

Πληροφορική II

Python

Εισαγωγή

- Διαδικαστικά μαθήματος
- Υλικό και Λογισμικό
- Αποθήκευση Δεδομένων
- Προγράμματα
- Python

Διαδικαστικά

- Στόχος του μαθήματος
- Προαπαιτούμενα
- Μέθοδος διδασκαλίας
- Μέθοδος αξιολόγησης
- Βιβλιογραφία

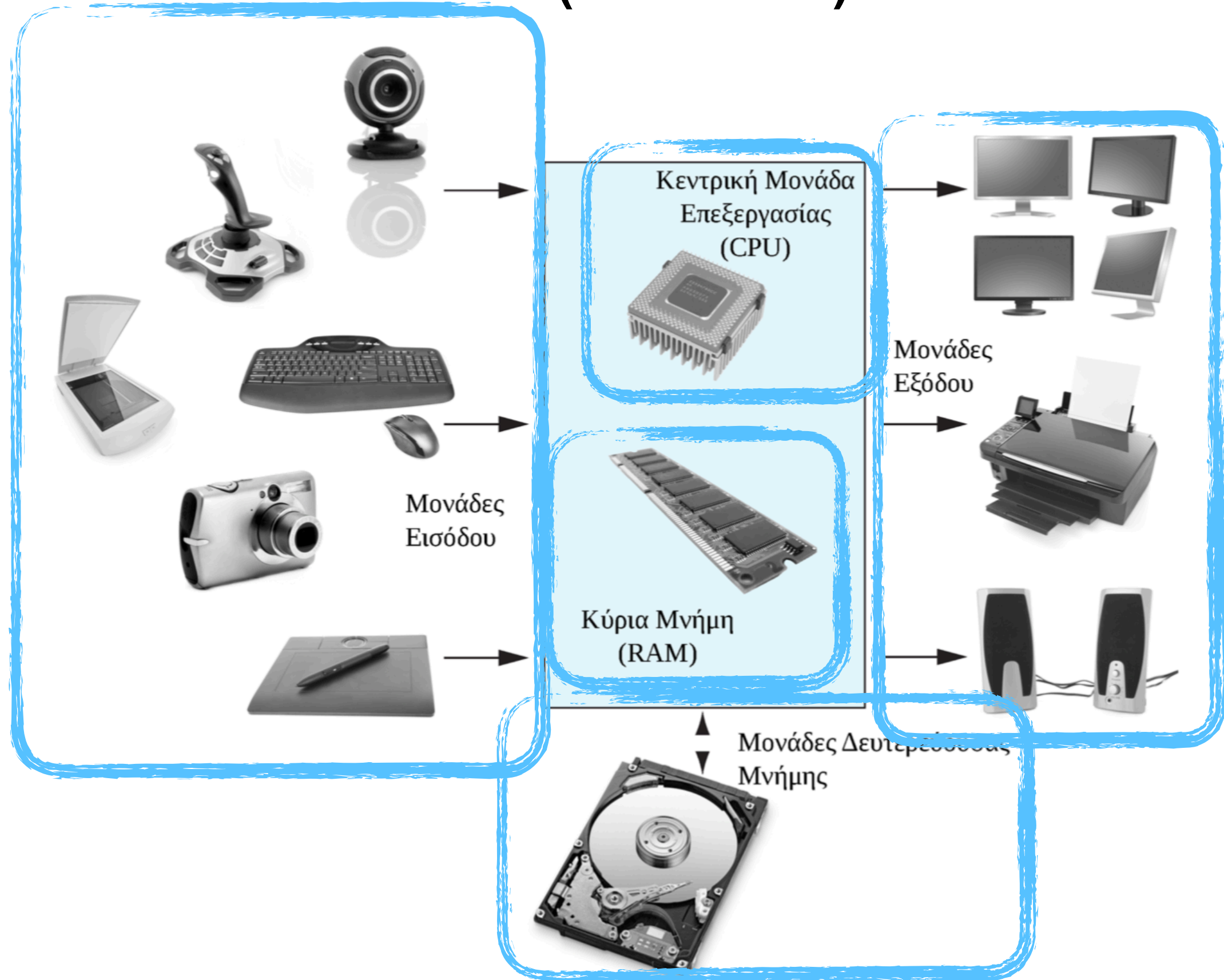
Προτεινόμενα Συγγράμματα στον Εύδοξο

- Tony Gaddis: *Ξεκινώντας με την Python*, Εκδόσεις Da Vinci (Έκδοση του 2020)
- John Guttag: *Υπολογισμοί και προγραμματισμός με την Python*, Εκδόσεις Κλειδάριθμος

Επιπλέον βιβλιογραφία

- Αβούρης Ν., Κούκias Μ., Παλιούρας Β., Σγάρμπας Κ.: *Python, Εισαγωγή στους υπολογιστές*, Πανεπιστημιακές Εκδόσεις Κρήτης
- Σαμαράς Νικόλαος , Τσιπλίδης Κωνσταντίνος: *Το βιβλίο της PYTHON*, Εκδόσεις Κριτική
- Allen B. Downey: *Σκέψου σε Python*, Εκδόσεις Κλειδάριθμος
- Eric Matthes: *Η γλώσσα προγραμματισμού Python μέσα από παραδείγματα και ασκήσεις*, Εκδόσεις Δίσιγμα

Υλικό (hardware)



Λογισμικό (software)

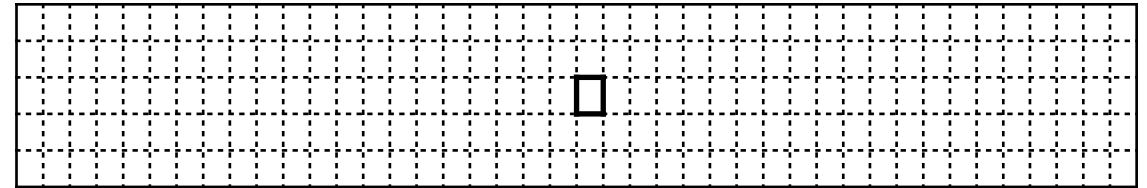
Λογισμικό Συστήματος

- Λειτουργικό σύστημα
Mac OS, Windows, Linux...
- Βοηθητικά προγράμματα
Antivirus, Zip, Drivers...
- Εργαλεία ανάπτυξης λογισμικού
Assembler, Compiler, Interpreter...

Εφαρμογές



Αποθήκευση Δεδομένων



RAM
(Random Access Memory)

byte

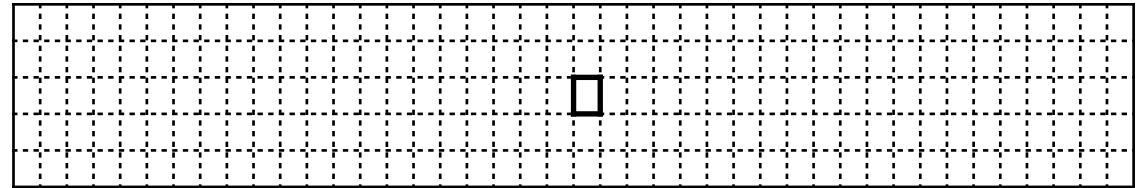
A ή **157**

byte:

1	2	3	4	5	6	7	8
1	0	0	1	1	1	0	1

bit

Αποθήκευση Δεδομένων



RAM
(Random Access Memory)

byte

A ή **157**

byte:

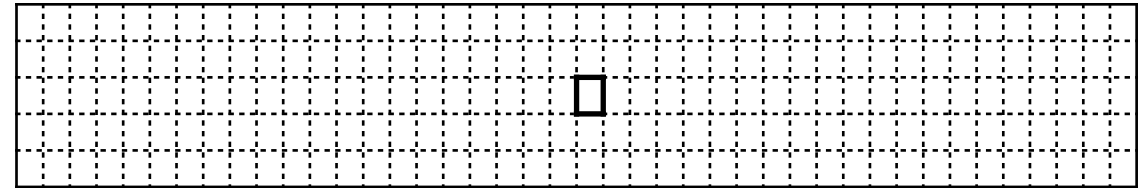
1	2	3	4	5	6	7	8
on	off	off	on	on	on	off	on

bit

Αποθήκευση Δεδομένων



RAM
(Random Access Memory)



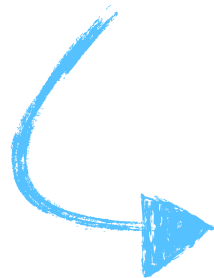
byte

A ή **157**

byte:

7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	1

bit

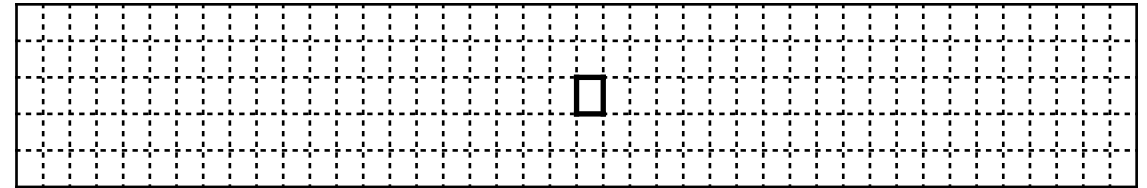


$$2^7 + 2^4 + 2^3 + 2^2 + 2^0 = 157$$

Αποθήκευση Δεδομένων



RAM
(Random Access Memory)



byte

A ή **157**

byte:

7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1



65 ή **A (ASCII)**

Πως ξεχωρίζει η CPU αν είναι αριθμός ή χαρακτήρας;

Αποθήκευση Δεδομένων

Άλλοι τύποι δεδομένων:

- **Αρνητικοί αριθμοί**

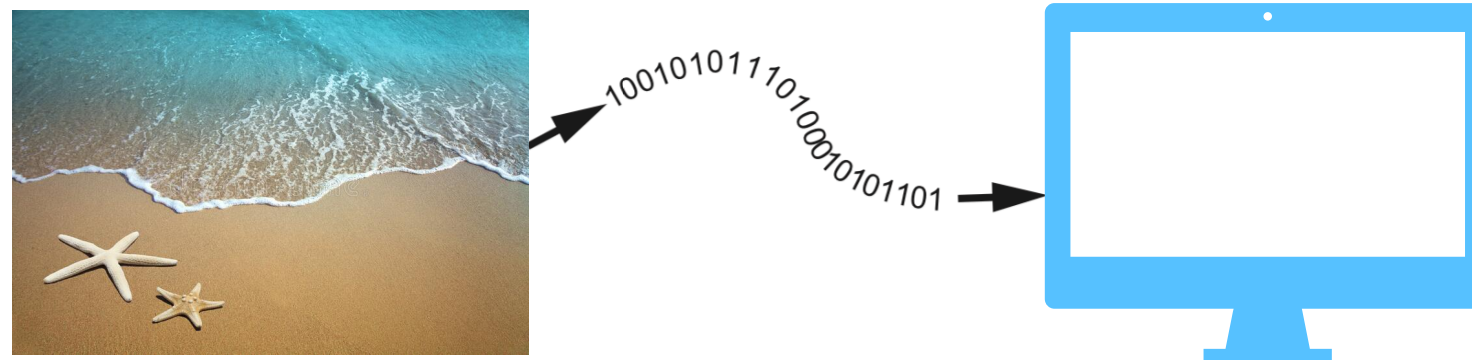
“Συμπλήρωση ως προς το δύο”

- **Προσεγγίσεις πραγματικών αριθμών**

“Αριθμοί κινητής υποδιαστολής”

- **Ψηφιακά δεδομένα**

Αρχεία ήχου, αρχεία φωτογραφίας κλπ.



Προγράμματα

Αλγόριθμος: Μια πεπερασμένη ακολουθία (αυστηρά καθορισμένων) απλών οδηγιών που διεκπεραιώνουν κάποια εργασία.

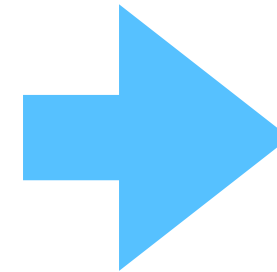
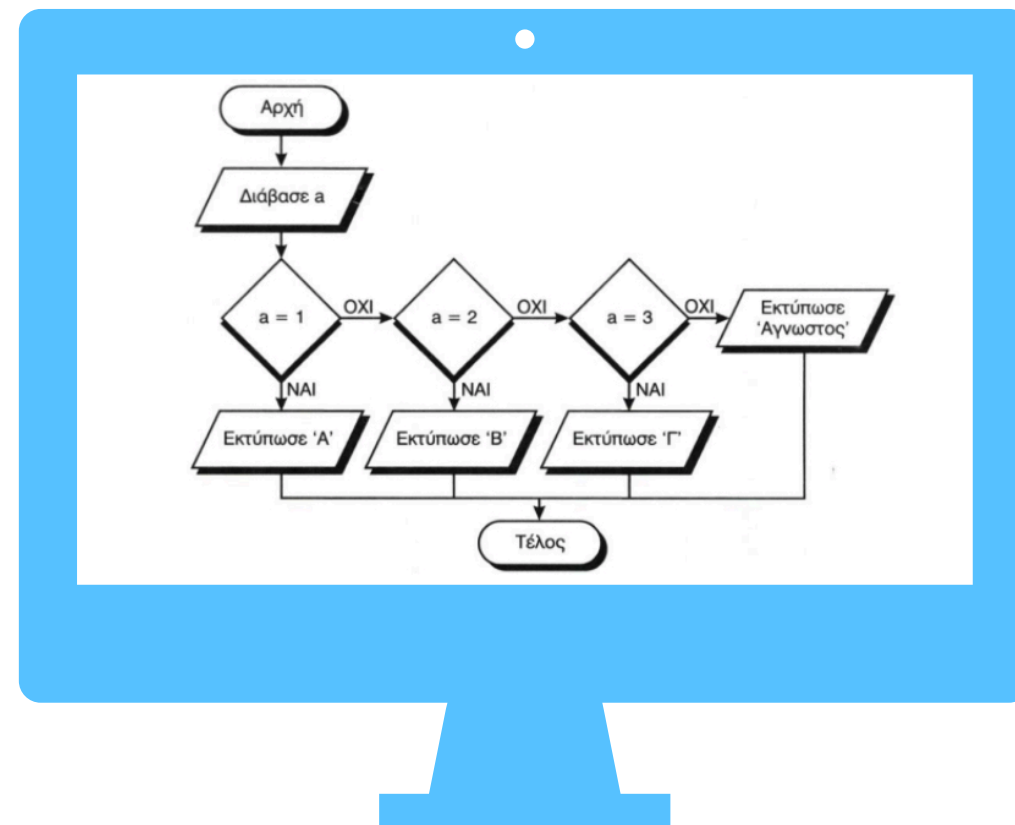
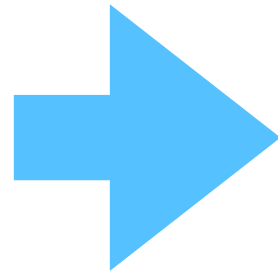


Ψευδοκώδικας

Προγράμματα

Αλγόριθμος: Μια πεπερασμένη ακολουθία (αυστηρά καθορισμένων) απλών οδηγιών που διεκπεραιώνουν κάποια εργασία.

Είσοδος

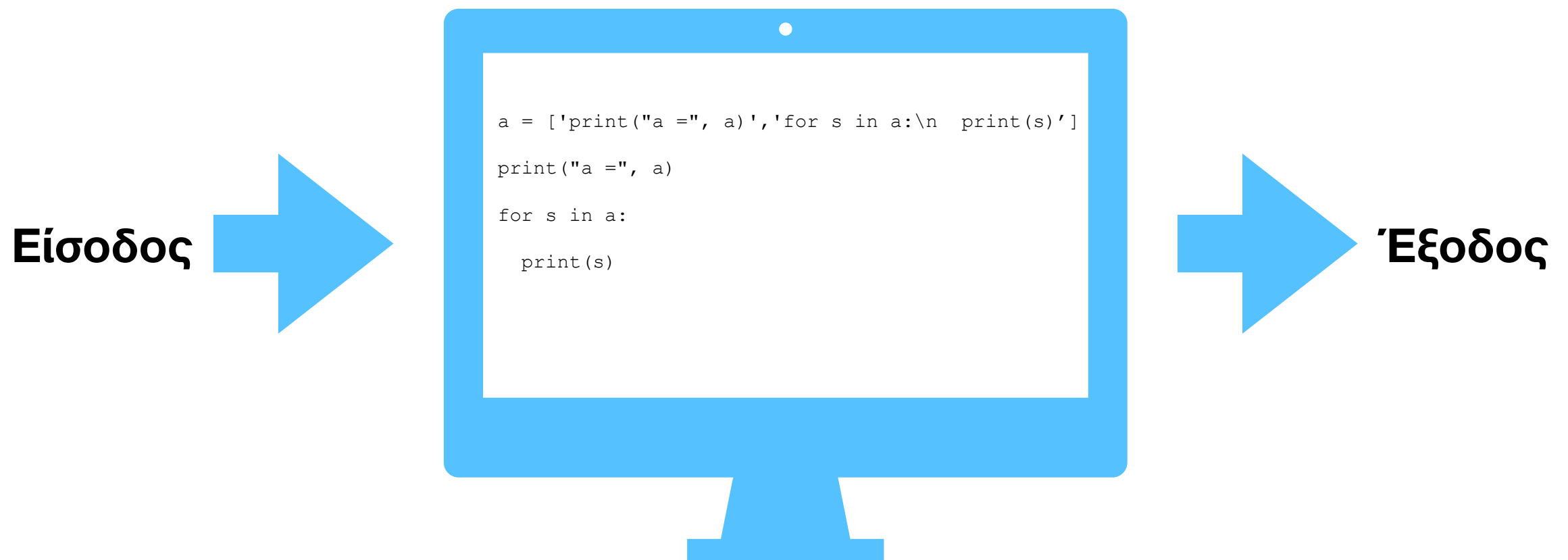


Έξοδος

Διάγραμμα ροής

Προγράμματα

Αλγόριθμος: Μια πεπερασμένη ακολουθία (αυστηρά καθορισμένων) απλών οδηγιών που διεκπεραιώνουν κάποια εργασία.



Πρόγραμμα: Ακριβής διατύπωση του αλγορίθμου σε μια γλώσσα προγραμματισμού

Πως Δουλεύει Ένα Πρόγραμμα



CPU

Λειτουργίες:

- Διαβάζει ένα τμήμα δεδομένων από τη RAM
- Κάνει απλές αριθμητικές πράξεις μεταξύ δύο αριθμών
- Μεταφέρει ένα τμήμα δεδομένων από μία θέση της RAM σε μία άλλη
- Ελέγχει αν μία τιμή είναι ίση με κάποια άλλη

Γλώσσα Μηχανής:

Για κάθε λειτουργία υπάρχει και μία εντολή
(μία ακολουθία από 0 και 1)

Πρόγραμμα

10100001

10111000

10011110

...

