write(line)
```



- UTF → “Unicode Transformation Format”, ‘8’ → 8-bit για κωδικοποίηση
- Υπάρχουν επίσης τα λιγότερο συχνά χρησιμοποιούμενα UTF-16 και UTF-32
- Κανόνες UTF-8 :
- Για χαρακτήρες με κωδικό < 128, χρησιμοποιείται η αντίστοχη τιμή (1 byte)
- Για χαρακτήρες με κωδικό >= 128, χρησιμοποιούνται ομάδες, 2, 3 ή 4 bytes



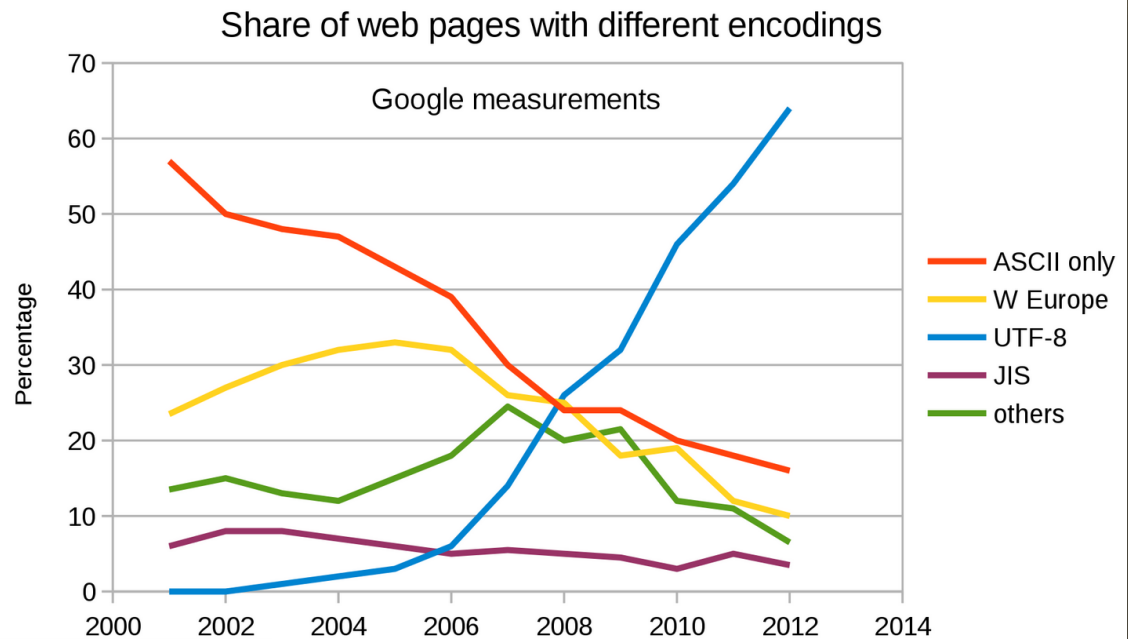
UTF-8 Η βασική δομή του ([Wikipedia](#))

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF ^[12]	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

- Οι πρώτοι 128 characters (US-ASCII) απαιτούν 1 byte.
- Οι επόμενοι 1920 χαρακτήρες κωδικοποιούνται με 2 bytes και περιλαμβάνουν μεταξύ άλλων αλφάβητα όπως: Ελληνικό, Κυριλλικό, Κοπτικό, Αρμένικο, Εβραϊκό, Αραβικό, Συριακό, κλπ.
- Κωδικοί χαρακτήρων στο UTF-8



UTF-8 στο Web



UTF-8	96.8%
ISO-8859-1	1.3%
Windows-1251	0.8%
Windows-1252	0.3%
GB2312	0.2%
Shift JIS	0.1%
ISO-8859-9	0.1%
Windows-1254	0.1%
EUC-KR	0.1%
GBK	0.1%
EUC-JP	0.1%

W3Techs.com, 12 May 2021

Percentages of websites using various character encodings



UnicodeDecodeError

cp1253

		Russian, Serbian
cp1252	windows-1252	Western Europe
cp1253	windows-1253	Greek
cp1254	windows-1254	Turkish
cp1255	windows-1255	Hebrew

- Αν δεν προσδιορίσετε encoding value χρησιμοποιείται το encoding του λειτουργικού (cp1253) οπότε μπορεί να εμφανιστεί **UnicodeDecodeError**
- Καλύτερη πρακτική:
- Ανοίγετε αρχεία ελληνικών με