Πως Δουλεύει Ένα Πρόγραμμα



CPU

Λειτουργίες:

- Διαβάζει ένα τμήμα δεδομένων από τη RAM
- Κάνει απλές αριθμητικές πράξεις μεταξύ δύο αριθμών
- Μεταφέρει ένα τμήμα δεδομένων από μία θέση της RAM σε μία άλλη
- Ελέγχει αν μία τιμή είναι ίση με κάποια άλλη

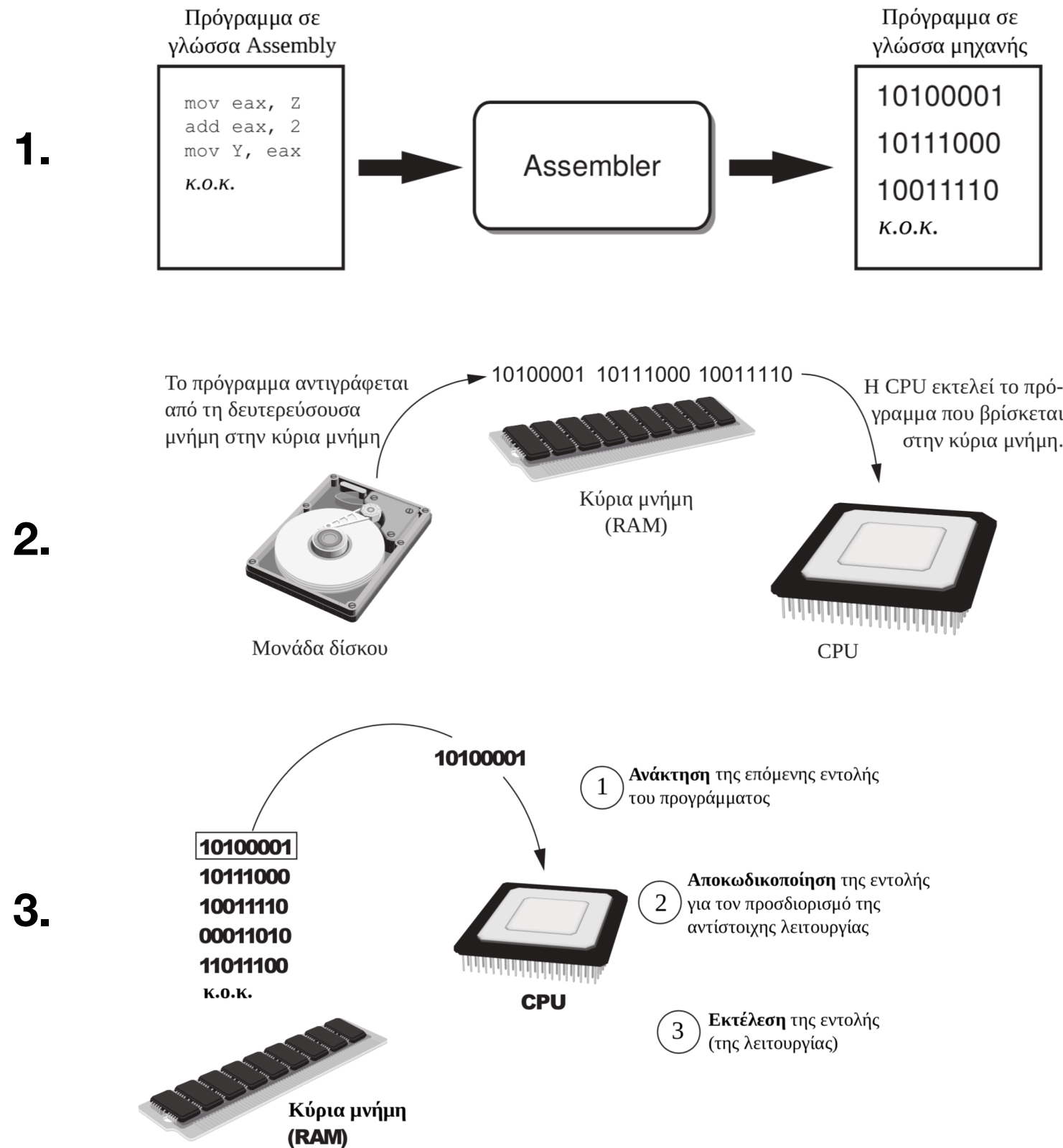
Assembly: Για κάθε λειτουργία υπάρχει και μία “μνημονική εντολή” (add, mul, mov κλπ.)

Πρόγραμμα

```
mov eax, Z  
add eax, 2  
mov Y, eax
```

...

Πως Δουλεύει Ένα Πρόγραμμα



Γλώσσες Υψηλού Επιπέδου

- **Λέξεις κλειδιά**
(if, while, for κλπ.)
- **Τελεστές**
(+, -, = κλπ.)
- **Σύνταξη**

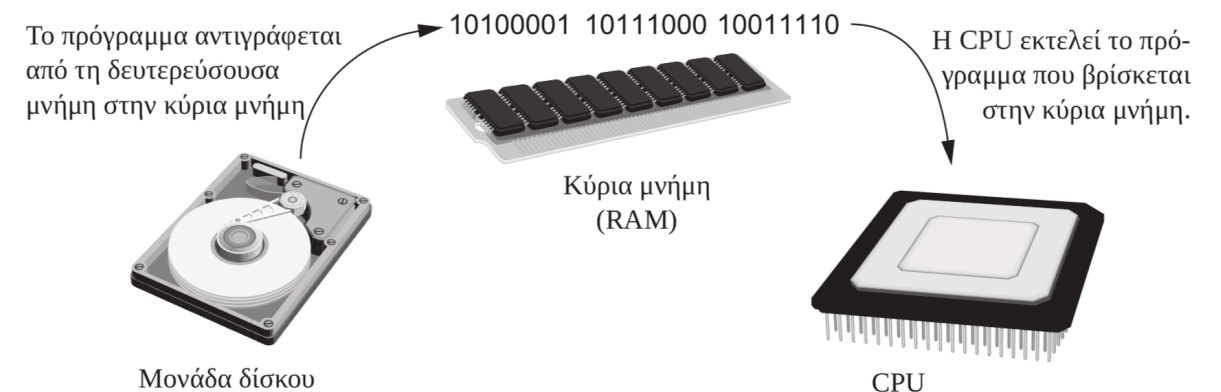


```
a = ['print("a =", a)', 'for s in a:\n    print(s)']  
print("a =", a)  
for s in a:  
    print(s)
```

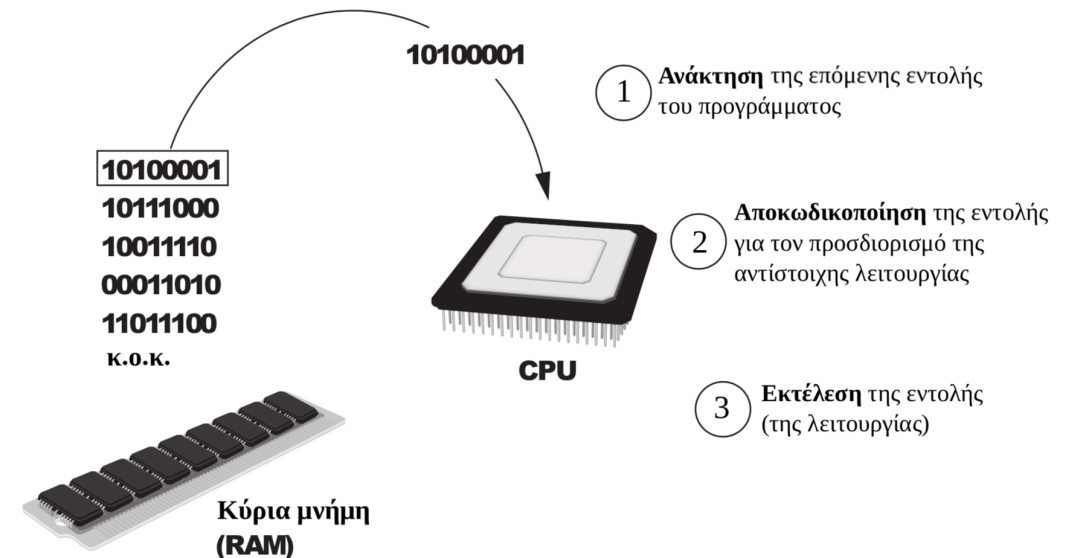
1.



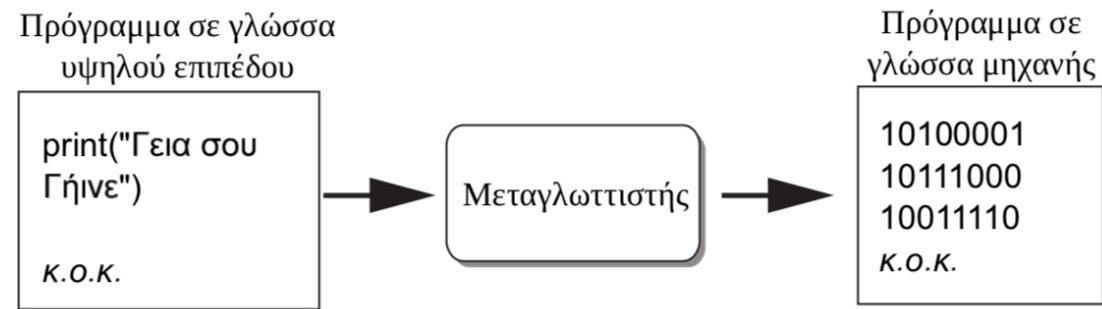
2.



3.

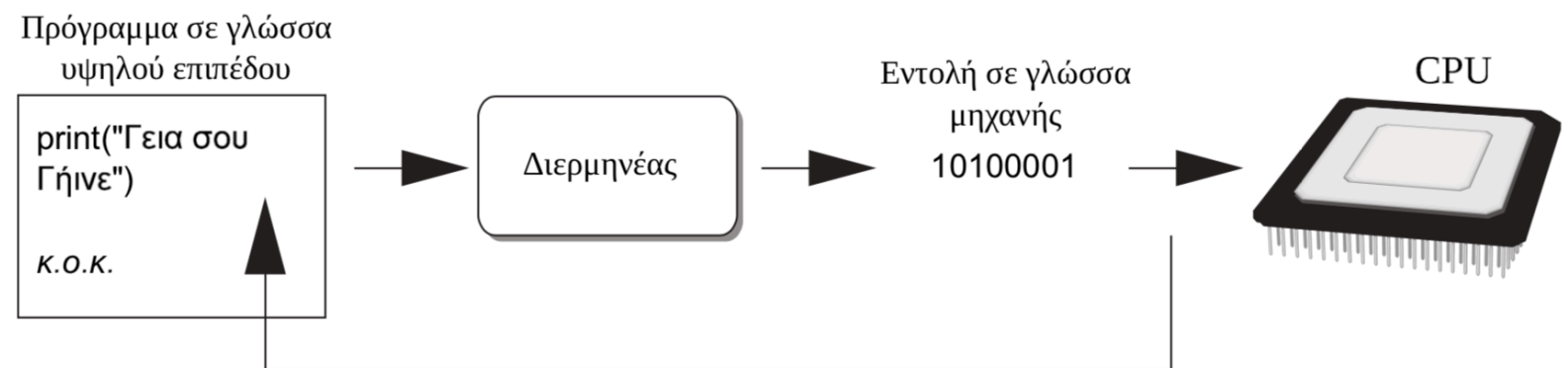


Γλώσσες Υψηλού Επιπέδου



Compiler: Μετατρέπει συνολικά το πρόγραμμα σε γλώσσα μηχανής

Interpreter: Μετατρέπει κάθε εντολή στις αντίστοιχες εντολές σε γλώσσα μηχανής και (αμέσως) τις εκτελεί



(Σφάλματα Προγραμματισμού)

- **Συντακτικά σφάλματα:** Τα βρίσκει ο compiler/interpreter

```
print(Hello world!)
```

- **Εξαιρέσεις:** Σφάλματα κατά την εκτέλεση του προγράμματος που προκαλούν τον απότομο τερματισμό του. Τα προλαμβάνουμε με τις εντολές `try/except`.

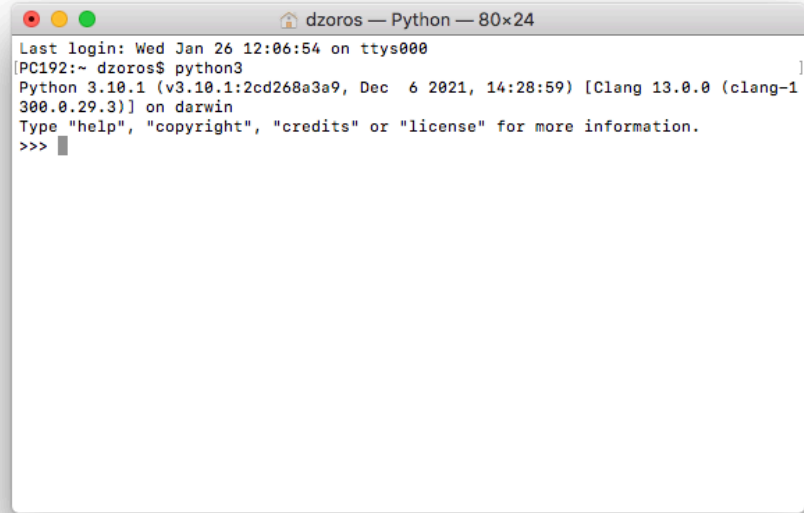
```
x = float(input('Δώσε μου έναν αριθμό: '))
y = float(input('Δώσε μου ακόμα έναν: '))
try:
    print(x/y)
except ZeroDivisionError:
    print('Ο δεύτερος αριθμός είναι 0!')
```

- **Λογικά σφάλματα:** Τα βρίσκει ο προγραμματιστής (debugging)



- Υποστηρίζει τόσο **Διαδικαστικό** όσο και **Αντικειμενοστραφή Προγραμματισμό**.
- Χρησιμοποιεί **διερμηνέα** (scripting language).
- Εκτελείται:
 - Είτε ανά εντολή στο Python shell / κέλυφος (**Διαδραστική λειτουργία**).
 - Είτε ως πλήρες αρχείο προγράμματος με κατάληξη .py (**Λειτουργία script**).
- **Δυναμική γλώσσα** στον ορισμό τύπου μεταβλητών.
- **Εύκολη** στη σύνταξη.
- **Υποχρεωτική στοίχιση** (indentation), όχι μόνο για μορφοποίηση!
- Δημιουργία **μικρότερων σε μέγεθος προγραμμάτων** σε σχέση με άλλες γλώσσες.
- **Μεγάλη πρότυπη βιβλιοθήκη** έτοιμων συναρτήσεων.

Γραμμή Εντολών



A terminal window titled "dzoros — Python — 80x24". The text inside shows the last login time, the command to run Python 3, and the Python 3.10.1 startup banner. The prompt is ">>>".

```
Last login: Wed Jan 26 12:06:54 on ttys000
[PC192:~ dzoros$ python3
Python 3.10.1 (v3.10.1:2cd268a3a9, Dec  6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> █
```

IDE

(Integrated Development Environment)



An IDLE Shell window titled "IDLE Shell 3.10.1". The text inside is identical to the terminal window above. The prompt is ">>>". The status bar at the bottom right shows "Ln: 3 Col: 0".

```
IDLE Shell 3.10.1
Python 3.10.1 (v3.10.1:2cd268a3a9, Dec  6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
Ln: 3 Col: 0
```

Περιεχόμενα Μαθήματος

- Είσοδος, επεξεργασία και έξοδος
- Δομές απόφασης και Boolean Λογική
- Δομές επανάληψης
- Συναρτήσεις
- Λίστες και πλειάδες
- Λεξικά και σύνολα
- Αντικειμενοστραφής προγραμματισμός
- Κληρονομικότητα
- Αναδρομή
- Εξαιρέσεις
- Συμβολοσειρές
- Αρχεία

Βασικοί Τύποι Δεδομένων

Είσοδος / Έξοδος

- Έξοδος
- Συμβολοσειρές
- Μεταβλητές
- Αριθμητικοί Τύποι Δεδομένων
- Είσοδος

Έξοδος (print)

Σύνταξη:

```
print ( ' Συμβολοσειρά  εξόδου ' )
```

Παράδειγμα: (διαδραστική λειτουργία)

```
>>> print('Hello world')  
Hello world  
>>>
```

Παράδειγμα: (output.py)

```
print('Δημήτρης Ζώρος')  
print('Γραφείο: 306')  
print('Email: dzoros@math.uoa.gr')
```

Έξοδος:

```
Δημήτρης Ζώρος  
Γραφείο: 306  
Email: dzoros@math.uoa.gr
```

Ἐξοδος (print)

Εναλλακτική Σύνταξη:

```
print("Συμβολοσειρά  εξόδου")
print(' ' 'Συμβολοσειρά  εξόδου' ' ' ' )
print("""Συμβολοσειρά  εξόδου""")
```

Παράδειγμα: (διαδραστική λειτουργία)

```
>>> print("Δώσ' μου έναν αριθμό: ")
Δώσ' μου έναν αριθμό:
>>>
```

Παράδειγμα: (διαδραστική λειτουργία)

```
>>> print('Είμαι ο "δολοφόνος"!')
Είμαι ο "δολοφόνος"!
>>>
```

Παράδειγμα: (διαδραστική λειτουργία)

```
>>> print('Εσύ' σαι ο "δολοφόνος";')
Εσύ' σαι ο "δολοφόνος";
>>>
```

Έξοδος (print)

Εναλλακτική Σύνταξη:

```
print ("Συμβολοσειρά  εξόδου")
print (' ' 'Συμβολοσειρά  εξόδου' ' ' ' )
print ("\" \" \"Συμβολοσειρά  εξόδου\" \" \"")
```

Παράδειγμα: (διαδραστική λειτουργία)

```
>>> print(''Δημήτρης Ζώρος  
Γραφείο: 306  
Email: dzoros@math.uoa.gr'')  
Δημήτρης Ζώρος  
Γραφείο: 306  
Email: dzoros@math.uoa.gr  
>>>
```

Εμφάνιση πολλών συμβολοσειρών με , :

Παράδειγμα: (διαδραστική λειτουργία)

```
>>> print('Hello', 'world', '!')
Hello world !
>>>
```

Σχόλια

Σύνταξη:

Ότι ακολουθεί αυτό το σύμβολο αγνοείται
από τον διερμηνέα

Παράδειγμα: (output.py)

```
# Ας σας συστηθώ:  
print('Δημήτρης Ζώρος') # Εμφάνιση ονόματος  
print('Γραφείο: 306') # Εμφάνιση γραφείου  
print('Email: dzoros@math.uoa.gr') # Εμφάνιση email
```

Έξοδος:

```
Δημήτρης Ζώρος  
Γραφείο: 306  
Email: dzoros@math.uoa.gr
```


Συμβολοσειρές (`str`)

Τύποι δεδομένων:

`str`
`int`
`float`
...



Συμβολοσειρά

- Αλληλουχία χαρακτήρων
- Οριοθετείται από εισαγωγικά ('', '', '')

Η συνάρτηση `type` επιστρέφει τον τύπο του ορίσματος.

Παράδειγμα: (διαδραστική λειτουργία)

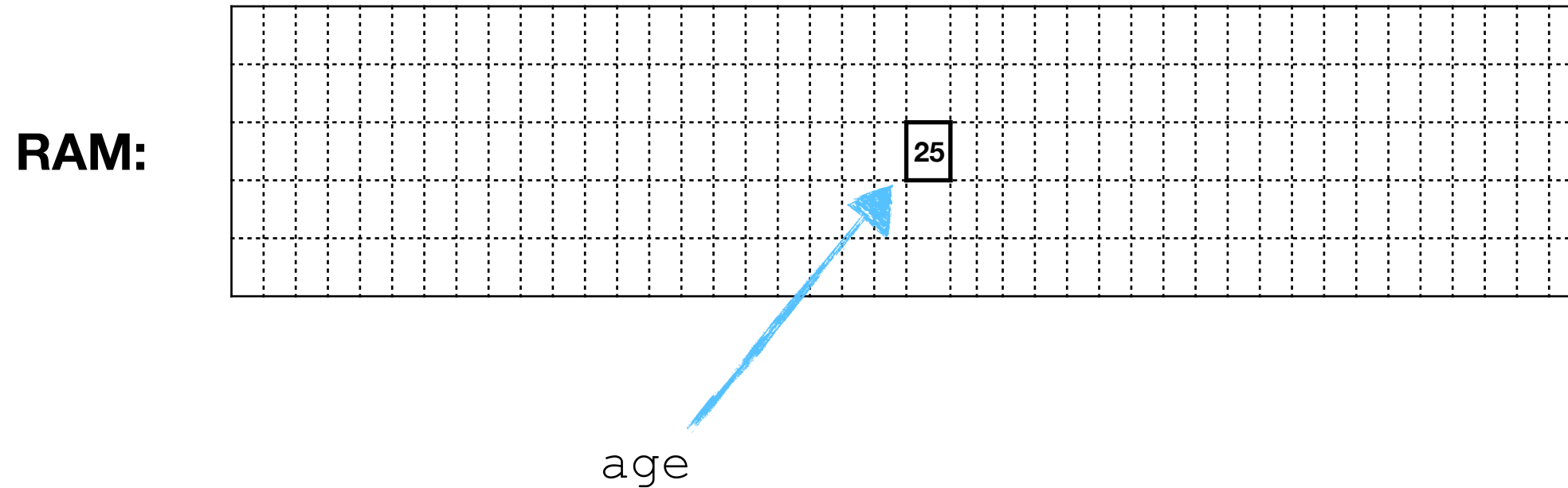
```
>>> type('Hello world')  
<class 'str'>  
>>>
```



(Στην πραγματικότητα επιστρέφει την κλάση που ανήκει το αντικείμενο που δέχεται σαν όρισμα...)

Μεταβλητές

Ονόματα που αντιπροσωπεύουν τιμές αποθηκευμένες στη μνήμη.



Μια μεταβλητή έχει:

- Τύπο `int` (ακέραιος)
- Όνομα (αναγνωριστικό) `age`
- Τιμή `25`

Μεταβλητές

Δημιουργία μεταβλητής: (εντολή εκχώρησης =)

όνομα_μεταβλητής = τιμή
όνομα_μεταβλητής = έκφραση

Παράδειγμα: (διαδραστική λειτουργία)

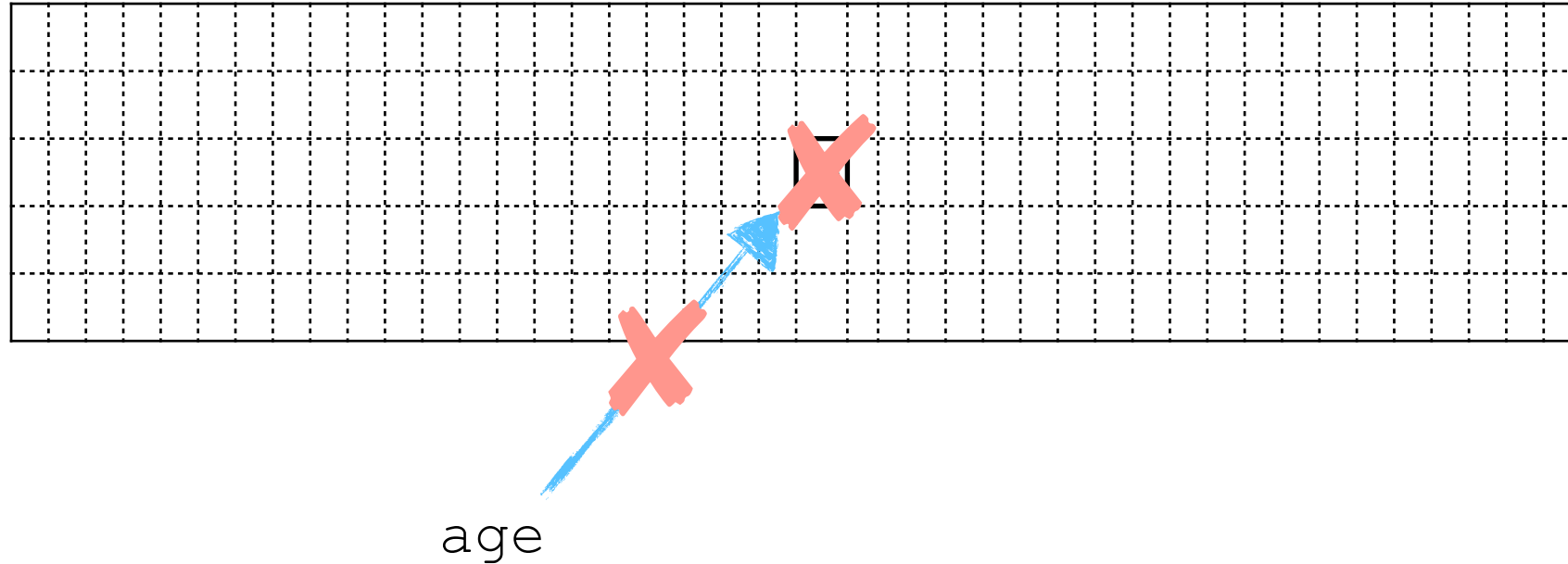
```
>>> age = 25  
>>> print(age)  
25
```

Στην Python **δεν ορίζουμε** τον τύπο της μεταβλητής! Ο διερμηνέας τον συμπεραίνει από την τιμή που της δίνουμε.

```
>>> type(age)  
<class 'int'>  
>>> age_1 = 'ηλικία'  
>>> type(age_1)  
<class 'str'>
```

Μεταβλητές

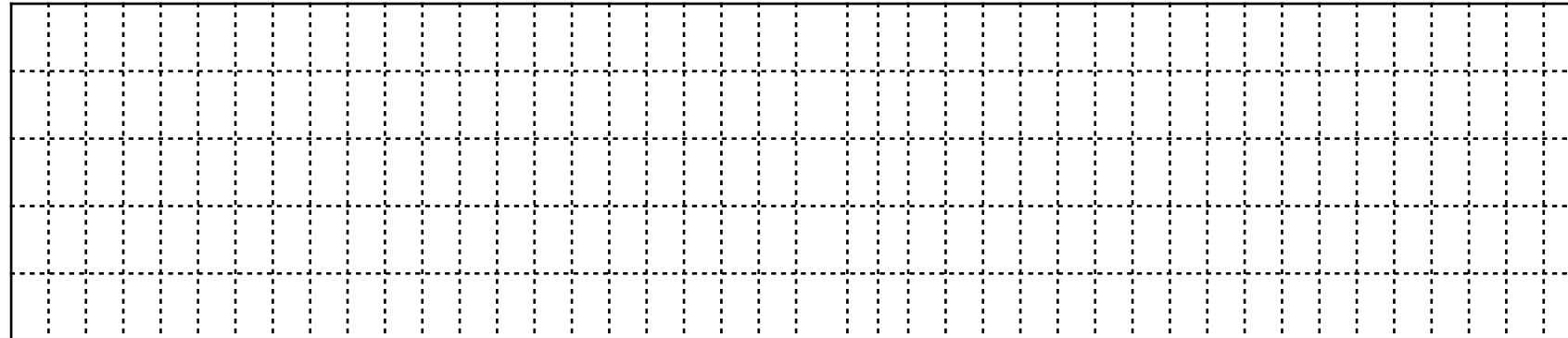
RAM:



Όταν μία τιμή δεν αναφέρεται από κάποια μεταβλητή ο διερμηνέας την αφαιρεί από τη μνήμη (garbage collection)

Μεταβλητές

RAM:



age

Όταν μία τιμή δεν αναφέρεται από κάποια μεταβλητή ο διερμηνέας την αφαιρεί από τη μνήμη (garbage collection)

Κανόνες Ονομασίας

- Δε μπορούν να χρησιμοποιηθούν δεσμευμένες λέξεις (λέξεις-κλειδιά):
`class, for, if, else, while...`
- Ξεκινάνε με γράμμα ή κάτω παύλα (`_`)
- Περιέχουν γράμματα, αριθμούς, `_` (όχι τελείες, `*`, `#` κτλ.)
- Όχι κενά
- Διάκριση μεταξύ κεφαλαίων και πεζών (case sensitive)

Σύμβαση:

- Περιγραφικά ονόματα
- Χωρισμός λέξεων με `_`
- Σταθερές με κεφαλαία

Παράδειγμα

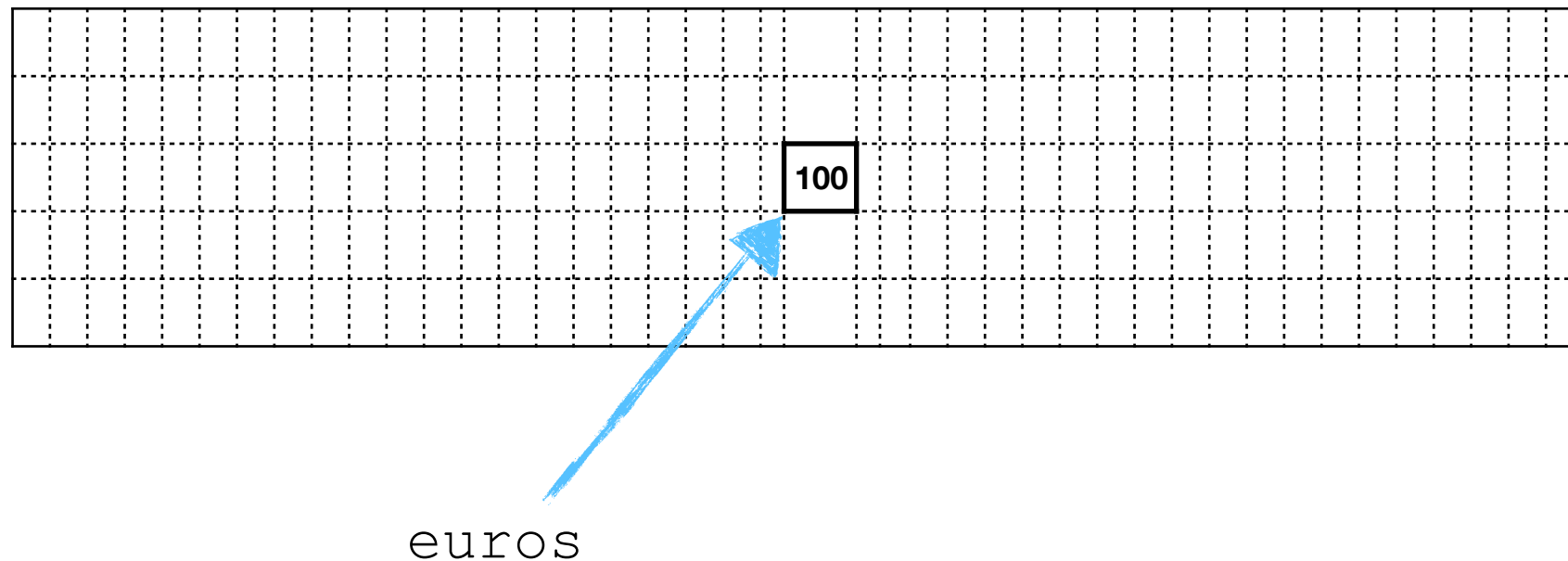
Παράδειγμα: (script.py)

```
#Το πρόγραμμα αυτό παρουσιάζει μία μεταβλητή  
#Εκχώρηση τιμής στη μεταβλητή euros  
euros = 100  
print('Έχω', euros, 'ευρώ στον λογαριασμό μου.')  
  
#Επανεκχώρηση στη μεταβλητή euros έτσι ώστε να  
#αναφέρεται σε άλλη τιμή.  
euros = 99  
print('Τώρα όμως έχω', euros, 'ευρώ στον λογαριασμό μου!')
```

Έξοδος:

Έχω 100 ευρώ στον λογαριασμό μου.
Τώρα όμως έχω 99 ευρώ στον λογαριασμό μου!

RAM:



Παράδειγμα

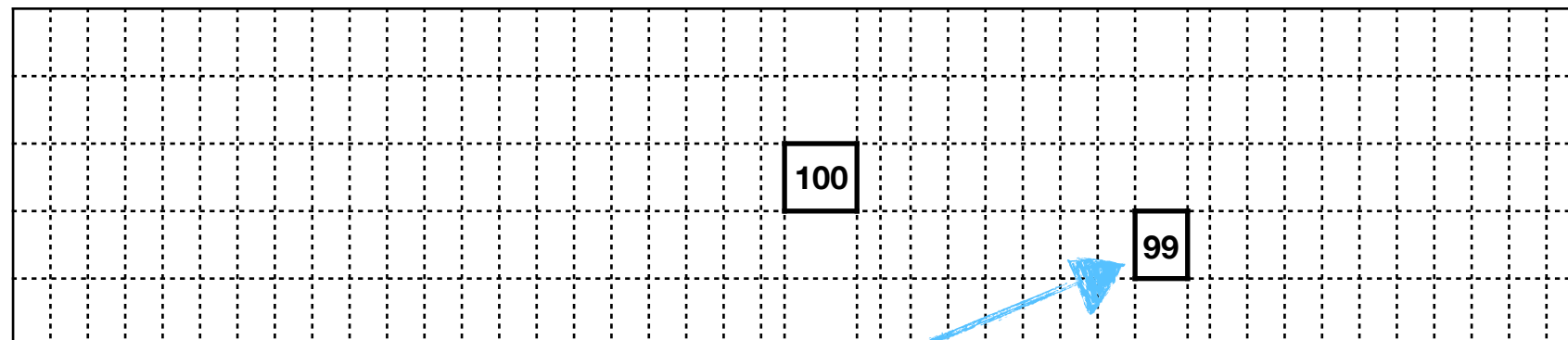
Παράδειγμα: (script.py)

```
#Το πρόγραμμα αυτό παρουσιάζει μία μεταβλητή  
#Εκχώρηση τιμής στη μεταβλητή euros  
euros = 100  
print('Έχω', euros, 'ευρώ στον λογαριασμό μου.')  
  
#Επανεκχώρηση στη μεταβλητή euros έτσι ώστε να  
#αναφέρεται σε άλλη τιμή.  
euros = 99  
print('Τώρα όμως έχω', euros, 'ευρώ στον λογαριασμό μου!')
```

Έξοδος:

Έχω 100 ευρώ στον λογαριασμό μου.
Τώρα όμως έχω 99 ευρώ στον λογαριασμό μου!

RAM:



euros

Παράδειγμα

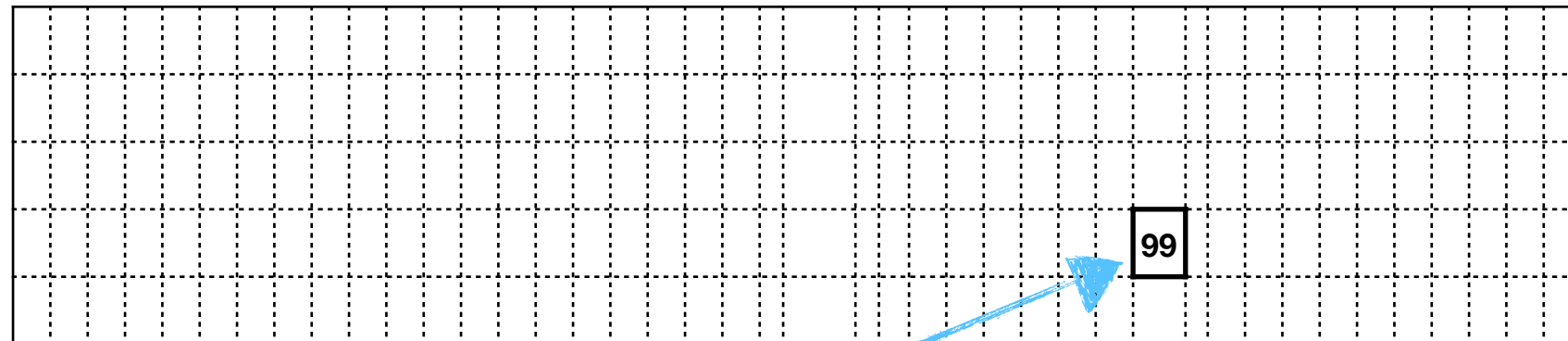
Παράδειγμα: (script.py)

```
#Το πρόγραμμα αυτό παρουσιάζει μία μεταβλητή  
#Εκχώρηση τιμής στη μεταβλητή euros  
euros = 100  
print('Έχω', euros, 'ευρώ στον λογαριασμό μου.')  
  
#Επανεκχώρηση στη μεταβλητή euros έτσι ώστε να  
#αναφέρεται σε άλλη τιμή.  
euros = 99  
print('Τώρα όμως έχω', euros, 'ευρώ στον λογαριασμό μου!')
```

Έξοδος:

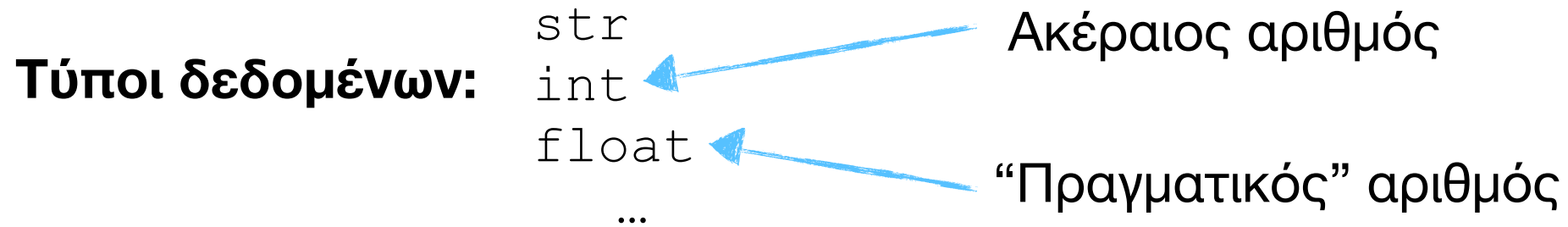
Έχω 100 ευρώ στον λογαριασμό μου.
Τώρα όμως έχω 99 ευρώ στον λογαριασμό μου!

RAM:



euros

Αριθμητικοί Τύποι Δεδομένων (`int`, `float`)



Κανόνες προσδιορισμού τύπου:

- Αριθμητική σταθερά **χωρίς** υποδιαστολή (.) θεωρείται `int`

π.χ. 5, -5 ...