```
encoding="utf-8")
```



2) Δομή δεδομένων για τα γράμματα του παιχνιδιού

- Μια παρτίδα Scrabble ξεκινά με:
- Α) ένα **συγκεκριμένο αριθμό γραμμάτων** στο «σακουλάκι» (απ' όπου παίρνουν οι παίκτες γράμματα)
- Β) με **καθορισμένη αξία κάθε γράμματος** («πόντοι» που δίνει το κάθε γράμμα όταν συμπεριλαμβάνεται σε μια λέξη)
- Σχεδιάστε μια δομή δεδομένων στον κώδικά σας η οποία να περιλαμβάνει τις δύο παραπάνω πληροφορίες για κάθε γράμμα



Προτεινόμενη δομή

```
lets = {'A': [12, 1], 'B': [1, 8], 'Γ': [2, 4], 'Δ': [2, 4], 'Ε': [8, 1],  
        'Ζ': [1, 10], 'Η': [7, 1], 'Θ': [1, 10], 'Ι': [8, 1], 'Κ': [4, 2],  
        'Λ': [3, 3], 'Μ': [3, 3], 'Ν': [6, 1], 'Ξ': [1, 10], 'Ο': [9, 1],  
        'Π': [4, 2], 'Ρ': [5, 2], 'Σ': [7, 1], 'Τ': [8, 1], 'Υ': [4, 2],  
        'Φ': [1, 8], 'Χ': [1, 8], 'Ψ': [1, 10], 'Ω': [3, 3]  
}
```

- Στο λεξικό lets σε κάθε χαρακτήρα αντιστοιχεί ως τιμή μια λίστα όπου:
 - 1) Πρώτη τιμή = πλήθος γραμμάτων στο παιχνίδι
 - 2) Δεύτερη τιμή = αξία γράμματος στο παιχνίδι
- **Γράψτε τον κώδικα** που υπολογίζει την «αξία» μιας λέξης πχ. 'ΣΚΡΑΜΠΛ' με βάση την παραπάνω αναπαράσταση
- *Η δομή αυτή σας δίνεται στο αρχείο letters.py στην ενότητα «Scripts For Scrabble – 2022»*



3) Λέξεις γλώσσας: Οργάνωση σε Λίστα ή Λεξικό; 1/2

- Ο **έλεγχος εγκυρότητας** της κάθε προτεινόμενης λέξης μπορεί να γίνεται με αναζήτηση στις αποδεκτές λέξεις («Λεξικό γλώσσας»)
- Τι **δομή** θα χρησιμοποιήσετε για να έχετε διαθέσιμες τις αποδεκτές λέξεις του αρχείου greek7.txt στη διάρκεια του παιχνιδιού;
- Ο κώδικάς σας θα πρέπει να «**συμβουλευέται**» αυτή τη δομή ώστε να αποφασίζει:
 - Αν η λέξη που έχει παίξει ο παίκτης είναι αποδεκτή
 - Ποια λέξη θα παίξει ο υπολογιστής
- Μια δομή δεδομένων που μπορεί να χρησιμοποιηθεί είναι η λίστα.

-
- Σαν άσκηση: **Γράψτε κώδικα** που να διαβάσει τις λέξεις από το αρχείο greek7.txt και να τις **τοποθετεί στις θέσεις μια λίστας**.
 - Προσοχή στη διαχείριση του συμβόλου αλλαγής γραμμής '\n'



```
# Read from greek7.txt
with open('greek7.txt','r') as f7:
    words = f7.readlines()
    for index, i in enumerate(words):
        words[index] = i.strip('\n')
```

```
words = []
with open('greek7.txt','r') as f7:
    for line in f7:
        words.append(line.strip('\n'))
```



3) Λέξεις γλώσσας: Οργάνωση σε Λίστα ή Λεξικό; 2/2

- Σκεφθείτε επίσης τη μορφή ενός Λεξικού (dictionary) που θα ήταν **αποδοτικό** να χρησιμοποιηθεί στο παιχνίδι.
 - Δηλ. ποια κλειδιά και ποιες τιμές θα είχε ένα τέτοιο λεξικό;
 - -----
 - *Σαν άσκηση: Γράψτε* κώδικα που να διαβάζει τις λέξεις από το αρχείο greek7.txt (ή αν θέλετε από τη λίστα words που κατασκευάσατε στο βήμα 3) και να **δημιουργεί το λεξικό** (dict) που σχεδιάσατε.
- Τί **δομή** θα **προτιμήσετε**; Λεξικό ή Λίστα;
 - *Κριτήριο 2:* Εξηγήστε ποια είναι η δομή που χρησιμοποιήσατε για να οργανώσετε τις λέξεις της γλώσσας στον κώδικά σας



4) Ενδιάμεση αποθήκευση: **pickle** vs **json**

- Για να κάνετε **ενδιάμεση αποθήκευση** των δεδομένων του παιχνιδιού (π.χ. για να κρατήσετε στατιστικά για τους παίκτες που παίζουν ή για να αποθηκεύσετε τη δομή λίστας/λεξικού που έχει τις λέξεις της γλώσσας) μπορείτε να χρησιμοποιήσετε:
 - 1) είτε αρχείο **pickle**, 2) είτε αρχείο **json**
 - -----
 - *Σαν άσκηση:*
 - Α) Γράψτε τον **κώδικα** που **αποθηκεύει** τη δομή σε αρχείο pickle
 - Β) Γράψτε τον **κώδικα** που **διαβάζει** το αρχείο τύπου pickle (διαβάζονται σωστά τα δεδομένα σας;)
 - -----
 - Γ) Κάντε τα δύο προηγούμενα βήματα για αρχείο json
 - Δ) Τί **αρχείο** θα **προτιμήσετε** για αποθήκευση; pickle ή json;
 - Δείτε σε επόμενες διαφάνειες τα μειονεκτήματα/πλεονεκτήματα κάθε μορφής