- Αριθμητική σταθερά **με** υποδιαστολή (.) θεωρείται `float`

π.χ. 3.14, 5.0 ...

```
>>> x = 75
>>> type(x)
<class 'int'>
>>> y = 75.2
>>> type(y)
<class 'float'>
```

Επανεκχώρηση Τιμών Διαφορετικού Τύπου

Στην Python (σε αντίθεση με τις περισσότερες γλώσσες) μπορούμε να επανεκχωρήσουμε σε μία μεταβλητή τιμή **διαφορετικού τύπου**!

```
>>> x = 75
>>> type(x)
<class 'int'>
>>> x = 75.2
>>> type(x)
<class 'float'>
>>> x = 'string'
>>> type(x)
<class 'str'>
```

Γιατί έχουμε τύπους δεδομένων;

- Καθορίζουν τον τρόπο που αποθηκεύονται οι τιμές στη μνήμη
- Καθορίζουν τον τρόπο που χειριζόμαστε τις μεταβλητές
(κάποιοι τελεστές εφαρμόζονται μόνο πάνω σε συγκεκριμένους τύπους)

Είσοδος (input)

Σύνταξη:

```
όνομα_μεταβλητής = input('μήνυμα')
```

Παράδειγμα: (με bold η είσοδος)

```
>>> name = input('Πως σε λένε; ')
Πως σε λένε; Δημήτρη
>>> name
'Δημήτρη'
```

Προσοχή: Η συνάρτηση `input` επιστρέφει **πάντα** συμβολοσειρά

```
>>> age = input('Ποια είναι η ηλικία σου; ')
Ποια είναι η ηλικία σου; 25
>>> type(age)
<class 'str'>
```

Είσοδος Αριθμών με την `input`

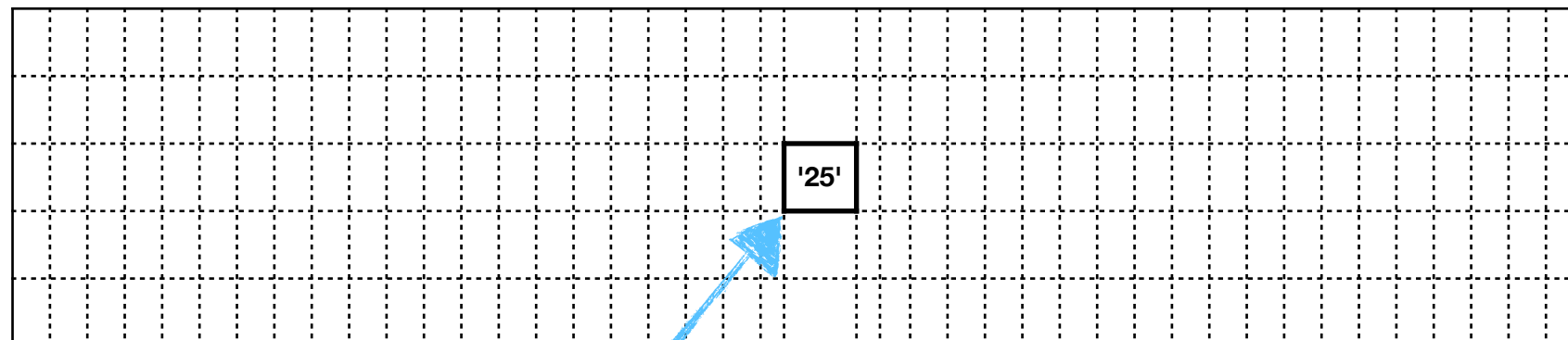
Μετατροπή τύπων:

- Σε ακέραιο: `int()`
- Σε πραγματικό: `float()`
- Σε συμβολοσειρά: `str()`

```
>>> age = input('Ποια είναι η ηλικία σου; ')
Ποια είναι η ηλικία σου; 25
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> type(age)
<class 'int'>
```

Πρώτα υπολογίζεται η έκφραση
και έπειτα εκχωρείται η τιμή
της στην μεταβλητή

RAM:



age

Είσοδος Αριθμών με την `input`

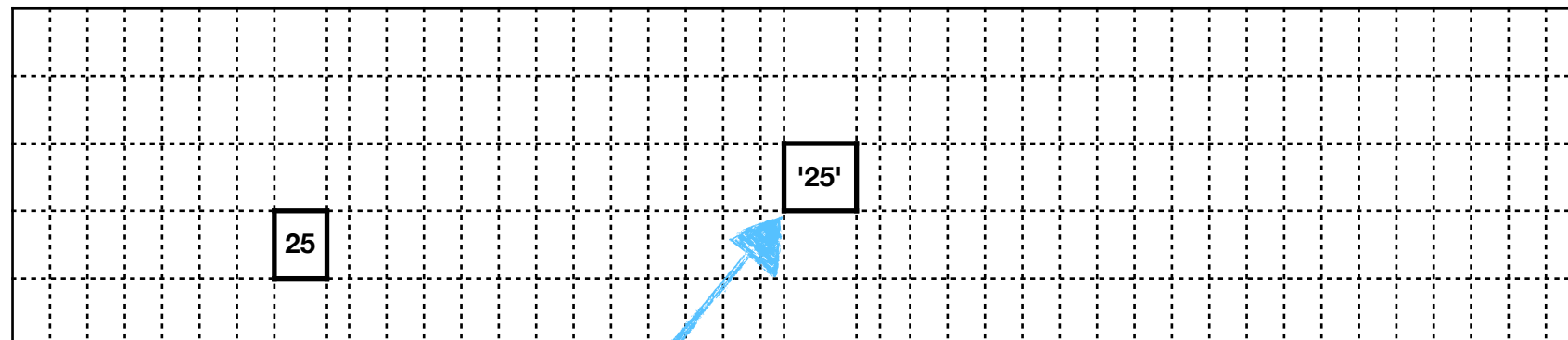
Μετατροπή τύπων:

- Σε ακέραιο: `int()`
- Σε πραγματικό: `float()`
- Σε συμβολοσειρά: `str()`

```
>>> age = input('Ποια είναι η ηλικία σου; ')
Ποια είναι η ηλικία σου; 25
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> type(age)
<class 'int'>
```

Πρώτα υπολογίζεται η έκφραση
και έπειτα εκχωρείται η τιμή
της στην μεταβλητή

RAM:



age

Είσοδος Αριθμών με την `input`

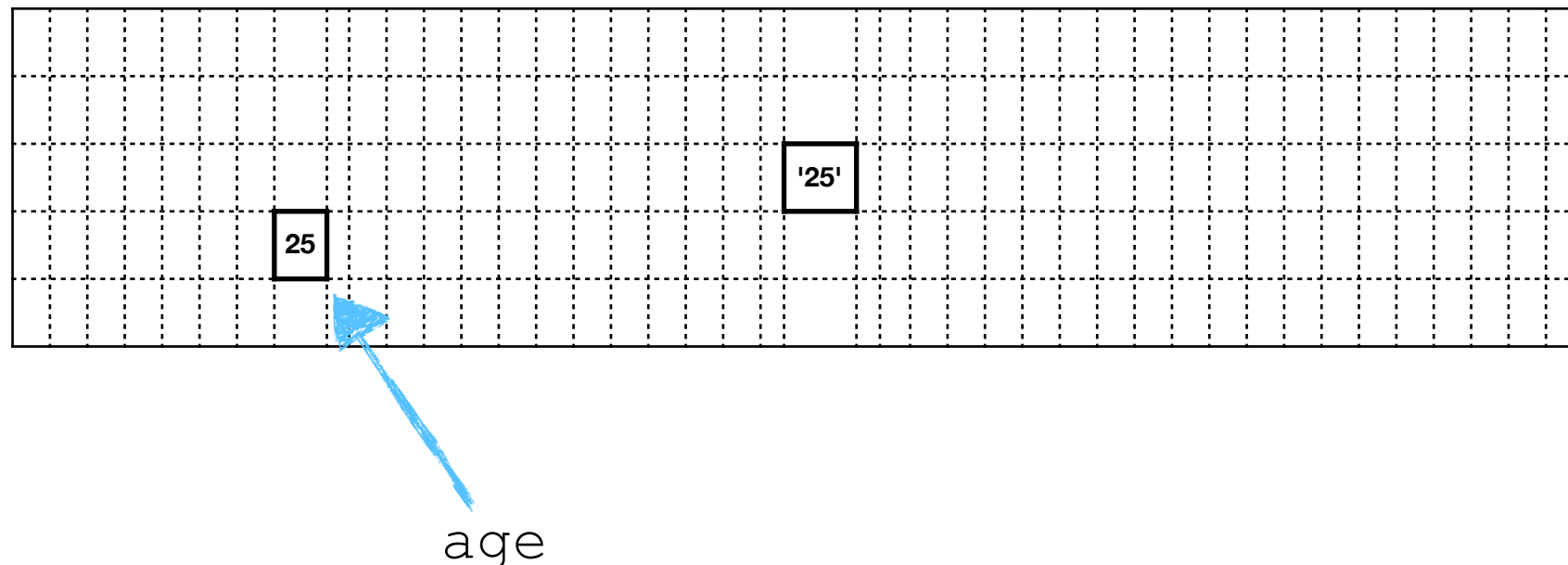
Μετατροπή τύπων:

- Σε ακέραιο: `int()`
- Σε πραγματικό: `float()`
- Σε συμβολοσειρά: `str()`

```
>>> age = input('Ποια είναι η ηλικία σου; ')
Ποια είναι η ηλικία σου; 25
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> type(age)
<class 'int'>
```

Πρώτα υπολογίζεται η έκφραση
και έπειτα εκχωρείται η τιμή
της στην μεταβλητή

RAM:



Είσοδος Αριθμών με την `input`

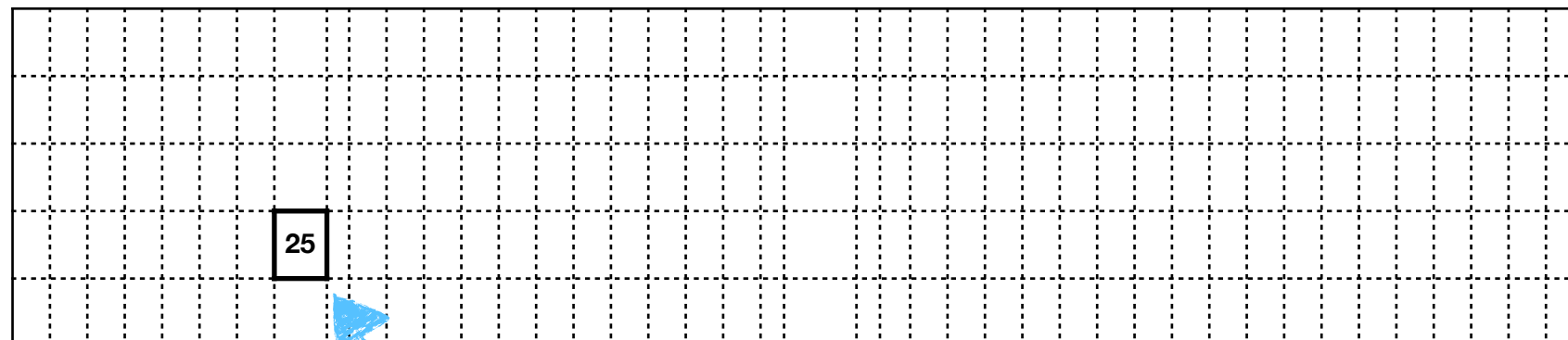
Μετατροπή τύπων:

- Σε ακέραιο: `int()`
- Σε πραγματικό: `float()`
- Σε συμβολοσειρά: `str()`

```
>>> age = input('Ποια είναι η ηλικία σου; ')
Ποια είναι η ηλικία σου; 25
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> type(age)
<class 'int'>
```

Πρώτα υπολογίζεται η έκφραση
και έπειτα εκχωρείται η τιμή
της στην μεταβλητή

RAM:



age

Είσοδος Αριθμών με την `input`

Μετατροπή τύπων:

- Σε ακέραιο: `int()`
- Σε πραγματικό: `float()`
- Σε συμβολοσειρά: `str()`

```
>>> age = input('Ποια είναι η ηλικία σου; ')
Ποια είναι η ηλικία σου; 25
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> type(age)
<class 'int'>
```

```
>>> age = int(input('Ποια είναι η ηλικία σου; '))
Ποια είναι η ηλικία σου; Εικοσιπέντε
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    age = int(input('Ποια είναι η ηλικία σου; '))
ValueError: invalid literal for int() with base 10: 'Εικοσιπέντε'
```



Εξαίρεση!

Είσοδος Αριθμών με την `input`

Μετατροπή τύπων:

- Σε ακέραιο: `int()`
- Σε πραγματικό: `float()`
- Σε συμβολοσειρά: `str()`

```
>>> age = input('Ποια είναι η ηλικία σου; ')
Ποια είναι η ηλικία σου; 25
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> type(age)
<class 'int'>
```

```
>>> age = int(input('Ποια είναι η ηλικία σου; '))
Ποια είναι η ηλικία σου; 25.5
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    age = int(input('Ποια είναι η ηλικία σου; '))
ValueError: invalid literal for int() with base 10: '25.5'
```

Δεν είναι συμβολοσειρά
αριθμητικής τιμής του τύπου
που κάνουμε την μετατροπή!

Είσοδος Αριθμών με την `input`

Μετατροπή τύπων:

- Σε ακέραιο: `int()`
- Σε πραγματικό: `float()`
- Σε συμβολοσειρά: `str()`

```
>>> age = input('Ποια είναι η ηλικία σου; ')
Ποια είναι η ηλικία σου; 25
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> type(age)
<class 'int'>
```

```
>>> age = float(input('Ποια είναι η ηλικία σου; '))
Ποια είναι η ηλικία σου; 25.5
>>>
```

Είσοδος Αριθμών με την `input`

Μετατροπή τύπων:

- Σε ακέραιο: `int()`
- Σε πραγματικό: `float()`
- Σε συμβολοσειρά: `str()`

```
>>> age = input('Ποια είναι η ηλικία σου; ')
Ποια είναι η ηλικία σου; 25
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> type(age)
<class 'int'>
```

```
>>> age = float(input('Ποια είναι η ηλικία σου; '))
Ποια είναι η ηλικία σου; 25
>>>
```

Είσοδος Αριθμών με την `input`

Μετατροπή τύπων:

- Σε ακέραιο: `int()`
- Σε πραγματικό: `float()`
- Σε συμβολοσειρά: `str()`

```
>>> age = input('Ποια είναι η ηλικία σου; ')
Ποια είναι η ηλικία σου; 25
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> type(age)
<class 'int'>
```

```
>>> age = float(input('Ποια είναι η ηλικία σου; '))
Ποια είναι η ηλικία σου; 25
>>> age
25.0
```

Παραδείγματα Μετατροπών

float **σε** int: (αποκοπή δεκαδικών ψηφίων)

```
>>> x = 5.5
>>> int(x)
5
>>> x = -5.5
>>> int(x)
-5
```

int **σε** float:

```
>>> x = 5
>>> float(x)
5.0
```

float **σε** str:

```
>>> x = 5.5
>>> str(x)
'5.5'
```

Παραδείγματα Μετατροπών

float σε int: (αποκοπή δεκαδικών ψηφίων)

```
>>> x = 5.5
>>> int(x)
5
>>> x = -5.5
>>> int(x)
-5
```

**Αναπαράσταση στο
δυναμικό σύστημα:**

int σε float:

```
>>> x = 5
>>> float(x)
5.0
```

5	101
5.0	01000000101000000000000000000000
'5.0'	00110101 00101110 00110000

float σε str:

```
>>> x = 5.5
>>> str(x)
'5.5'
```

Διακοπή Ροής Προγράμματος

(Χρησιμοποιώντας την `input`)

Παράδειγμα: (polis.py)

```
print('                Η πόλις του Κ. Καβάφη')
print()
print('''Είπες· «Θα πάγω σ' άλλη γη, θα πάγω σ' άλλη θάλασσα.
Μια πόλις άλλη θα βρεθεί καλύτερη από αυτή.
Κάθε προσπάθεια μου μια καταδίκη είναι γραφτή·
κι είν' η καρδιά μου — σαν νεκρός — θαμμένη.
Ο νους μου ως τότε μες στον μαρασμόν αυτόν θα μένει.
Όπου το μάτι μου γυρίσω, όπου κι αν δω
ερείπια μαύρα της ζωής μου βλέπω εδώ,
που τόσα χρόνια πέρασα και ρήμαξα και χάλασα.»''')

input('''-----
                Να συνεχίσω;
-----''')

print('''Καινούριους τόπους δεν θα βρεις, δεν θά βρεις άλλες θάλασσες.
Η πόλις θα σε ακολουθεί. Στους δρόμους θα γυρνάς
τους ίδιους. Και στες γειτονιές τες ίδιες θα γερνάς·
και μες στα ίδια σπίτια αυτά θ' ασπρίζεις.
Πάντα στην πόλι αυτή θα φθάνεις. Για τα αλλού — μη ελπίζεις —
δεν έχει πλοίο για σε, δεν έχει οδό.
Έτσι που τη ζωή σου ρήμαξες εδώ
στην κόχνη τούτη την μικρή, σ' όλην την γη την χάλασες.'''')
```


Μαθηματικοί Υπολογισμοί

Έξοδος (συνέχεια)

- Τελεστές Πράξεων
- Εκφράσεις Μεικτών Τύπων
- Μορφοποίηση Εξόδου
- Μορφοποίηση Αριθμών

Τελεστές Πράξεων

Τελεστής	Πράξη	Λειτουργία
+	Πρόσθεση	Προσθέτει δύο αριθμούς
-	Αφαίρεση	Αφαιρεί έναν αριθμό από έναν άλλο
*	Πολλαπλασιασμός	Πολλαπλασιάζει έναν αριθμό με έναν άλλο
/	Διαίρεση	Διαιρεί έναν αριθμό με έναν άλλο και δίνει το αποτέλεσμα ως αριθμό κινητής υποδιαστολής
//	Ακέραια διαίρεση	Διαιρεί έναν αριθμό με έναν άλλο και δίνει το αποτέλεσμα ως ακέραιο
%	Υπόλοιπο διαίρεσης	Διαιρεί έναν αριθμό με έναν άλλο και δίνει το υπόλοιπο
**	Ύψωση σε δύναμη	Υψώνει έναν αριθμό σε κάποια δύναμη

Παράδειγμα: (διαίρεση)

```
>>> # Διαίρεση κινητής υποδιαστολής
>>> 5/2
2.5
>>> 4/2
2.0
>>> # Ακέραια διαίρεση
>>> 5//2
2
>>> # Υπόλοιπο
>>> r = 7%2
>>> q = 7//2
>>> print('7 =', q, '* 2', '+', r)
7 = 3 * 2 + 1
```

Τελεστές Πράξεων

Παράδειγμα: (script.py)

```
# Το πρόγραμμα αυτό διαβάζει την αρχική τιμή ενός προϊόντος  
# και υπολογίζει τη μειωμένη τιμή του, με έκπτωση 20%.  
  
# Είσοδος της αρχικής τιμής του προϊόντος.  
original_price = float(input('Δώσε την αρχική τιμή: '))  
  
# Υπολογισμός του ποσού της έκπτωσης.  
discount = original_price * 0.2  
  
# Υπολογισμός της μειωμένης τιμής.  
sale_price = original_price - discount  
  
# Εμφάνιση της τελικής τιμής πώλησης.  
print('Η τελική τιμή πώλησης είναι', sale_price)
```

Έξοδος:

```
Δώσε την αρχική τιμή: 253  
Η τελική τιμή πώλησης είναι 202.4
```

Προτεραιότητα Τελεστών

Πρώτα: $**$

Μετά: $*, /, //, \%$

Τέλος: $+, -$

Αλλάζει με τη χρήση παρενθέσεων

Παράδειγμα:

```
>>> 1+2*3**4
163
>>> ((1+2)*3)**4
6561
```

Πράξεις της ίδιας προτεραιότητας εκτελούνται από τα αριστερά προς τα δεξιά **εκτός από την ύψωση σε δύναμη!**

Παράδειγμα: (ύψωση σε δύναμη)

```
>>> 4**3**2
262144
>>> (4**3)**2
4096
```

Προτεραιότητα Τελεστών

Παράδειγμα: (script.py)

```
# Είσοδος του πλήθους δευτερολέπτων από τον χρήστη.  
total_seconds = int(input('Δώσε πλήθος δευτερολέπτων: '))  
  
# Υπολογισμός του πλήθους των ωρών.  
hours = total_seconds // 3600  
  
# Υπολογισμός του πλήθους των λεπτών που απομένουν.  
minutes = (total_seconds // 60) % 60  
  
# Υπολογισμός του πλήθους των δευτερολέπτων που απομένουν.  
seconds = total_seconds % 60  
  
# Εμφάνιση των αποτελεσμάτων.  
print('Ο χρόνος σε ώρες, λεπτά και δευτερόλεπτα:')  
print('Ώρες:', hours)  
print('Λεπτά:', minutes)  
print('Δευτερόλεπτα:', seconds)
```

Έξοδος:

```
Δώσε πλήθος δευτερολέπτων: 343845  
Ο χρόνος σε ώρες, λεπτά και δευτερόλεπτα:  
Ώρες: 95  
Λεπτά: 30  
Δευτερόλεπτα: 45
```

Εκφράσεις Μεικτών Τύπων

- Πράξη μεταξύ δύο τιμών τύπου `int`, το αποτέλεσμα είναι τύπου `int`.
- Πράξη μεταξύ δύο τιμών τύπου `float`, το αποτέλεσμα είναι τύπου `float`.
- Πράξη μεταξύ ενός `int` και ενός `float`, **το αποτέλεσμα είναι τύπου `float`**.
(Ο `int` μετατρέπεται προσωρινά σε `float`.)

Παράδειγμα:

```
>>> 5+2
7
>>> 5.0+2.0
7.0
>>> 5+2.0
7.0
```

Τελεστές επαυξημένης εκχώρησης:

Γράφουμε	Αντί για
<code>a += x;</code>	<code>a = a + x</code>
<code>a -= x</code>	<code>a = a - x</code>
<code>a *= x</code>	<code>a = a * x</code>
<code>a /= x</code>	<code>a = a / x</code>
<code>a %= x</code>	<code>a = a % x</code>
<code>a **= x</code>	<code>a = a ** x</code>
<code>a //= x</code>	<code>a = a // x</code>

(Πράξεις με Αριθμούς Κινητής Υποδιαστολής)

```
>>> 1.1 + 2.2  
3.300000000000000003
```

WTF?!

```
>>> 0.1 + 0.2  
0.300000000000000004
```

Στις περισσότερες γλώσσες προγραμματισμού οι αριθμοί κινητής υποδιαστολής αποθηκεύονται σύμφωνα με το πρότυπο IEEE 754, όπου οι περισσότεροι πραγματικού αριθμοί δεν εκφράζονται ακριβώς...

Για παράδειγμα ο αριθμός 0.1 αποθηκεύεται στον Η/Υ ως:

00111101110011001100110011001101

Που αντιστοιχεί στον αριθμό:

0.1000000001490116119384765625

Έτοιμες Συναρτήσεις

Συνάρτηση	Λειτουργία
<code>abs()</code>	Απόλυτη τιμή ορίσματος
<code>min()</code> / <code>max()</code>	Ελάχιστο/Μέγιστο μεταξύ των ορισμάτων
<code>pow()</code>	Ύψωση του πρώτου ορίσματος στο δεύτερο όρισμα
<code>round()</code>	Στρογγυλοποίηση ορίσματος στον κοντινότερο ακέραιο

Παράδειγμα:

```
>>> abs(-3), min(-2.0, 0, 3, 5.5), max(-1, 3.4, 5, 9)
(3, -2.0, 9)
>>> pow(2, 3), round(3.14)
(8, 3)
```

Πλειάδες

Στο module `math`:

`sqrt`, `floor`, `ceil`, `cos`, `sin`, `tan`, `exp`, `log`, `log2`, `log10`
`pi`, `e`, `inf` (σταθερές)

Παράδειγμα:

```
>>> import math
>>> math.sqrt(9), math.log2(8), math.log10(1000), math.pi
(3.0, 3.0, 3.0, 3.141592653589793)
```


Μορφοποίηση Εξόδου

Ο τελεστής sep:

```
>>> print('Γεια', 'χαρά', '!')
Γεια χαρά !
>>> print('Γεια', 'χαρά', '!', sep = '')
Γειαχαρά!
>>> print('Γεια ', 'χαρά', '!', sep = ' ')
Γεια χαρά!
>>> print('Εγώ', 'εσύ.', sep = ' και ')
Εγώ και εσύ.
```

Ο τελεστής end:

```
print('Ένα', end = '')
print('Δύο', end = '')
print('Τρία')
```

ΈναΔύοΤρία

```
print('Ένα', end = ' ')
print('Δύο', end = ' ')
print('Τρία')
```

Ένα Δύο Τρία

```
>>> print(3,2,1,sep = '--', end = ' πυρ!')
3--2--1 πυρ!
```


Μορφοποίηση Εξόδου

Ο τελεστής¹ `+`: (παράθεση)

```
>>> message = 'abcd' + 'efg'
>>> print(message)
abcdefg
>>> print(message+'hijk'+ 'lmn'+ 'opqr')
abcdefghijklmnopqr
```

Προσοχή:

```
>>> print(5)
5
>>> print('5')
5
```

Αλλά:

```
>>> print('5'+5)
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    print('5'+5)
TypeError: can only concatenate str (not "int") to str
```

Όμως:

```
>>> print(5+5)
10
```

¹ Τελεστής δημιουργίας `str`

Μορφοποίηση Εξόδου

Προσοχή:

```
>>> age = 25
>>> print('Η ηλικία μου είναι: ' + age)
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    print('Η ηλικία μου είναι: ' + age)
TypeError: can only concatenate str (not "int") to str
```

Σωστό:

```
>>> print('Η ηλικία μου είναι: ' + str(age))
Η ηλικία μου είναι: 25
```

Ο τελεστής¹ *:

```
>>> print('a' * 10)
aaaaaaaaaa
>>> print(10 * 'a')
aaaaaaaaaa
>>> print('A' + 'πα' * 2 + '!')
Απαπα!
```

¹ Τελεστής δημιουργίας `str`

Ακολουθίες Διαφυγής

Σύνταξη: \ σύμβολο

Ακολουθία	Λειτουργία
\n	Αλλαγή γραμμής
\t	Tab
\'	Εμφάνιση '
\"	Εμφάνιση "
\\	Εμφάνιση \
\	Συνέχιση εντολής στην επόμενη γραμμή

```
>>> print('1\n2\n3')
1
2
3
>>> print('1\t2\t3\n4\t5\t6')
1         2         3
4         5         6
>>> print('To \'να \
τ\' άλλο.')
To 'να τ' άλλο.
```

Μορφοποίηση Αριθμών (format)

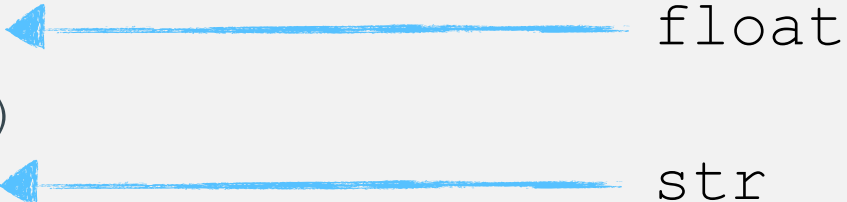
Σύνταξη:

`format(τιμή, 'τελεστής προσαρμογής')`

Πλήθος δεκαδικών ψηφίων: `' .xf '`

ΠΡΟΣΟΧΗ!

```
>>> 10/3
3.3333333333333335
>>> format(10/3, '.3f')
'3.333'
```



```
>>> round(10/3, 3)
3.333
```

```
>>> round(10/3)
3
```

Στην Python μπορούμε να ορίζουμε συναρτήσεις με προεπιλεγμένες τιμές (οπότε όταν καλούμε τη συνάρτησης μπορούμε να παραλείπουμε ορίσματα)

Μορφοποίηση Αριθμών (format)

Σύνταξη:

`format(τιμή, 'τελεστής προσαρμογής')`

Πλήθος δεκαδικών ψηφίων: `' .xf '`

```
>>> 10/3
3.3333333333333335
>>> format(10/3, '.3f')
'3.333'
```

Εύρος πεδίου: `' x '`

```
>>> format(10/3, '25')
'          3.3333333333333335'
>>> format(10/3, '25.2f')
'          3.33'
```

Διαχωρισμός χιλιάδων : `' , '`

```
>>> format(1000, ',')
'1,000'
>>> format(10000/3, '25,.2f')
'          3,333.33'
```

Μορφοποίηση Αριθμών (format)

Εκθετική μορφή: 'e'

```
>>> format(1000000, 'e')  
'1.000000e+06'  
>>> format(1000000, '.1e')  
'1.0e+06'
```

Ποσοστό: '%'

```
>>> format(0.33333333, '%')  
'33.333333%'  
>>> format(0.33333333, '.2%')  
'33.33%'
```

Παράδειγμα:

```
#Το πρόγραμμα αυτό δείχνει πώς μπορεί να μορφοποιηθεί  
#ένας αριθμός κινητής υποδιαστολής ως νόμισμα  
  
monthly_pay = 499.9999  
annual_pay = monthly_pay * 12  
  
print('Ο ετήσιος μισθός σου είναι €', annual_pay, \  
'\nή στρογγυλοποιημένος €', format(annual_pay, ',.2f'), \  
sep=' ')
```

Μορφοποίηση Αριθμών (format)

Εκθετική μορφή: 'e'

```
>>> format(1000000, 'e')  
'1.000000e+06'  
>>> format(1000000, '.1e')  
'1.0e+06'
```

Ποσοστό: '%'

```
>>> format(0.33333333, '%')  
'33.333333%'  
>>> format(0.33333333, '.2%')  
'33.33%'
```

Παράδειγμα:

```
#Το πρόγραμμα αυτό δείχνει πώς μπορεί να μορφοποιηθεί  
#ένας αριθμός κινητής υποδιαστολής ως νόμισμα  
  
monthly_pay = 499.9999  
annual_pay = monthly_pay * 12  
  
print('Ο ετήσιος μισθός σου είναι €', annual_pay, \  
'\nή στρογγυλοποιημένος €', format(annual_pay, ',.2f'), \  
sep='')
```

```
Ο ετήσιος μισθός σου είναι €5999.9988  
ή στρογγυλοποιημένος €6,000.00
```


Boolean Λογική

Έλεγχος Ροής

- Λογικές Εκφράσεις
- Σύγκριση Συμβολοσειρών
- Λογικοί Τελεστές
- Δομές Ελέγχου
- Δομές Επανάληψης
- Το Πρωτόκολλο Επανάληψης

Λογικές Μεταβλητές (bool)

Τύπος δεδομένων: bool ← Τιμή αλήθειας: True, False

Ειδίκευση του int: το True αντιστοιχεί στο 1
και το False στο 0

Παράδειγμα:

```
>>> type(True)
<class 'bool'>
>>> True + 1
2
>>> False * 2
0
```

Σχесιακοί τελεστές: (δημιουργούν λογικές εκφράσεις)

Μαθηματικά	Python
=	==
<	<
>	>
≤	<=
≥	>=
≠	!=

Λογικές Εκφράσεις

Εκφράσεις που η τιμή τους είναι `True` ή `False`:

Παράδειγμα:

```
>>> 4 == 5, 4 <= 5, 4 != 5
(False, True, True)
>>> 8 == (5-1) * 2
True
>>> type(0 == 1)
<class 'bool'>
>>> True > 2
False
>>> True + 1 >= 2
True
```

Τριπλές ανισότητες:

```
>>> x = 0.5
>>> 0 <= x <= 1
True
```

Μετατροπή σε `bool`

Σύνταξη: `bool(Τιμή)`

- Κάθε τιμή διαφορετική του μηδενός μετατρέπεται σε `True`.
- Το `0` και το κενό (`' '`) μετατρέπονται σε `False`

Παράδειγμα:

```
>>> bool(12.3)
True
>>> bool('Python')
True
>>> bool(0)
False
>>> bool(0.0)
False
>>> bool('')
False
>>> bool(' ')
True
```

Σύγκριση Συμβολοσειρών

Στην Python εκτός από αριθμούς μπορούμε να συγκρίνουμε και συμβολοσειρές!

== και !=:

```
>>> 'aabba' == 'aabba'
True
>>> 'aabba' != 'aabaa'
True
>>> 'aabba' != 'Aabaa'
True
```

<, >, <=, και >=:

```
>>> 'a' <= 'b'
True
```

Ο αριθμός ASCII του a είναι 97 ενώ του b 98...

```
>>> ord('A') # Επιστρέφει αριθμός ASCII / UNICODE
65
>>> chr(65) # Επιστρέφει τον χαρακτήρα
'A'
```

```
>>> 'A' <= 'a'
True
```

```
>>> 'aabba' >= 'aabaa'
True
>>> 'aabba' >= 'aab'
True
>>> 'aabaa' <= 'aabb'
True
```

Σύγκριση Συμβολοσειρών

Παράδειγμα:

```
# Έλεγχος αν ένας χρήστης έχει κάνει εγγραφή

username = input('Δώσε το username σου: ')
registered_user = 'admin'

print('Έχει γίνει εγγραφή:', registered_user == username)
```

```
Δώσε το username σου: dzoros
Έχει γίνει εγγραφή: False
```

```
Δώσε το username σου: admin
Έχει γίνει εγγραφή: True
```

Λογικοί Τελεστές

Οι λογικοί τελεστές δρουν πάνω σε λογικές εκφράσεις.

Λογική	Python
Σύζευξη ($\wedge$)	and
Διάζευξη ($\vee$)	or
Άρνηση ($\neg$)	not ()

Παράδειγμα:

```
>>> 4 <= 5 and 5 <= 6
True
>>> 4 <= 5 <= 6
True
>>> False and True
False
>>> False or True
True
>>> not (True + 1 >= 2)
False
```

Short-Circuit Αποτίμηση

Αν η τιμή της έκφρασης μπορεί να υπολογιστεί μόνο από το πρώτο μέλος του `and` ή του `or` το δεύτερο μέλος δεν υπολογίζεται:

```
>>> 1<'2'  
Traceback (most recent call last):  
  File "<pyshell#44>", line 1, in <module>  
    1<'2'  
TypeError: '<' not supported between instances of 'int' and 'str'
```

Όμως:

```
>>> True or 1<'2'  
True  
>>> False and 1<'2'  
False
```


Δομές Ελέγχου (if-elif-else)

Σύνταξη:

```
if συνθήκη:  
    εντολή  
    εντολή  
    ...
```

Μπλοκ κώδικα

Προσοχή:

Στην Python, η οριοθέτηση των μπλοκ κώδικα **δεν** γίνεται με άγκιστρα, αγκύλες, end κτλ. αλλά με την **κατάλληλη στοίχιση** της κάθε γραμμής (indentation).

Παράδειγμα:

```
sales = int(input('Πόσα ευρώ ήταν οι πωλήσεις: '))
```

```
if sales >= 5000:  
    print('Φτάσαμε τον στόχο.')  
    print('Συγχαρητήρια!')
```



Δομές Ελέγχου (if-elif-else)

Σύνταξη:

```
if συνθήκη:  
    εντολή  
    εντολή  
    ...  
else:  
    εντολή  
    εντολή  
    ...
```

Παράδειγμα:

```
sales = int(input('Πόσα ευρώ ήταν οι πωλήσεις: '))  
  
if sales >= 5000:  
    print('Φτάσαμε τον στόχο.')  
    print('Συγχαρητήρια!')  
else:  
    print('Δεν τα πήγαμε και τόσο καλά...')
```



Δομές Ελέγχου (if-elif-else)

Σύνταξη:

```
if συνθήκη:
    εντολή
    εντολή
    ...
elif: ← Όσες φορές θέλουμε!
    εντολή
    εντολή
    ...
else:
    εντολή
    εντολή
    ...
```

Παράδειγμα:

```
sales = int(input('Πόσα ευρώ ήταν οι πωλήσεις: '))

if sales > 5000:
    print('Επεράσαμε τον στόχο!')
elif sales == 5000:
    print('Φτάσαμε τον στόχο!')
elif 4900 <= sales < 5000:
    print('Ημασταν πολύ κοντά.')
else:
    print('Δεν τα πήγαμε και τόσο καλά...')

print('Καλή συνέχεια!')
```

Δομές Ελέγχου (if-elif-else)

Παράδειγμα:

```
score = int(input('Δώσε τη βαθμολογία σου (0-100): '))

if score > 90:
    grade = 'A'
elif score > 80:
    grade = 'B'
elif score > 70:
    grade = 'C'
elif score > 60:
    grade = 'D'
else:
    grade = 'F'

print('Με βαθμολογία', score, 'στα 100, ο βαθμός σου είναι:',
      grade)
```

Δομές Ελέγχου (if-elif-else)

Παράδειγμα:

```
username = input('Username: ')
password = input('Password: ')

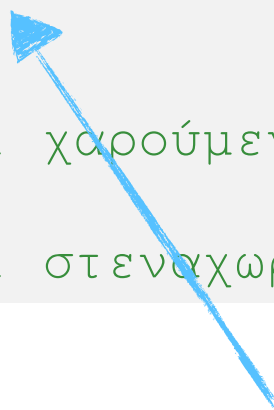
registered_user = 'admin'
registered_password = '1234'

if username == registered_user and password ==\
    registered_password:
    print('Πρόσβαση δεκτή')
elif username != registered_user:
    print('Λάθος όνομα χρήστη')
else:
    print('Λάθος κωδικός')
```

Δομές Ελέγχου (if-elif-else)

Παράδειγμα:

```
print('Είσαι ικανοποιημένος/η από το μάθημα;')
test = bool(input('Πληκτρολόγησε 1 για ναι και 0 για όχι: '))
# test == True
if test:
    print('Είμαι χαρούμενος!')
else:
    print('Είμαι στεναχωρημένος...')
```



Πάντα επιστρέφει str!

Σωστό:

```
print('Είσαι ικανοποιημένος/η από το μάθημα;')
test = int(input('Πληκτρολόγησε 1 για ναι και 0 για όχι: '))
# 1 == True και 0 == False
if test:
    print('Είμαι χαρούμενος!')
else:
    print('Είμαι στεναχωρημένος...')
```

Δομές Επανάληψης (while)

Σύνταξη:

```
while συνθήκη:  
    εντολή  
    εντολή  
    ...
```

Όσο η συνθήκη είναι `True` το μπλοκ εντολών θα εκτελείται.

Παράδειγμα:

```
x = 5  
while x > 0:  
    print(x, end = ' ')  
    x -= 1
```

```
5 4 3 2 1
```

Δομές Επανάληψης (while)

Παράδειγμα:

```
# Υπολογισμός δυνάμεων του 2

N = int(input('Δώσε έναν φυσικό αριθμό: '))
v = 1
i = 0

while i <= N:
    print('2^'+format(i, '2'), '=', format(v, '4'))
    v *= 2
    i += 1
```

```
Δώσε έναν φυσικό αριθμό: 10
2^ 0 =      1
2^ 1 =      2
2^ 2 =      4
2^ 3 =      8
2^ 4 =     16
2^ 5 =     32
2^ 6 =     64
2^ 7 =    128
2^ 8 =    256
2^ 9 =    512
2^10 =   1024
```


Δομές Επανάληψης (while)

Παράδειγμα:

```
# Άθροισμα άγνωστου πλήθους αριθμών

sum = 0
keep_on = 'Y'

while keep_on == 'Y':
    x = int(input('Δώσε αριθμό: '))
    sum += x
    keep_on = input('Θες να συνεχίσεις; (Y/N) ')
print('Το άθροισμα των αριθμών που έδωσες είναι:', sum)
```

```
# Άθροισμα ψηφίων θετικού ακεραίου

n = int(input('Δώσε θετικό ακέραιο: '))
tmp = n
sum = 0

while tmp > 0:
    sum += tmp%10 # Έυρεση τελευταίου ψηφίου
    tmp = int(tmp/10) # Ακέραια διαίρεση οπότε αποκόπτουμε το
                    # τελευταίο ψηφίο
print('Το άθροισμα των ψηφίων του', n, 'είναι', str(sum)+'.'
```

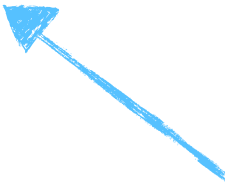
?!