```
# pickle the newly created dictionary
import pickle
with open('word_dict.pkl', 'wb') as f:
    pickle.dump(wdict, f)
```

```
# ..or save as a json file
import json
with open('word_dict.json', 'w') as f:
    json.dump(wdict, f)
```

```
# import word dict from pickle
with open('word_dict.pkl', 'rb') as f:
    word_dict = pickle.load(f)

print(word_dict['ENA'])
```

```
# import word dict from json
with open('word_dict.json', 'r') as f:
    word_dict = json.load(f)

print(word_dict['ENA'])
```

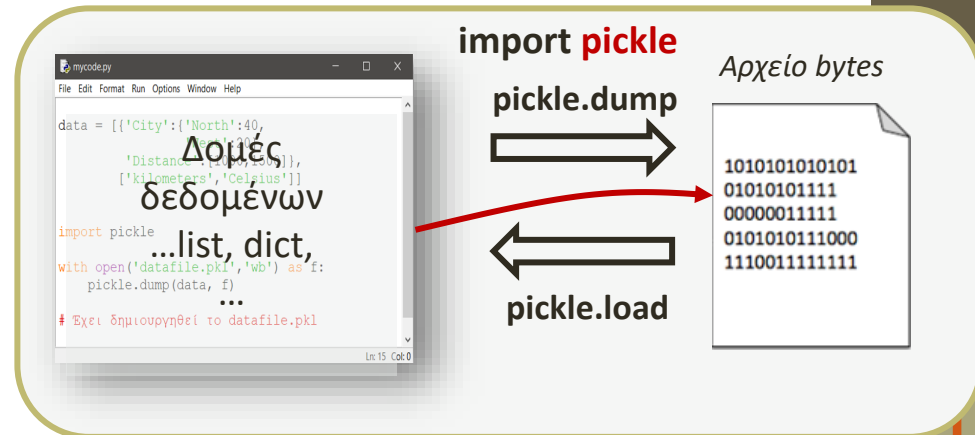
«Διατήρηση» (pickling): τι εννοούμε;

```
data = [{ 'City': { 'North': 40,  
                  'West': 20 },  
         'Distance': [1000, 1500] },  
        [ 'kilometers', 'Celsius' ] ]
```

- Πώς αποθηκεύουμε μια σύνθετη δομή 'φωλιασμένων' δεδομένων;

~~f.write(str(data))~~

- **Διατήρηση** (pickling): αποθήκευση δομών δεδομένων με τεχνική 'σειριοποίησης' (serialization)
- **Serialization**: Δομή δεδομένων σε ροή bytes
- **Deserialization**: Ροή bytes σε δομή δεδομένων



import pickle & dump

The screenshot shows a Python IDE window titled 'mycode.py'. The code defines a dictionary 'data' and uses 'pickle.dump' to save it to 'datafile.pkl'. Annotations include a red dashed arrow pointing from the word 'binary' to the file mode 'wb', and a green arrow pointing from the file path 'datafile.pkl' to a file explorer window. The file explorer shows 'datafile.pkl' as a PKL File and 'mycode.py' as a Python File. A 'Run Module F5' button is also visible.