Ατέρμονες Βρόχοι

```
# Το πρόγραμμα αυτό δεν τερματίζει ποτέ!  
  
while True:  
    print('Βρέχει...', end = ' ')
```

Η εντολή `break` διακόπτει την επανάληψη:

```
# Το πρόγραμμα αυτό πάλι δεν τερματίζει ποτέ!  
  
i = 5  
  
while True:  
    if i == 0:  
        break  
    print('Βρέχει...', end = ' ')  
    i -= 1
```



$i = -1$

Ατέρμονες Βρόχοι

```
# Το πρόγραμμα αυτό δεν τερματίζει ποτέ!  
  
while True:  
    print('Βρέχει...', end = ' ')
```

Η εντολή `break` διακόπτει την επανάληψη:

```
# Το πρόγραμμα αυτό τερματίζει.  
  
i = 5  
  
while True:  
    if i == 0:  
        break  
    print('Βρέχει...', end = ' ')  
    i -= 1
```

Βρέχει... Βρέχει... Βρέχει... Βρέχει... Βρέχει...

Δομές Επανάληψης (while)

Παράδειγμα:

```
# Βρίσκει τον πρώτο ακέραιο που διαιρείται με το 11 και  
# με το 13  
  
x = 1 # Το μηδέν διαιρείται με 11 και 13  
  
while True:  
    if x%11 == 0 and x%13 == 0:  
        break  
    x = x+1  
  
print('Ο',x,'διαίρεται με το 11 και το 13.')
```

Ο 143 διαιρείται με το 11 και το 13.

Δομές Επανάληψης (for)

Σύνταξη:

```
for μεταβλητή in [τιμή1, τιμή2, ...]:  
    εντολή  
    εντολή  
    ...
```

- Το μπλοκ εντολών εκτελείται τόσες φορές όσες οι τιμές μέσα στα [].
- Σε κάθε επανάληψη στη μεταβλητή εκχωρείται μία από τις τιμές (από τα αριστερά προς τα δεξιά)

Παράδειγμα:

```
for i in [5,4,3,2,1]:  
    print(i, end = ' ')
```

5 4 3 2 1

Παράδειγμα:

```
for i in ['Χιούη', 'Λιούη', 'Ντιούη']:  
    print(i, end = ' ')
```

Χιούη Λιούη Ντιούη

Αλληλουχίες Αριθμών (range)

Η συνάρτηση `range()` δημιουργεί μία αλληλουχία αριθμών:

```
>>> x = range(10)
>>> for i in x:
...     print(i, end = ' ')
...
...
0 1 2 3 4 5 6 7 8 9
>>> type(x)
<class 'range'>
```

```
>>> y = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> for i in y:
...     print(i, end = ' ')
...
...
0 1 2 3 4 5 6 7 8 9
>>> type(y)
<class 'list'>
>>> x == y
False
```

```
>>> x = y
>>> type(x)
<class 'list'>
```

Αλληλουχίες Αριθμών (range)

- Ένα όρισμα:

`range(x)` → Δημιουργεί την αλληλουχία αριθμών από **0** έως και **x-1**

- Δύο ορίσματα:

`range(x, y)` → Δημιουργεί την αλληλουχία αριθμών από **x** έως **y-1**
(αν **x > y** έχουμε **κενή** αλληλουχία)

- Τρία ορίσματα:

`range(x, y, z)` → Δημιουργεί την αλληλουχία αριθμών από **x** έως **y-1**
με βήμα z

Παραδείγματα:

```
for i in range(1,5):  
    print(i, end = ' ')
```

1 2 3 4

```
for i in range(-10,-1):  
    print(i, end = ' ')
```

-10 -9 -8 -7 -6 -5 -4 -3 -2

```
for i in range(1,10,2):  
    print(i, end = ' ')
```

1 3 5 7 9

Αλληλουχίες Αριθμών (range)

Αρνητικό βήμα:

```
for i in range(10,1,-1):  
    print(i, end = ' ')
```

10 9 8 7 6 5 4 3 2

```
for i in range(1,10,-1):  
    print(i, end = ' ')
```

Αφού έχουμε αρνητικό βήμα θα πρέπει $x > y$!

Παράδειγμα:

```
# Άθροισμα γνωστού πλήθους αριθμών  
  
n = int(input('Πόσους αριθμούς θέλεις να αθροίσω: '))  
sum = 0  
  
for i in range(1,n+1):  
    x = int(input('Δώσε ' + str(i) + 'ο αριθμό: '))  
    sum += x  
print('Το άθροισμα των αριθμών που έδωσες είναι:', sum)
```



Η input δέχεται ένα μόνο όρισμα τύπου str
(δεν μπορούμε να χρησιμοποιήσουμε , , sep ή end)

Ιδιότητες for

- Αλλαγές του μετρητή στο μπλοκ εντολών **δεν** επηρεάζουν το πλήθος των επαναλήψεων:

```
for i in range(1, 5):  
    print(i, end = ' ')  
    i *= 2  
    print(i)
```

```
1 2  
2 4  
3 6  
4 8
```

- Ο μετρητής **συνεχίζει** να υπάρχει και μετά το for:

```
for i in range(3):  
    print('skip')  
print(i)
```

```
skip  
skip  
skip  
2
```


break **και** continue

- Η εντολή `break` διακόπτει τις επαναλήψεις του `while` ή του `for`:

```
for i in range(3):  
    print(i)  
    break
```

0

- Η εντολή `continue` παραλείπει την τρέχουσα επανάληψη στο `while` ή στο `for`:

```
# Άθροισμα των περιττών αριθμών μέχρι το 20 με for  
  
sum = 0  
  
print('Το άθροισμα των αριθμών:', end = ' ')  
for i in range(21):  
    if i%2 == 0:  
        continue  
    else:  
        sum += i  
        print(i, end = ' ')  
print('\nΕίναι:', sum)
```

Το άθροισμα των αριθμών: 1 3 5 7 9 11 13 15 17 19
Είναι: 100

break **και** continue

- Η εντολή `break` διακόπτει τις επαναλήψεις του `while` ή του `for`:

```
for i in range(3):  
    print(i)  
    break
```

0

- Η εντολή `continue` παραλείπει την τρέχουσα επανάληψη στο `while` ή στο `for`:

```
# Άθροισμα των περιττών αριθμών μέχρι το 20 με while  
  
sum = 0  
  
print('Το άθροισμα των αριθμών:', end = ' ')  
  
i = 0  
while i < 20:  
    i += 1  
    if i%2 == 0:  
        continue  
    else:  
        sum += i  
        print(i, end = ' ')  
print('\nΕίναι:', sum)
```

else **ΣΤΟ** while **και ΣΤΟ** for;

Οι εντολές while και for μπορούν να έχουν και έναν κλάδο else που εκτελείται στο τέλος **μόνο αν** δεν έγινε διακοπή από break:



```
# Έλεγχος για το αν ένας αριθμός είναι πρώτος  
  
n = int(input('Δώσε έναν φυσικό αριθμό: '))  
  
for x in range(2, n//2):  
    if n%x == 0:  
        print('Ο', x, 'διαίρει τον', n)  
        break  
else:  
    print('Ο', n, 'είναι πρώτος')
```

```
Δώσε έναν φυσικό αριθμό: 1234  
Ο 2 διαίρει τον 1234
```

```
Δώσε έναν φυσικό αριθμό: 127  
Ο 127 είναι πρώτος
```

else **στο** while **και στο** for;

Οι εντολές while και for μπορούν να έχουν και έναν κλάδο else που εκτελείται στο τέλος **μόνο αν** δεν έγινε διακοπή από break:

```
# Έλεγχος για το αν ένας αριθμός είναι πρώτος

n = int(input('Δώσε έναν φυσικό αριθμό: '))

for x in range(2, n//2):
    if n%x == 0:
        print('Ο', x, 'διαιρεί τον', n)
        break
    else:
        print('Ο', n, 'είναι πρώτος')
```

expected an indented block after 'else' statement on line 9

else **ΣΤΟ** while **και ΣΤΟ** for;

Οι εντολές while και for μπορούν να έχουν και έναν κλάδο else που εκτελείται στο τέλος **μόνο αν** δεν έγινε διακοπή από break:

```
# Έλεγχος για το αν ένας αριθμός είναι πρώτος

n = int(input('Δώσε έναν φυσικό αριθμό: '))

for x in range(2, n//2):
    if n%x == 0:
        print('Ο', x, 'διαιρεί τον', n)
        break
    else:
        print('Ο', n, 'είναι πρώτος')
```

```
Δώσε έναν φυσικό αριθμό: 49
Ο 49 είναι πρώτος
Ο 49 είναι πρώτος
Ο 49 είναι πρώτος
Ο 49 είναι πρώτος
Ο 49 είναι πρώτος
Ο 7 διαιρεί τον 49
```

else **ΣΤΟ** while **και ΣΤΟ** for;

Οι εντολές while και for μπορούν να έχουν και έναν κλάδο else που εκτελείται στο τέλος **μόνο αν** δεν έγινε διακοπή από break:

```
# Έλεγχος για το αν ένας αριθμός είναι πρώτος

n = int(input('Δώσε έναν φυσικό αριθμό: '))

for x in range(2, n//2):
    if n%x == 0:
        print('Ο', x, 'διαιρεί τον', n)
        break
    print('Ο', n, 'είναι πρώτος')
```

```
Δώσε έναν φυσικό αριθμό: 49
Ο 49 είναι πρώτος
Ο 49 είναι πρώτος
Ο 49 είναι πρώτος
Ο 49 είναι πρώτος
Ο 49 είναι πρώτος
Ο 7 διαιρεί τον 49
```

else **στο** while **και στο** for;

Παραδοσιακός τρόπος:

```
# Έλεγχος για το αν ένας αριθμός είναι πρώτος  
n = int(input('Δώσε έναν φυσικό αριθμό: '))  
flag = True # Αν παραμείνει True είναι πρώτος  
  
for x in range(2, n//2):  
    if n%x == 0:  
        flag = False  
        break  
if not(flag):  
    print('Ο', x, 'διαίρει τον', n)  
else:  
    print('Ο', n, 'είναι πρώτος')
```

Με την Python μπορούμε να γράψουμε συντομότερο κώδικα!

Το Πρωτόκολλο Επανάληψης

Κάποια αντικείμενα στην Python μπορούν να δίνουν τα στοιχεία τους **ένα-ένα**:

```
# Λίστα  
L = [1, 2, 3]  
for i in L:  
    print(i, end = ' ')
```

1 2 3

```
# Αλληλουχίες  
L = range(1, 4)  
for i in L:  
    print(i, end = ' ')
```

1 2 3

```
# Συμβολοσειρές  
L = '123'  
for i in L:  
    print(i, end = ' ')
```

1 2 3

Τα αντικείμενα αυτά ακολουθούν το **Πρωτόκολλο Επανάληψης** (είναι **iterable**).

Το Πρωτόκολλο Επανάληψης

Πως δουλεύει:

```
>>> # Αρχικοποιούμε τρία iterable:
>>> L_1 = [1,2,3]
>>> L_2 = range(1,4)
>>> L_3 = '123'
```

```
>>> # Αρχικοποιούμε τρεις iterators με τη συνάρτηση iter():
>>> it_1 = iter(L_1)
>>> it_2 = iter(L_2)
>>> it_3 = iter(L_3)
```

```
>>> # Τρεις τύποι iterators:
>>> type(it_1)
<class 'list_iterator'>
>>> type(it_2)
<class 'range_iterator'>
>>> type(it_3)
<class 'str_iterator'>
```

```
>>> # Παίρνουμε το επόμενο στοιχείο με τη συνάρτηση next():
>>> next(it_1)
1
>>> next(it_1)
2
>>> next(it_1)
3
```

Πως Δουλεύει η for:

```
>>> next(it_1)
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    next(it_1)
StopIteration
```

Εξαίρεση με όνομα:
StopIteration

Παραδοσιακός τρόπος:

```
for i in range(1,4):
    print(i, end = ' ')
```

1 2 3

Στην πραγματικότητα «μέσα» σε ένα for γίνονται τα εξής

```
L = range(1,4) # Δημιουργία iterable
```

```
it = iter(L) # Δημιουργία iterator
```

```
while True:
```

```
    try:
```

```
        i = next(it)
```

```
    except StopIteration:
```

```
        break
```

```
    print(i, end = ' ')
```



try **μπλοκ**
(«δοκιμαστικός» κώδικας)

1 2 3

Περισσότερα για τα iterable

- Μπορούμε να ορίσουμε παραπάνω από έναν iterator:

```
>>> L = 'abcd'
>>> it_1 = iter(L)
>>> it_2 = iter(L)
>>> next(it_1)
'a'
>>> next(it_1)
'b'
>>> next(it_2)
'a'
>>> next(it_1)
'c'
>>> next(it_2)
'b'
```

- Αντί να «προκαλέσουμε» εξαίρεση μπορούμε να επιστρέψουμε κάποια συγκεκριμένη τιμή:

```
>>> it = iter([1,2])
>>> next(it, 'Τέλος')
1
>>> next(it, 'Τέλος')
2
>>> next(it, 'Τέλος')
'Τέλος'
>>> next(it, 'Τέλος')
'Τέλος'
```

Συναρτήσεις

- Ενσωματωμένες Συναρτήσεις
- Ορισμός Συναρτήσεων
- Pass-by-Object-Reference
- Καθολικές και Τοπικές Μεταβλητές
- Δημιουργία Module
- Διαδικαστικός Προγραμματισμός

Συναρτήσεις

Παράδειγμα:

```
x = 15  
y = 3  
z = 57  
w = max(x, y, z)  
w += 1  
w -= y  
print(w)
```

max(x, y, z)

Συναρτήσεις

Παράδειγμα:

```
x = 15  
y = 3  
z = 57  
w = max(x, y, z)  
w += 1  
w -= y  
print(w)
```



Όταν ένα πρόγραμμα καλεί μια συνάρτηση:

- Η ροή του μεταφέρεται προσωρινά στις εντολές της συνάρτησης
- Εκτελούνται οι εντολές της συνάρτησης
- Η ροή επιστρέφει στο πρόγραμμα

Ο κώδικας της συνάρτησης δεν μας απασχολεί!

Μας ενδιαφέρει μόνο:

- Το όνομά της
- Η είσοδός της (αν δέχεται κάποιο όρισμα)
- Η έξοδός της (αν επιστρέφει κάποια τιμή)

Ενσωματωμένες Συναρτήσεις

Πρότυπη βιβλιοθήκη

Συνάρτηση	Λειτουργία
<code>print()</code>	Εκτύπωση ορίσματος στην οθόνη
<code>input()</code>	Εκτύπωση ορίσματος στην οθόνη και επιστροφή της συμβολοσειράς που πληκτρολόγησε ο χρήστης
<code>abs()</code>	Επιστροφή τη απόλυτη τιμή ορίσματος
<code>min()/max()</code>	Επιστροφή ελάχιστου/μέγιστου μεταξύ των ορισμάτων
<code>pow()</code>	Επιστροφή της ύψωσης του πρώτου ορίσματος στο δεύτερο όρισμα
<code>round()</code>	Επιστροφή του ορίσματος στρογγυλοποιημένο στον κοντινότερο ακέραιο
<code>range()</code>	Επιστροφή αλληλουχίας αριθμών

Συναρτήσεις από module

- Εισαγωγή module: `import όνομα_module`
- Κλήση συνάρτησης: `όνομα_module.όνομα_συνάρτησης(παράμετροι)`

Παράδειγμα:

```
>>> import math
>>> math.sqrt(2)
1.4142135623730951
```


To module `math`

Συνάρτηση	Λειτουργία
<code>sqrt()</code>	Τετραγωνικής ρίζας ορίσματος
<code>log()</code>	Λογαρίθμου ορίσματος (με βάση e)
<code>log2()</code>	Λογαρίθμου ορίσματος (με βάση 2)
<code>log10()</code>	Λογαρίθμου ορίσματος (με βάση 10)
<code>exp()</code>	Ύψωσης του e στο όρισμα
<code>factorial()</code>	Παραγοντικού του ορίσματος
<code>floor()</code>	Κάτω ακέραιο μέρος ορίσματος
<code>ceil()</code>	Άνω ακέραιο μέρος ορίσματος
<code>sin()</code>	Ημίτονο ορίσματος
<code>cos()</code>	Συνημίτονο ορίσματος
<code>tan(x)</code>	Εφαπτομένη ορίσματος

Μπορούμε να καλέσουμε και σταθερές από ένα module

- Κλήση σταθεράς `όνομα_module.όνομα_σταθεράς`

Όνομα	Σταθερά
<code>pi</code>	π
<code>e</code>	Σταθερά Euler

Παράδειγμα:

```
>>> import math
>>> math.exp(4), math.log2(8), math.ceil(6.76), math.e
(54.598150033144236, 3.0, 7, 2.718281828459045)
```


To module math

```
# Το πρόγραμμα αυτό επιτρέπει στον χρήστη να επιλέξει διάφορους γεωμετρικούς υπολογισμούς
# από μενού

import math

# Επιλογές μενού
AREA_CIRCLE_CHOICE = 1
CIRCUMFERENCE_CHOICE = 2
QUIT_CHOICE = 3

choice = 0

while choice != QUIT_CHOICE:
    # Εμφάνιση μενού
    print('\n      ΜΕΝΟΥ')
    print('1) Εμβαδόν κύκλου')
    print('2) Περίμετρος κύκλου')
    print('3) Τερματισμός\n')

    # Επιλογή χρήστη
    choice = int(input('Δώσε την επιλογή σου: '))

    # Εμβαδόν κύκλου
    if choice == AREA_CIRCLE_CHOICE:
        print('\nΕμβαδόν κύκλου:')
        radius = float(input('Δώσε ακτίνα: '))
        print('Το εμβαδόν είναι ', math.pi * radius**2)

    # Περίμετρος κύκλου
    elif choice == CIRCUMFERENCE_CHOICE:
        print('\nΠερίμετρος κύκλου:')
        radius = float(input('Δώσε ακτίνα: '))
        print('Η περίμετρος είναι ', 2 * math.pi * radius)

    # Τερματισμός
    elif choice == QUIT_CHOICE:
        print('\nΤερματισμός...')

    # Λάθος είσοδος από χρήστη
    else:
        print('\nΣφάλμα: μη έγκυρη επιλογή!')
```

To module math

```
Python 3.10.1 (v3.10.1:2cd268a3a9, Dec 6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)]
on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/dzoros/Desktop/output.py =====

      ΜΕΝΟΥ
1) Εμβαδόν κύκλου
2) Περίμετρος κύκλου
3) Τερματισμός

Δώσε την επιλογή σου: 1

Εμβαδόν κύκλου:
Δώσε ακτίνα: 7.8
Το εμβαδόν είναι 191.134497044403

      ΜΕΝΟΥ
1) Εμβαδόν κύκλου
2) Περίμετρος κύκλου
3) Τερματισμός

Δώσε την επιλογή σου: 2

Περίμετρος κύκλου:
Δώσε ακτίνα: 3.45
Η περίμετρος είναι 21.676989309769574

      ΜΕΝΟΥ
1) Εμβαδόν κύκλου
2) Περίμετρος κύκλου
3) Τερματισμός

Δώσε την επιλογή σου: 7

Σφάλμα: μη έγκυρη επιλογή!

      ΜΕΝΟΥ
1) Εμβαδόν κύκλου
2) Περίμετρος κύκλου
3) Τερματισμός

Δώσε την επιλογή σου: 3

Τερματισμός...
>>> |
```

Ln: 45 Col: 0

To module `math`

```
# Το πρόγραμμα αυτό επιτρέπει στον χρήστη να επιλέξει διάφορους γεωμετρικούς υπολογισμούς  
# από μενού
```

```
import math
```

```
# Επιλογές μενού
```

```
AREA_CIRCLE_CHOICE = 1
```

```
CIRCUMFERENCE_CHOICE = 2
```

```
QUIT_CHOICE = 3
```

```
choice = 0
```

```
while choice != QUIT_CHOICE:
```

```
    # Εμφάνιση μενού
```

```
    print('\n      ΜΕΝΟΥ')
```

```
    print('1) Εμβαδόν κύκλου')
```

```
    print('2) Περίμετρος κύκλου')
```

```
    print('3) Τερματισμός\n')
```

```
    # Επιλογή χρήστη
```

```
    choice = int(input('Δώσε την επιλογή σου: '))
```

```
    # Εμβαδόν κύκλου
```

```
    if choice == AREA_CIRCLE_CHOICE:
```

```
        print('\nΕμβαδόν κύκλου:')
```

```
        radius = float(input('Δώσε ακτίνα: '))
```

```
        print('Το εμβαδόν είναι ', math.pi * radius**2)
```

```
    # Περίμετρος κύκλου
```

```
    elif choice == CIRCUMFERENCE_CHOICE:
```

```
        print('\nΠερίμετρος κύκλου:')
```

```
        radius = float(input('Δώσε ακτίνα: '))
```

```
        print('Η περίμετρος είναι ', 2 * math.pi * radius)
```

```
    # Τερματισμός
```

```
    elif choice == QUIT_CHOICE:
```

```
        print('\nΤερματισμός...')
```

```
    # Λάθος είσοδος από χρήστη
```

```
    else:
```

```
        print('\nΣφάλμα: μη έγκυρη επιλογή!')
```

Αν ο χρήστης δεν δώσει `int`
θα πετάξει εξαίρεση...

To module math

```
# Το πρόγραμμα αυτό επιτρέπει στον χρήστη να επιλέξει διάφορους γεωμετρικούς υπολογισμούς
# από μενού

import math

# Επιλογές μενού
AREA_CIRCLE_CHOICE = '1'
CIRCUMFERENCE_CHOICE = '2'
QUIT_CHOICE = '3'

choice = 0

while choice != QUIT_CHOICE:
    # Εμφάνιση μενού
    print('\n      ΜΕΝΟΥ')
    print('1) Εμβαδόν κύκλου')
    print('2) Περίμετρος κύκλου')
    print('3) Τερματισμός\n')

    # Επιλογή χρήστη
    choice = input('Δώσε την επιλογή σου: ')

    # Εμβαδόν κύκλου
    if choice == AREA_CIRCLE_CHOICE:
        print('\nΕμβαδόν κύκλου:')
        radius = float(input('Δώσε ακτίνα: '))
        print('Το εμβαδόν είναι ', math.pi * radius**2)

    # Περίμετρος κύκλου
    elif choice == CIRCUMFERENCE_CHOICE:
        print('\nΠερίμετρος κύκλου:')
        radius = float(input('Δώσε ακτίνα: '))
        print('Η περίμετρος είναι ', 2 * math.pi * radius)

    # Τερματισμός
    elif choice == QUIT_CHOICE:
        print('\nΤερματισμός...')

    # Λάθος είσοδος από χρήστη
    else:
        print('\nΣφάλμα: μη έγκυρη επιλογή!')
```

To module `random`

(βιβλιοθήκη για τη χρήση τυχαίων αριθμών)

- Φόρτωση module

```
>>> import random
```

- `randint(x, y)` : Τυχαίος **ακέραιος** στο διάστημα $[x,y]$

```
>>> random.randint(1, 100)
7
>>> random.randint(1, 100)
58
```

- `randrange(x)` : Τυχαίος **ακέραιος** στο διάστημα $[0,x)$
`randrange(x, y)` : Τυχαίος **ακέραιος** στο διάστημα $[x,y)$
`randrange(x, y, z)` : Τυχαίος **ακέραιος** στο διάστημα $[x,y)$ με βήμα z

```
>>> random.randrange(2, 10, 2) # Ακέραιο από το {2, 4, 6, 8}
4
```

- `random()` : Τυχαίος **πραγματικός** στο διάστημα $[0,1)$

```
>>> random.random()
0.20505638562339845
```

- `uniform(x, y)` : Τυχαίος **πραγματικός** στο διάστημα $[x,y)$

```
>>> random.uniform(1, 2)
1.9174661343422783
```

To module `random`

(βιβλιοθήκη για τη χρήση τυχαίων αριθμών)

```
# Το πρόγραμμα αυτό προσομοιώνει τη ρίψη ενός νομίσματος

import random

# 1 για κορόνα
# 2 για γράμματα

if random.randint(1,2) == 1:
    print('Κορόνα')

else:
    print('Γράμματα')
```

```
# Το πρόγραμμα αυτό προσομοιώνει τη ρίψη δύο ζαριών

import random

for i in ['Πρώτη ρίψη: ', 'Δεύτερη ρίψη: ']:
    print(i, random.randint(1,6))
```

```
Πρώτη ρίψη: 6
Δεύτερη ρίψη: 4
```

Το Seed Τυχαίων Αριθμών

Οι τυχαίοι αριθμοί δεν είναι στην πραγματικότητα τυχαίοι!

Προκύπτουν με ντετερμινιστικό τρόπο από κάποια αρχική τιμή (seed) που αλλάζει κάθε φορά που κάνουμε `import random`.

Στην Python το seed είναι η ώρα του συστήματος (ακρίβεια εκατοστού δευτερολέπτου).

- `seed(x)` : Θέτει το seed ίσο με x

```
>>> random.seed(1)
>>> random.random()
0.13436424411240122
>>> random.random()
0.8474337369372327
>>> random.random()
0.763774618976614
```

```
>>> random.seed(1)
>>> random.random()
0.13436424411240122
>>> random.random()
0.8474337369372327
>>> random.random()
0.763774618976614
```


Ορισμός Συναρτήσεων

- Συναρτήσεις void (δεν επιστρέφουν τιμή):

```
def όνομα_συνάρτησης() :  
    εντολή  
    εντολή  
    ...
```

Παράδειγμα:

```
def menu() :  
  
    # Εμφάνιση μενού  
    print('\n      ΜΕΝΟΥ')  
    print('1) Εμβαδόν κύκλου')  
    print('2) Περίμετρος κύκλου')  
    print('3) Τερματισμός\n')  
  
def main() :  
  
    # Επιλογές μενού  
    AREA_CIRCLE_CHOICE = '1'  
    CIRCUMFERENCE_CHOICE = '2'  
    QUIT_CHOICE = '3'  
  
    choice = 0  
  
    while choice != QUIT_CHOICE:  
        # Καλούμε την menu()  
        menu()  
        # Επιλογή χρήστη  
        choice = input('Δώσε την επιλογή σου: ')  
  
    # Καλούμε τη main()  
    main()
```

**Η σειρά ορισμού
δεν έχει σημασία!**
(Αρκεί η κλήση της `main()`
να είναι μετά τον ορισμό των
συναρτήσεων)

Ορισμός Συναρτήσεων

Κανόνες ονομασίας

- Δε μπορούν να χρησιμοποιηθούν δεσμευμένες λέξεις (λέξεις-κλειδιά):
`class, for, if, else, while...`
- Ξεκινάνε με γράμμα ή κάτω παύλα (`_`)
- Περιέχουν γράμματα, αριθμούς, `_` (όχι τελείες, `*`, `#` κτλ.)
- Όχι κενά
- Διάκριση μεταξύ κεφαλαίων και πεζών (case sensitive)

Σύμβαση:

- Περιγραφικά ονόματα
- Χωρισμός λέξεων με `_`
- Χρησιμοποιούμε ρήματα

Παράδειγμα:

```
def get_info():  
    name = input('Όνομα: ')  
    surname = input('Επώνυμο: ')  
    id_number = input('Αριθμός Ταυτότητας: ')
```

Μεταβίβαση Ορισμάτων

- Συναρτήσεις με είσοδο:

```
def όνομα_συνάρτησης(παράμετρος_1, παράμετρος_2,...):  
    εντολή  
    εντολή  
    ...
```

Κλήση: `όνομα_συνάρτησης(όρισμα_1, όρισμα_2,...)`

Παράδειγμα:

```
def main():  
    x = float(input('Δώσε τον 1ο αριθμό: '))  
    y = float(input('Δώσε τον 2ο αριθμό: '))  
    print_max(x, y)  
  
def print_max(a, b):  
    if a>b:  
        print('Το μέγιστο είναι το', a)  
    elif a<b:  
        print('Το μέγιστο είναι το', b)  
    else:  
        print('Οι τιμές είναι ίσες')  
main()
```

Pass-by-Value

Μία συνάρτηση μπορεί να αλλάζει την τιμή του ορίσματος της, η αλλαγή αυτή όμως **δεν μεταβιβάζεται** πίσω στην συνάρτηση που την κάλεσε:

```
def main():  
    value = 1  
    print('Η τιμή είναι: ', value)  
    change_it(value)  
    print('Όμως η τιμή είναι ακόμα: ', value)  
  
def change_it(arg):  
    arg += 1  
    print('Αλλαξα την τιμή σε: ', arg )  
  
main()
```

```
Η τιμή είναι: 1  
Αλλαξα την τιμή σε: 2  
Όμως η τιμή είναι ακόμα: 1
```

Στην `change_it` μεταβιβάζεται **μόνο η τιμή** του ορίσματος (μεταβίβαση με τιμή, `pass-by-value`).

Pass-by-Reference

Μία συνάρτηση αλλάζει την τιμή του ορίσματός της, η αλλαγή αυτή όμως μπορεί να **μεταβιβάζεται** πίσω στην συνάρτηση που την κάλεσε?!

```
def main():  
    value = [1,2,3] # Λίστα  
    print('Η τιμή είναι: ', value)  
    change_it(value)  
    print('Και η τιμή τελικά είναι: : ', value)  
  
def change_it(arg):  
    arg += [4,5,6] # Παράθεση λιστών  
    print('Αλλαξα την τιμή σε: ', arg )  
  
main()
```

```
Η τιμή είναι:  [1, 2, 3]  
Αλλαξα την τιμή σε:  [1, 2, 3, 4, 5, 6]  
Και η τιμή τελικά είναι: :  [1, 2, 3, 4, 5, 6]
```

Εδώ έχουμε μεταβίβαση της **αναφοράς στη θέση μνήμης** και όχι μόνο της τιμής (pass-by-reference)!

Pass-by-Value

```
def main():
    value = 1
    print('Η τιμή είναι: ', value)
    print('και είναι αποθηκευμένη στη διεύθυνση: ', id(value))
    change_it(value)
    print('Όμως η τιμή είναι ακόμα: ', value)
    print('και ακόμα είναι αποθηκευμένη στη διεύθυνση: ', id(value))

def change_it(arg):
    arg += 1
    print('Άλλαξα την τιμή σε: ', arg)
    print('και τώρα είναι αποθηκευμένη στη διεύθυνση: ', id(arg))

main()
```

```
Η τιμή είναι: 1
και είναι αποθηκευμένη στη διεύθυνση: 4413718768
Άλλαξα την τιμή σε: 2
και τώρα είναι αποθηκευμένη στη διεύθυνση: 4413718736
Όμως η τιμή είναι ακόμα: 1
και ακόμα είναι αποθηκευμένη στη διεύθυνση: 4413718768
```

* Η `id()` επιστρέφει ένα μοναδικό αναγνωριστικό για κάθε αντικείμενο (ας πούμε τη διεύθυνση της θέσης μνήμης που είναι αποθηκευμένο).

Pass-by-Reference

```
def main():
    value = [1,2,3] # Λίστα
    print('Η τιμή είναι: ', value)
    print('και είναι αποθηκευμένη στη διεύθυνση: ', id(value))
    change_it(value)
    print('Και η τιμή τελικά είναι: : ', value)
    print('και ακόμα είναι αποθηκευμένη στη διεύθυνση: ', id(value))

def change_it(arg):
    arg += [4,5,6] # Παράθεση λιστών
    print('Αλλαξα την τιμή σε: ', arg )
    print('και παραμένει αποθηκευμένη στη διεύθυνση: ', id(arg))

main()
```

```
Η τιμή είναι: [1, 2, 3]
και είναι αποθηκευμένη στη διεύθυνση: 4351449472
Αλλαξα την τιμή σε: [1, 2, 3, 4, 5, 6]
και παραμένει αποθηκευμένη στη διεύθυνση: 4351449472
Και η τιμή τελικά είναι: : [1, 2, 3, 4, 5, 6]
και ακόμα είναι αποθηκευμένη στη διεύθυνση: 4351449472
```

* Η `id()` επιστρέφει ένα μοναδικό αναγνωριστικό για κάθε αντικείμενο (ας πούμε τη διεύθυνση της θέσης μνήμης που είναι αποθηκευμένο).

Pass-by-Object-Reference!

```
def main():
    value = 1
    print('Η τιμή είναι: ', value)
    print('και είναι αποθηκευμένη στη διεύθυνση: ', id(value))
    change_it(value)
    print('Όμως η τιμή είναι ακόμα: ', value)
    print('και ακόμα είναι αποθηκευμένη στη διεύθυνση: ', id(value))

def change_it(arg):
    print('Το όρισμα της change_it πριν την αλλαγή είναι αποθηκευμένο')
    print('στη διεύθυνση: ', id(arg))
    arg += 1
    print('Αλλαξα την τιμή σε: ', arg)
    print('και τώρα είναι αποθηκευμένη στη διεύθυνση: ', id(arg))

main()
```

```
Η τιμή είναι: 1
και είναι αποθηκευμένη στη διεύθυνση: 4418584816
Το όρισμα της change_it πριν την αλλαγή είναι αποθηκευμένο
στη διεύθυνση: 4418584816
Αλλαξα την τιμή σε: 2
και τώρα είναι αποθηκευμένη στη διεύθυνση: 4418584784
Όμως η τιμή είναι ακόμα: 1
και ακόμα είναι αποθηκευμένη στη διεύθυνση: 4418584816
```

* Η `id()` επιστρέφει ένα μοναδικό αναγνωριστικό για κάθε αντικείμενο (ας πούμε τη διεύθυνση της θέσης μνήμης που είναι αποθηκευμένο).

Γιατί γίνεται αυτό;

Αντικείμενα που **δεν επιδέχονται** αλλαγές
(αμετάβλητα-Immutable):

int
float
bool
str
range
tuple

```
>>> x = 1
>>> id(x)
4523131120
>>> x += 1
>>> id(x)
4523131152
```

Πλεονέκτημα:

- Γρηγορότερη πρόσβαση

Αν τα ορίσματα είναι Immutable
έχουμε pass-by-value

Αντικείμενα που **επιδέχονται** αλλαγές
(μεταβλητά-Mutable):

list
dict
set

```
>>> L = [1, 2, 3]
>>> id(L)
4571857536
>>> L += [1, 2, 3]
>>> id(L)
4571857536
```

Πλεονέκτημα:

- Λιγότερη μνήμη κατά τις αλλαγές

Αν τα ορίσματα είναι Mutable
έχουμε κάτι σαν* pass-by-reference

* Αν αλλάζουμε το αντικείμενο και δεν δημιουργούμε καινούργιο

Μεταβίβαση Ορισμάτων κατά Θέση

Μεταβίβαση κατά θέση:

```
def όνομα_συνάρτησης(παράμετρος_1, παράμετρος_2,...):  
    εντολή  
    εντολή  
    ...
```



Κλήση: `όνομα_συνάρτησης(όρισμα_1, όρισμα_2,...)`

Παράδειγμα:

```
def function(a,b):  
    print('Πρώτο όρισμα', a, '/ Δεύτερο όρισμα', b)
```

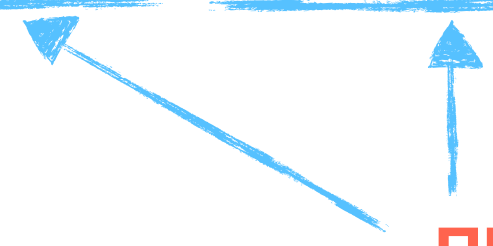
```
function(1,2)  
function(2,1)
```

```
Πρώτο όρισμα 1 / Δεύτερο όρισμα 2  
Πρώτο όρισμα 2 / Δεύτερο όρισμα 1
```

Προεπιλεγμένες Τιμές Ορισμάτων

Μπορούμε να ορίσουμε προεπιλεγμένες τιμές για τα ορίσματα :

```
def όνομα_συνάρτησης(παράμετρος_1, παράμετρος_2 = τιμή, ...) :  
    εντολή  
    εντολή  
    ...
```



ΠΡΟΣΟΧΗ!

```
def exponential(a=2, b) :  
    print(a**b)
```

Λάθος!

«Κανονικές» παράμετροι **πριν**
από τις «προεπιλεγμένες»

```
def exponential(b, a=2) :  
    print(a**b)
```

```
exponential(3)  
exponential(3, 4)
```

```
8  
64
```

```
def exponential(a=2, b=3) :  
    print(a**b)
```

```
exponential()  
exponential(3)
```

```
8  
27
```

Μεταβίβαση με Ονόματα Ορισμάτων

Μπορούμε επίσης να μεταβιβάσουμε τα ορίσματα **κατά όνομα** (λέξεις κλειδιά) και **όχι κατά θέση**:

```
def exponential(a, b):  
    print(a**b)  
  
exponential(b=2, a=3)
```

9

ΠΡΟΣΟΧΗ! Στην κλήση τα «κανονικά» ορίσματα πριν από τα ορίσματα με λέξεις κλειδιά!

```
>>> exponential(b=3, 2)
```

```
SyntaxError: positional argument follows keyword argument
```

```
>>> exponential(3, a=2)
```

```
Traceback (most recent call last):  
  File "<pyshell#6>", line 1, in <module>  
    exponential(2, a=3)  
TypeError: exponential() got multiple values for argument 'a'
```

```
>>> exponential(2, b=3)
```

8

Μεταβίβαση με Ονόματα Ορισμάτων

Παράδειγμα:

```
# Το πρόγραμμα αυτό διαβάζει την αρχική τιμή ενός προϊόντος  
# και υπολογίζει τη μειωμένη τιμή του, με έκπτωση 20%.  
  
DISCOUNT = 0.2  
  
def main():  
    original_price = float(input('Δώσε την αρχική τιμή: '))  
    show_final_price(price=original_price , discount=DISCOUNT)  
  
def show_final_price(discount, price):  
    price -= price * discount  
    print('Η τελική τιμή πώλησης είναι', price)  
  
main()
```

```
Δώσε την αρχική τιμή: 253  
Η τελική τιμή πώλησης είναι 202.4
```

Μικτή Μεταβίβαση Ορισμάτων

```
def mixed(a, b='b', c='c'):  
    print('a =', a, ', b =', b, ', c =', c)
```

```
>>> mixed(1,2)  
a = 1 , b = 2 , c = c
```

```
>>> mixed(1,c=2)  
a = 1 , b = b , c = 2
```

```
>>> mixed(1,c=2, b=3)  
a = 1 , b = 3 , c = 2
```

```
>>> mixed(c=2, b=3, 1) Λάθος: «Κανονικά» μετά από ορίσματα με λέξεις κλειδιά
```

```
>>> mixed(1, b=2, 3) Λάθος: «Κανονικά» μετά από ορίσματα με λέξεις κλειδιά
```

```
>>> mixed(1)  
a = 1 , b = b , c = c
```

```
>>> mixed(b=1) Λάθος: Δεν δώσαμε τιμή στην παράμετρο a
```

```
>>> mixed(1,2,b=3) Λάθος: Δώσαμε δύο τιμές στην παράμετρο b
```

Επιστροφή Τιμών (return)

Η επικοινωνία της καλούμενης συνάρτησης με την συνάρτηση που την κάλεσε* γίνεται με την εντολή `return`:

```
def όνομα_συνάρτησης(παράμετρος_1, παράμετρος_2, ...):  
    εντολή  
    εντολή  
    ...  
    return όνομα_μεταβλητής ή έκφραση
```

Παράδειγμα:

```
def my_sum(n=0, m=0):  
    result = n+m  
    return result
```

ή :

```
def my_sum(n=0, m=0):  
    return n+m
```

Κλήση:

```
x = my_sum(1, 2) # Εκχωρούμε την τιμή που επιστρέφει σε μεταβλ.
```

* πέρα από τις περιπτώσεις pass-by-reference...