```
data = [{'City':{'North':40,
                  'West':20},
         'Distance':[1000,1500]},
        ['kilometers','Celsius']]

import pickle

with open('datafile.pkl','wb') as f:
    pickle.dump(data, f)
```

Annotations:

- Red dashed arrow from **binary** to `'wb'`
- Green arrow from `'datafile.pkl'` to file explorer
- File Explorer 1: mycode.py (Python File)
- File Explorer 2: datafile.pkl (PKL File), mycode.py (Python File)
- Run Module F5 button

dump → serialization & αποθήκευση



import pickle & load

The screenshot shows a Python IDE window titled 'mycode2.py' with the following code:

```
import pickle
with open('datafile.pkl', 'rb') as f:
    data = pickle.load(f)
print(data)
```

Annotations include:

- A red dashed arrow pointing from the `'rb'` mode in `open()` to a label **binary**.
- A green arrow pointing from the `print(data)` statement to a text box containing the output data.
- A file explorer window is open, showing a list of files: `datafile.pkl` (PKL File), `mycode.py` (Python File), and `mycode2.py` (Python File).
- A 'Run Module F5' button is visible below the file explorer.

The output data shown in the text box is:

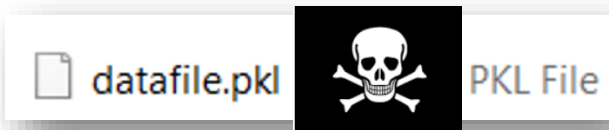
```
[{'City': {'North': 40, 'West': 20}, 'Distance': [1000, 1500]},  
 ['kilometers', 'Celsius']]
```

load → ανάγνωση & deserialization



Προσοχή στην pickle!

- Ανοίξτε μόνο pickle αρχεία που **εμπιστεύεστε!**



→
`pickle.load`

```
mycode.py
File Edit Format Run Options Window Help

data = [{'City':{'North':40,
                'West':20},
        'Distance':[1000,1500]},
        ['kilometers','Celsius']]

import pickle

with open('datafile.pkl','wb') as f:
    pickle.dump(data, f)

# Έχει δημιουργηθεί το datafile.pkl

Line 15 Col 0
```

- Αλλιώς **κακόβουλος** κώδικας μέσω του pickle μπορεί να 'τρέξει' στο πρόγραμμά σας



import json

dump → serialization & αποθήκευση

Name	Type
datafile.pkl	PKL File
datajs.json	JSON File
mycode.py	Python File
mycode2.py	Python File
myjsoncode.py	Python File

load → διάβασμα & deserialization

The screenshot shows a Python IDE window titled 'myjsoncode.py' with the following code:

```
import json

data = [{'City': {'North': 40, 'West': 20}, 'Distance': [1000, 1500]}, ['kilometers', 'Celsius']]

with open('datajs.json', 'w') as fout:
    json.dump(data, fout)
```

Annotations include a black arrow pointing to the 'import json' line, a blue 'Run Module F5' button, and a black arrow pointing to the 'json.dump' line. A red arrow points from the 'dump' text to the 'json.dump' line. Below the IDE, a file explorer shows a list of files, with 'datajs.json' highlighted. A red arrow points from the 'load' text to the 'datajs.json' file. To the right of the file explorer, a Notepad window titled 'datajs.json' shows the serialized JSON output:

```
[{"City": {"North": 40, "West": 20}, "Distance": [1000, 1500]}, ["kilometers", "Celsius"]]
```

The status bar at the bottom right of the IDE shows 'Ln: 28 Col: 0'.



python

5) **itertools**: βιβλιοθήκη για επαναληπτικές διαδικασίες (iterations)

- Πώς «**κατασκευάζει**» λέξεις ο υπολογιστής;
- Θα πρέπει να δημιουργούνται όλες οι **δυνατές μεταθέσεις** (permutations) των γραμμάτων που έχει ο υπολογιστής και να ελέγχεται για κάθε μετάθεση **αν αποτελεί αποδεκτή λέξη**.
- Στη συνέχεια ο κώδικας με βάση τον αλγόριθμο παιχνιδιού θα πρέπει να αποφασίζει αν η λέξη θα παιχτεί ή αν θα γίνει κάτι άλλο
- -----
- Θα χρησιμοποιήσουμε τη βιβλιοθήκη **itertools** η οποία περιέχει διάφορες συναρτήσεις για επαναληπτικές διαδικασίες (iterations)

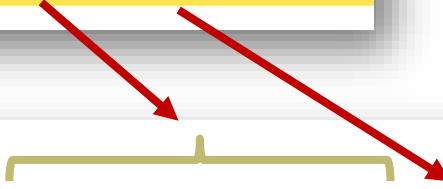


itertools.permutations

- Η `itertools.permutations` παρουσιάζει όλες τις δυνατές **μεταθέσεις** (permutations) των στοιχείων που περιλαμβάνονται σε μια επαναληπτική δομή ανά `r`

```
itertools.permutations(iterable, r=None)
```

```
>>> import itertools
>>> itertools.permutations(['A', 'B', 'C'], 2)
<itertools.permutations object at 0x039C1840>
```



```
>>> perms = itertools.permutations(['A', 'B', 'C'], 2)
>>> list(perms)
[('A', 'B'), ('A', 'C'), ('B', 'A'), ('B', 'C'), ('C', 'A'), ('C', 'B')]
```


itertools.permutations

Παράδειγμα

- Γράψτε κώδικα που να εμφανίζει τις **μεταθέσεις** 4^{ω} γραμμάτων μια λίστας (όποια γράμματα θέλετε) ανά k

```
import itertools as it

somelets = ['Α', 'Δ', 'Κ', 'Σ']

for i in it.permutations(somelets, 2):
    print(i)
```

- Γιατί μας ενδιαφέρουν οι **μεταθέσεις** (permutations) και όχι οι συνδυασμοί (combinations) των γραμμάτων;



Iterable vs. iterator **iteration**

1/3

- **Iterable**: ένα αντικείμενο που περιλαμβάνει τη μέθοδο

`__iter__`

```
>>> mylist = [1,2,3]
>>> dir(mylist)
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
 '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
 '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__',
 '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__re
duce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',
 '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', '
copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```



python

Iterable vs. iterator

2/3

- Όταν κληθεί η `iter` παράγει ένα **αντικείμενο iterator** που «γνωρίζει» πώς να καλέσει ένα προς ένα τα απλά στοιχεία που περιλαμβάνει η δομή `iterable`

```
>>> itor = iter(mylist)
>>> itor
<list_iterator object at 0x04096AD0>
```

- Ο `iterator` περιλαμβάνει μέθοδο `__next__` η οποία κάθε φορά που καλείται επιστρέφει το επόμενο στοιχείο της `iterable` δομής
- Στο τέλος επιστρέφει `'StopIteration'`

```
>>> next(itor)
1
>>> next(itor)
2
>>> next(itor)
3
>>> next(itor)
Traceback (most recent call last):
  File "<pyshell#53>", line 1, in <module>
    next(itor)
StopIteration
>>>
```

Iterable vs. iterator

3/3

- Η permutations (και άλλες συναρτήσεις της itertools) δημιουργούν αντικείμενα iterators

```
>>> import itertools as it
>>>
>>> perm = it.permutations(['A', 'B'])
>>> next(perm)
('A', 'B')
>>> next(perm)
('B', 'A')
>>> next(perm)
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    next(perm)
StopIteration
```



6) **random**: βιβλιοθήκη για διαχείριση ψευδο-τυχαίων αριθμών

- Σε πολλές περιπτώσεις η δημιουργία και διαχείριση ψευδοτυχαίων αριθμών είναι σημαντική στην επίλυση ενός προβλήματος
- Στην περίπτωση του Scrabble θα χρειαστεί να χειριστείτε με (ψευδο-) τυχαίο τρόπο τα δεδομένα που υπάρχουν στη δομή τύπου SakClass δηλ. στο «σακουλάκι» του παιχνιδιού.
- Για να το πετύχετε αυτό χρησιμοποιήστε τα εργαλεία που σας δίνει η βιβλιοθήκη **random** (τεκμηρίωση [εδώ](#))



Παράδειγμα με `random.randint`, `shuffle` & `sample`

- **randint**: Δημιουργήστε μια **λίστα με 10 ψευδοτυχαίους ακέραιους** στο διάστημα `[1,20]` (με την τεχνική «περιγραφής» - list comprehension και χρήση της `randint`) και εμφανίστε την στην οθόνη
- **shuffle**: Χρησιμοποιήστε τη μέθοδο **shuffle** για να «**τυχαιοποιήσετε**» τη λίστα (εμφανίστε την πάλι στην οθόνη)
- **sample**: Χρησιμοποιήστε τη μέθοδο **sample** για να πάρετε ένα **τυχαίο δείγμα N=7** στοιχείων από τη λίστα σας (εμφανίστε το δείγμα στην οθόνη)
- Σκεφθείτε πώς θα μπορούσατε να χρησιμοποιήσετε τις μεθόδους `randint`, `shuffle` & `sample` (ίσως και άλλες της `random`) στον κώδικα του παιχνιδιού Scrabble



```
# How to "randomize" a list
```

```
# import random library
```

```
import random as rn
```

```
# generate mylist with 10 pseudo-random ints from 1 to 20
```

```
mylist = [rn.randint(1,20) for i in range(10)]
```

```
print(mylist)
```

```
# shuffle mylist
```

```
rn.shuffle(mylist)
```

```
print(mylist)
```

```
# get a random sample of k items from mylist
```

```
sample = rn.sample(mylist,4)
```

```
print(sample)
```

```
[17, 10, 10, 5, 6, 13, 1, 5, 15, 2]
```

```
[5, 13, 6, 1, 15, 17, 10, 10, 5, 2]
```

```
[10, 5, 13, 15]
```



7) **get**: ασφαλής πρόσβαση σε δεδομένα dictionary

- Κατά την πρόσβαση σε δεδομένα dictionary **αν το κλειδί δεν υπάρχει** τότε σημειώνεται «εξαίρεση» (exception) και το πρόγραμμα σταματά
- Για να αποφύγετε τη διακοπή χρησιμοποιήστε τη μέθοδο **get()** για πρόσβαση στα δεδομένα λεξικού
- Σύνταξη: **dict.get(key[, value])**
- Όπου **value**: η τιμή που επιστρέφει το λεξικό αν δεν υπάρχει το κλειδί (optional)
- # default τιμή της value: **None**
- Δείτε στην επόμενη διαφάνεια παραδείγματα χρήσης




```
# Safe access to dictionary data

di = {1:'ένα', 2:'δύο', 3:'τρία'}
print(di[1])

# but...
# print(di[4])
# ... raises exception

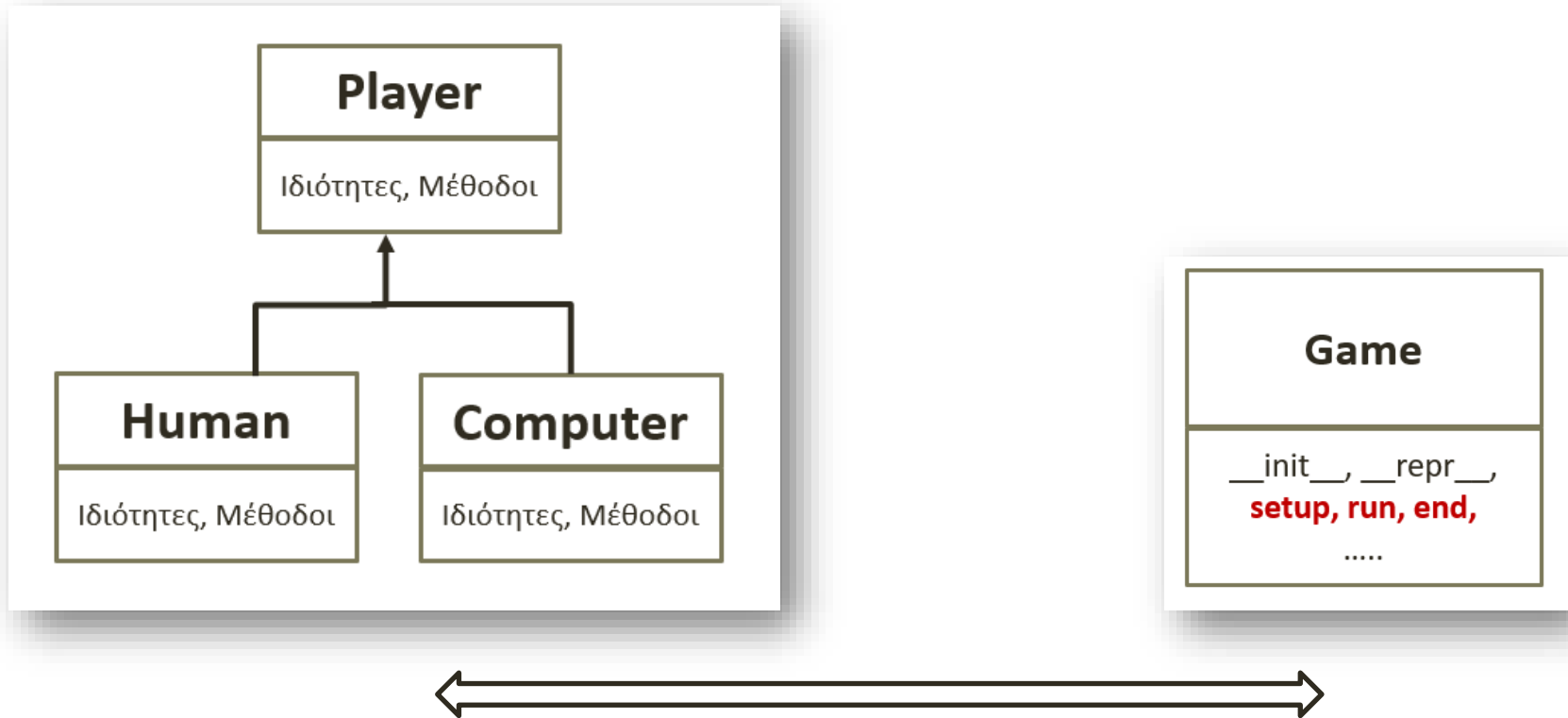
# Use 'get' instead to access data in a dictionary
# syntax: dict.get(key[, value]) ...
# value: the value to return (optional) if the key is not found
# default value: None

print(di.get(2))
print(di.get(4))
print(di.get(15, "Το κλειδί αυτό δεν υπάρχει"))
```

```
ένα
δύο
None
Το κλειδί αυτό δεν υπάρχει
```



8) Επικοινωνία κλάσεων μεταξύ τους



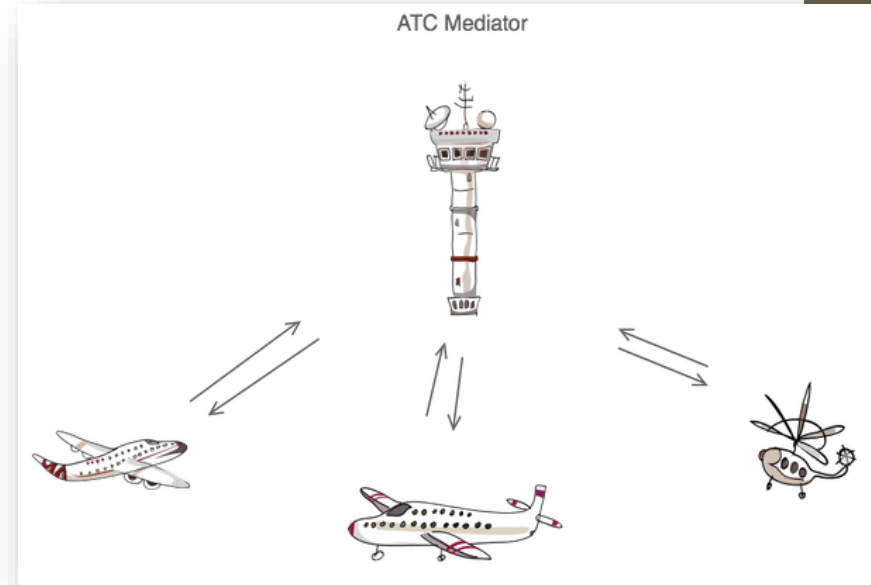
- Πώς επικοινωνούν τα αντικείμενα των κλάσεων μεταξύ τους;



“Mediator” software design pattern

(πρότυπο/υπόδειγμα σχεδίασης λογισμικού)

- Η ανεξάρτητη επικοινωνία πολλών αντικειμένων μεταξύ τους μπορεί να δημιουργήσει «χάος» στον κώδικα
- Σε μια τέτοια περίπτωση ένα αντικείμενο που παίζει το ρόλο του ενδιάμεσου «ρυθμιστή» (Mediator) λύνει ιδανικά το πρόβλημα
- *Κεντρική ιδέα:* Κάθε ανεξάρτητο αντικείμενο «επικοινωνεί» με τον Mediator που συντονίζει την εξέλιξη των ενεργειών τους



- **Software design pattern:** μια επαναχρησιμοποιούμενη λύση (λύση-μοτίβο) για ένα είδος συχνά εμφανιζόμενου προβλήματος



Η κλάση Game ως «ρυθμιστής» (Mediator)

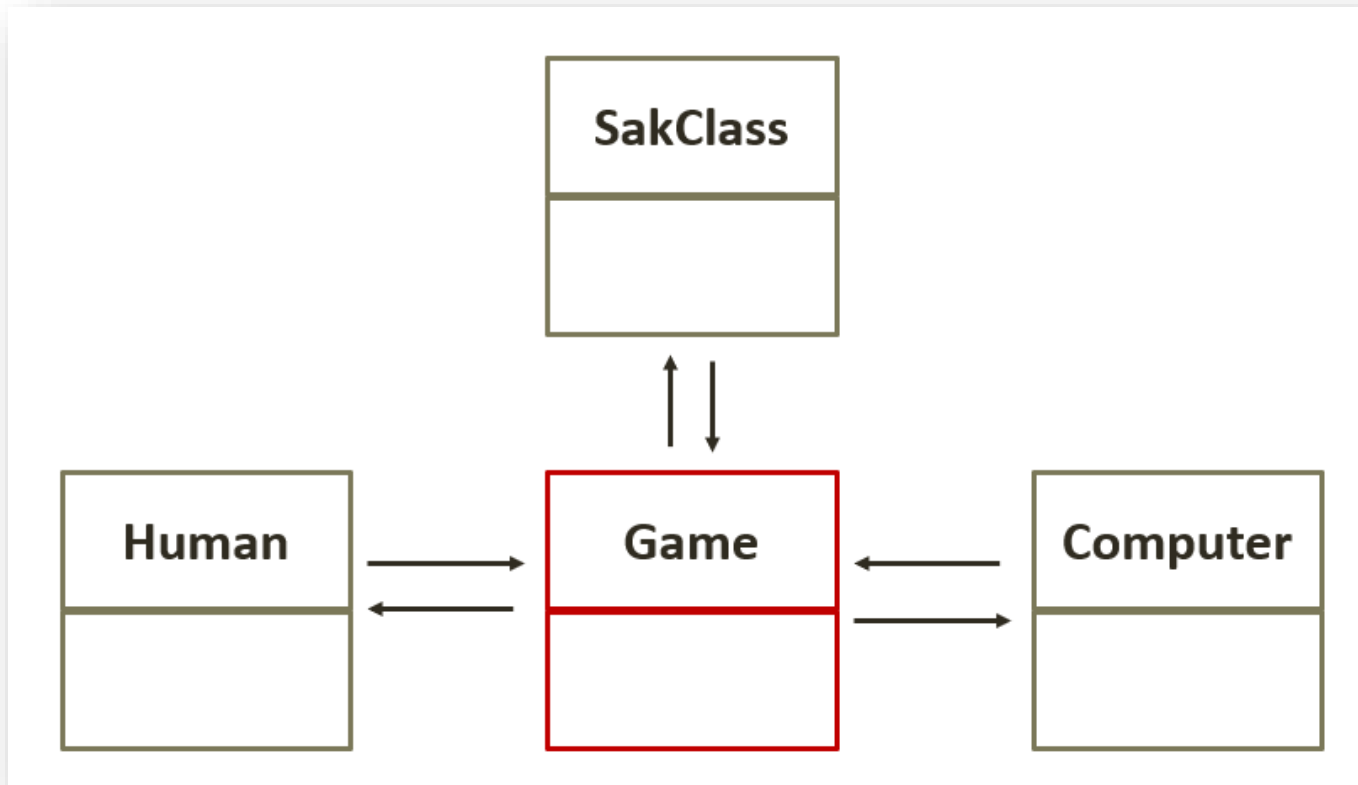
- Ο Mediator «γνωρίζει» τα αντικείμενα που συντονίζει
- Κατασκευάζονται στον κατασκευαστή του mediator
- Στη συνέχεια οι μέθοδοι του Mediator (στο παράδειγμά μας είναι η κλάση Game) μπορούν να χειρίζονται/καλούν τις μεθόδους των άλλων αντικειμένων ώστε να παίρνουν ή να τους περνάνε πληροφορία



```
# --- Game -----  
class Game:  
    def __init__(self):  
        self.ph = Human()  
        self.pc = Computer()  
        self.sak = SakClass()  
  
    def __repr__(self):  
        return 'Game instance'  
  
    def setup(self):  
        pass  
  
    def run(self):  
        pass  
  
    def end(self):  
        pass
```



Η κλάση Game ως «ρυθμιστής» (Mediator)



- Γράψτε την κλάση 'Game' ως «ρυθμιστή» που διαχειρίζεται την επικοινωνία μεταξύ των αντικειμένων των άλλων κλάσεων



Σύνοψη

- 1) Αρχείο λέξεων της Ελληνικής γλώσσας
 - Από το greek.txt στο greek7.txt
- 2) Δομή δεδομένων για τα γράμματα του παιχνιδιού
 - Ποιά δομή θα επιλέξετε για να διαχειριστείτε τα διαθέσιμα γράμματα του παιχνιδιού;
- 3) Λέξεις γλώσσας: Οργάνωση σε Λίστα ή Λεξικό;
 - Θα διαχειριστείτε τις λέξεις της γλώσσας σε λίστα ή λεξικό (dictionary);
- 4) Ενδιάμεση αποθήκευση: pickle vs json
 - Όπου χρειάζεστε ενδιάμεση αποθήκευση χρησιμοποιήστε τεχνική pickle ή json
- 5) itertools: βιβλιοθήκη για συνδυαστική ανάλυση
 - Για αλγόριθμο συνδυαστικής ανάλυσης χρησιμοποιήστε τη βιβλιοθήκη itertools
- 6) random: βιβλιοθήκη για διαχείριση ψευδοτυχαίων αριθμών
 - Για παραγωγή και διαχείριση ψευδοτυχαίων αριθμών
- 7) get: ασφαλής πρόσβαση σε δεδομένα dictionary
 - Για να κάνετε 'ασφαλή' πρόσβαση στα δεδομένα dictionary (δηλ. να μην σταματά η εκτέλεση του κώδικα αν κάποιο κλειδί που ψάχνετε δεν περιλαμβάνεται στο dictionary)
- 8) Επικοινωνία κλάσεων μεταξύ τους
 - Πρότυπο λογισμικού (software pattern) για να οργανώσετε την επικοινωνία των κλάσεων κατά την εκτέλεση του παιχνιδιού