Επιστροφή Τιμών (return)

Παράδειγμα: (πολλά return)

```
def find_max(a=0,b=0):  
    if a > b:  
        return a  
    else:  
        return b
```

Παράδειγμα: (return διαφορετικού τύπου)

```
import math  
  
def my_sqrt(x=0):  
    if x >= 0:  
        return math.sqrt(x)  
    else:  
        return 'Ο αριθμός είναι αρνητικός!'
```

Παράδειγμα: (μεταβίβαση τιμής από συνάρτηση **χωρίς** return)

```
def do_nothing():  
    x = True
```

```
>>> x = do_nothing()  
>>> print(x)  
None  
>>> type(x)  
<class 'NoneType'>
```


Τοπικές και Καθολικές Μεταβλητές

Εμβέλεια (scope): Το τμήμα του προγράμματος από το οποίο είναι προσβάσιμη η μεταβλητή

Τοπικές μεταβλητές:

```
def match_check():  
    return get_shirt_color() == get_pants_color()  
  
def get_shirt_color():  
    color = input('Χρώμα μπλούζας; ')  
    return color  
  
def get_pants_color():  
    color = input('Χρώμα παντελονιού; ')  
    return color
```

Εμβέλεια μόνο μέσα στη συνάρτηση

Τοπικές και Καθολικές Μεταβλητές

Εμβέλεια (scope): Το τμήμα του προγράμματος από το οποίο είναι προσβάσιμη η μεταβλητή

Καθολικές μεταβλητές:

```
COLOR = 'Μαύρο'
```

```
def match_check():  
    return get_shirt_color() == COLOR and \  
           get_pants_color() == COLOR
```

```
def get_shirt_color():  
    color = input('Χρώμα μπλούζας; ')  
    return color
```

```
def get_pants_color():  
    color = input('Χρώμα παντελονιού; ')  
    return color
```

Εμβέλεια σε όλο το πρόγραμμα

Εμβέλεια μόνο μέσα στη συνάρτηση

Τοπικές και Καθολικές Μεταβλητές

Εμβέλεια (scope): Το τμήμα του προγράμματος από το οποίο είναι προσβάσιμη η μεταβλητή

Καθολικές μεταβλητές:

```
COLOR = 'Μαύρο' # Καθολική
```

```
def match_check():
```

```
    return get_shirt_color() == COLOR and \
           get_pants_color() == COLOR
```

} Short-Circuit Αποτίμηση!

```
def get_shirt_color():
```

```
    color = input('Χρώμα μπλούζας; ') # Τοπική
    return color
```

```
def get_pants_color():
```

```
    color = input('Χρώμα παντελονιού; ') # Τοπική
    return color
```

```
>>> x = match_check()
```

```
Χρώμα μπλούζας; Μπλε
```

```
>>> x
```

```
False
```

Τοπικές και Καθολικές Μεταβλητές

Παράδειγμα:

```
x = 1    # Καθολική μεταβλητή

def func(x):
    print('Το καθολικό x είναι:', x)
    x = 2 # Τοπική μεταβλητή
    print('Άλλαξα το τοπικό x σε:', x)

func(x)
print('Το καθολικό x είναι ακόμα:', x)
```

```
Το καθολικό x είναι: 1
Άλλαξα το τοπικό x σε: 2
Το καθολικό x είναι ακόμα: 1
```

Τοπικές και Καθολικές Μεταβλητές

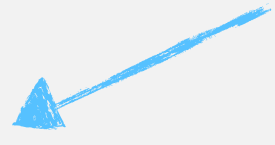
Παράδειγμα:

```
x = 1 # Καθολική μεταβλητή

def func():
    print('Το καθολικό x είναι:', x)
    x = 2 # Τοπική μεταβλητή
    print('Αλλάξα το τοπικό x σε:', x)

func()
print('Το καθολικό x είναι ακόμα:', x)
```

**Αναφέρεται στην
τοπική μεταβλητή x**



```
Traceback (most recent call last):
  File "/Users/dzoros/Desktop/script.py", line 8, in <module>
    func()
  File "/Users/dzoros/Desktop/script.py", line 4, in func
    print('Το καθολικό x είναι:', x)
UnboundLocalError: local variable 'x' referenced before
assignment
```

Τοπικές και Καθολικές Μεταβλητές

Παράδειγμα:

```
x = 1  # Καθολική μεταβλητή

def func():
    # print('Το καθολικό x είναι:', x)
    x = 2  # Τοπική μεταβλητή
    print('Αλλάξα το τοπικό x σε:', x)

func()
print('Το καθολικό x είναι ακόμα:', x)
```

```
Αλλάξα το τοπικό x σε: 2
Το καθολικό x είναι ακόμα: 1
```

Τοπικές και Καθολικές Μεταβλητές

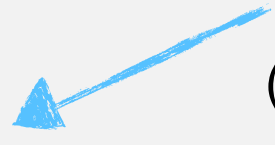
Παράδειγμα:

```
x = 1 # Καθολική μεταβλητή

def func():
    print('Το καθολικό x είναι:', x)
    # x = 2 # Τοπική μεταβλητή
    print('Αλλαξα το τοπικό x σε:', x)

func()
print('Το καθολικό x είναι ακόμα:', x)
```

**Αναφέρεται στην
καθολική μεταβλητή x**
(δεν υπάρχει τοπική μεταβλητή)



```
Το καθολικό x είναι: 1
Αλλαξα το τοπικό x σε: 1
Το καθολικό x είναι ακόμα: 1
```

Τοπικές και Καθολικές Μεταβλητές

Παράδειγμα:

```
x = 1  # Καθολική μεταβλητή

def func():
    global x
    print('Το καθολικό x είναι:', x)
    x = 2
    print('Αλλαξα το τοπικό x σε:', x)

func()
print('Το καθολικό x άλλαξε σε:', x)
```

**Αναφέρεται στην
καθολική μεταβλητή x**
(δεν υπάρχει τοπική μεταβλητή)

```
Το καθολικό x είναι: 1
Αλλαξα το τοπικό x σε: 2
Το καθολικό x άλλαξε σε: 2
```

Καθολικές μεταβλητές:

Τις αποφεύγουμε

Καθολικές σταθερές:
(καθολικές μεταβλητές χωρίς `global`)

Χρήσιμες (αν δεν θέλουμε να μεταβάλλεται
η τιμή τους από τις συναρτήσεις)

Τοπικές και Καθολικές Μεταβλητές

Παράδειγμα:

```
import random

REPS = 1000 # Καθολική σταθερά
heads_count = 0 # Καθολική μεταβλητή
tails_count = 0 # Καθολική μεταβλητή

# Διεξαγωγή πειράματος
def do_experiment():
    global heads_count, tails_count # Χρειάζεται γιατί αλλάζουμε τις μεταβλητές
    for i in range(REPS):
        if toss_coin() == 'Κορόνα':
            heads_count += 1 # Αν είναι τοπική, αλλαγή σε μη ορισμένη μεταβλητή!
        else:
            tails_count += 1

# Προσομοίωση ρίψης ενός νομίσματος
def toss_coin():
    if random.randint(1,2) == 1:
        return 'Κορόνα'

    else:
        return 'Γράμματα'

# Εκτύπωση αποτελεσμάτων
def print_results():
    # Εδώ δεν χρειάζεται το global γιατί δεν έχουμε τοπική μεταβλητή
    print('Ποσοστό ρίψεων που ήρθε κορόνα:', format(heads_count/REPS, '.2%'))
    print('Ποσοστό ρίψεων που ήρθαν γράμματα:', format(tails_count/REPS, '.2%'))
```

```
>>> do_experiment()
>>> print_results()
Ποσοστό ρίψεων που ήρθε κορόνα: 50.40%
Ποσοστό ρίψεων που ήρθαν γράμματα: 49.60%
```


Τοπικές και Καθολικές Μεταβλητές

Παράδειγμα:

```
import random

REPS = 1000 # Καθολική σταθερά

# Διεξαγωγή πειράματος
def do_experiment():
    heads_count = 0 # Τοπική
    tails_count = 0 # Τοπική
    for i in range(REPS):
        if toss_coin() == 'Κορόνα':
            heads_count += 1
        else:
            tails_count += 1

# Προσομοίωση ρίψης ενός νομίσματος
def toss_coin():
    if random.randint(1,2) == 1:
        return 'Κορόνα'

    else:
        return 'Γράμματα'

# Εκτύπωση αποτελεσμάτων
def print_results():
    print('Ποσοστό ρίψεων που ήρθε κορόνα:', format(heads_count/REPS, '.2%'))
    print('Ποσοστό ρίψεων που ήρθαν γράμματα:', format(tails_count/REPS, '.2%'))
```

```
>>> do_experiment()
>>> print_results()
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    print_results()
  File "/Users/dzoros/Desktop/script.py", line 25, in print_results
    print('Ποσοστό ρίψεων που ήρθε κορόνα:', format(heads_count/REPS, '.2%'))
NameError: name 'heads_count' is not defined
```

Τοπικές και Καθολικές Μεταβλητές

Παράδειγμα:

```
import random

REPS = 1000 # Καθολική σταθερά

# Διεξαγωγή πειράματος
def do_experiment():
    heads_count = 0 # Τοπική
    tails_count = 0 # Τοπική
    for i in range(REPS):
        if toss_coin() == 'Κορόνα':
            heads_count += 1
        else:
            tails_count += 1
    return heads_count, tails_count # Επιστροφή πολλαπλών τιμών

# Προσομοίωση ρίψης ενός νομίσματος
def toss_coin():
    if random.randint(1,2) == 1:
        return 'Κορόνα'

    else:
        return 'Γράμματα'

# Εκτύπωση αποτελεσμάτων
def print_results(heads_count, tails_count):
    print('Ποσοστό ρίψεων που ήρθε κορόνα:', format(heads_count/REPS, '.2%'))
    print('Ποσοστό ρίψεων που ήρθαν γράμματα:', format(tails_count/REPS, '.2%'))
```

```
>>> heads, tails = do_experiment()
>>> print(heads,tails)
517 483
>>> print_results(heads, tails)
Ποσοστό ρίψεων που ήρθε κορόνα: 51.70%
Ποσοστό ρίψεων που ήρθαν γράμματα: 48.30%
```

Συναρτήσεις με Ορίσματα Συναρτήσεις

```
>>> heads, tails = do_experiment()
>>> print(heads, tails)
517 483
>>> print_results(heads, tails)
Ποσοστό ρίψεων που ήρθε κορόνα: 51.70%
Ποσοστό ρίψεων που ήρθαν γράμματα: 48.30
>>> type(do_experiment())
<class 'tuple'>
>>> type(do_experiment)
<class 'function'>
```

Η Τιμή που επιστρέφει είναι μία πλειάδα
(περισσότερα αργότερα)

Οι συναρτήσεις είναι αντικείμενα τύπου `function`!

Μπορούν να είναι όρισμα σε συνάρτηση (όπως τα αντικείμενα `int`, `float` κλπ.):

```
# Υπολογίζει την προσέγγιση της παραγώγου της f στο σημείο x
# (h είναι η ακρίβεια)
def derivative(f, x, h):
    return (f(x+h) - f(x))/h

# Η συνάρτηση 2x^2-3x+5
def g(x):
    return 2*x**2 - 3*x + 5

y = derivative(g, 1, 0.1)
print(y)

y = derivative(g, 1, 0.01)
print(y)

y = derivative(g, 1, 0.001)
print(y)
```

```
1.2000000000000001
1.0199999999999932
1.00199999999997253
```

Δημιουργία Module

Μπορούμε να αποθηκεύσουμε τις συναρτήσεις μας σε ένα module και έπειτα να το κάνουμε `import` στο πρόγραμμά μας:

Δημιουργία module: `όνομα_module.py` (Είναι ένα απλό αρχείο .py)

Φόρτωση module: `import όνομα_module` Το module θα πρέπει να βρίσκεται στον ίδιο φάκελο με το αρχείο που το χρησιμοποιεί

Κλήση συνάρτησης: `όνομα_module.όνομα_συνάρτησης()`

Παράδειγμα: (my_module.py)

```
def get_username():
    return input('Όνομα χρήστη: ')

def get_password():
    return input('Κωδικός: ')

def register():
    name = get_username()
    password = get_password()
    return [name, password] # Δημιουργία λίστας

def change_password(user):
    print('-- Αλλαγή κωδικού --')
    user[1] = get_password() # Αλλαγή 2ου στοιχείου λίστας
    return user
```

Δημιουργία Module

Μπορούμε να αποθηκεύσουμε τις συναρτήσεις μας σε ένα module και έπειτα να το κάνουμε `import` στο πρόγραμμά μας:

Δημιουργία module: `όνομα_module.py` (Είναι ένα απλό αρχείο .py)

Φόρτωση module: `import όνομα_module` Το module θα πρέπει να βρίσκεται στον ίδιο φάκελο με το αρχείο που το χρησιμοποιεί

Κλήση συνάρτησης: `όνομα_module.όνομα_συνάρτησης()`

Παράδειγμα: (script.py)

```
import my_module

user = my_module.register()

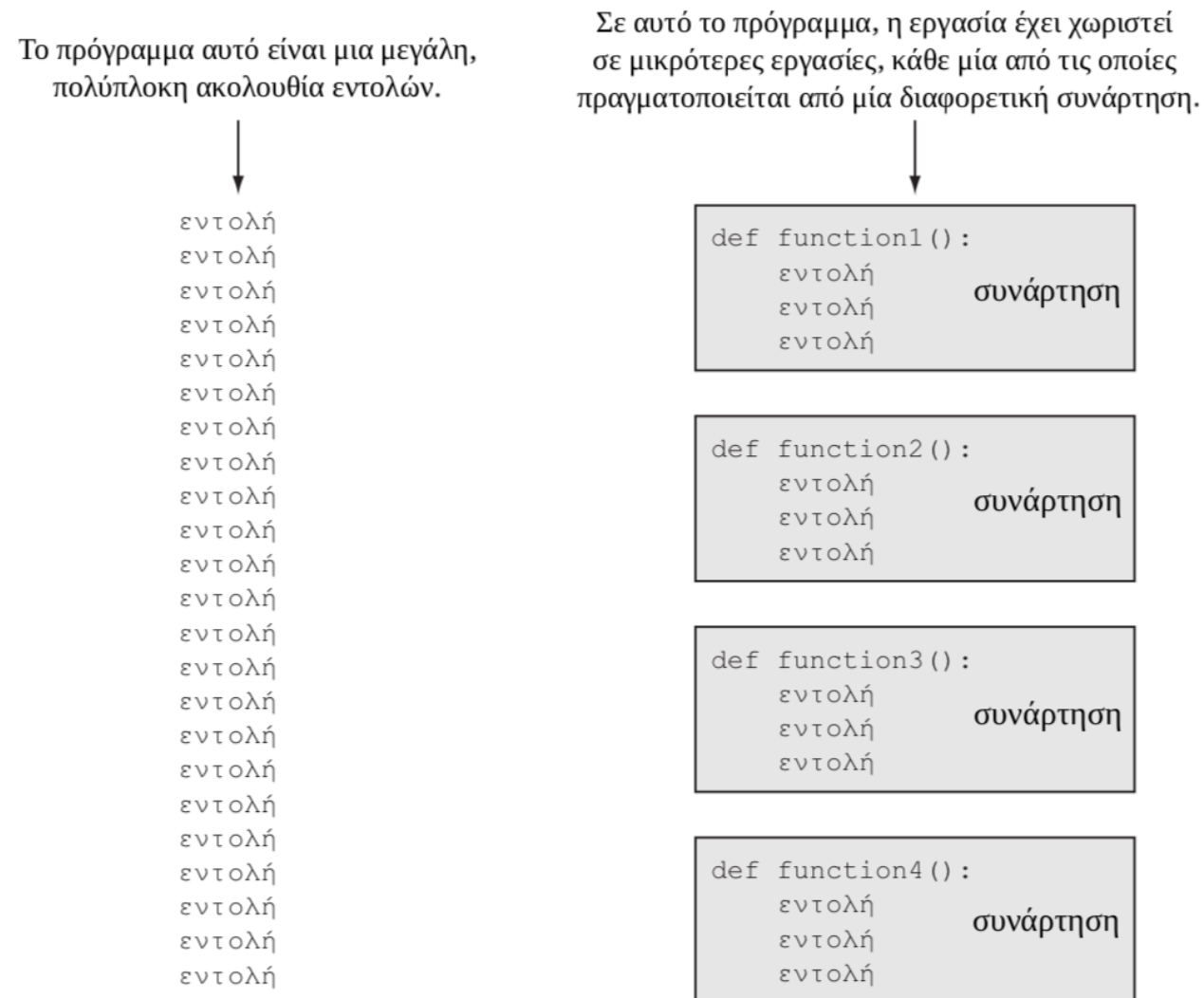
user = my_module.change_password(user)

print('Username:', user[0])
print('Password:', user[1])
```

```
Όνομα χρήστη: dzoros
Κωδικός: 1234
-- Αλλαγή κωδικού --
Κωδικός: 0000
Username: dzoros
Password: 0000
```

Διαδικαστικός Προγραμματισμός

- Χωρισμός προβλήματος σε μικρότερα υποπροβλήματα και επίλυσή τους ξεχωριστά με τον ορισμό συναρτήσεων



- Το πρόγραμμα στην ουσία είναι μια αλληλουχία συναρτήσεων

Πλεονεκτήματα

- Απλούστερος κώδικας
- Επαναχρησιμοποίηση κώδικα (module)
- Ευκολότερη αποσφαλμάτωση
- Ομαδική ανάπτυξη προγράμματος

Αλλά:

- **Οι συναρτήσεις δρουν πάνω στα δεδομένα**
(συναρτήσει και δεδομένα είναι δύο διαφορετικά πράγματα)
- **Συνήθως χρησιμοποιούμε μόνο τους βασικούς τύπους δεδομένων**

Στον **αντικειμενοστραφή προγραμματισμό** έχουμε ενοποίηση συναρτήσεων και δεδομένων μέσα στα αντικείμενα (που αποτελούν σύνθετους τύπους δεδομένων).

Παράδειγμα

```
import math

# Επιλογές μενού
AREA_CIRCLE_CHOICE = 1
CIRCUMFERENCE_CHOICE = 2
QUIT_CHOICE = 3

choice = 0

while choice != QUIT_CHOICE:
    # Εμφάνιση μενού
    print('\n      ΜΕΝΟΥ')
    print('1) Εμβαδόν κύκλου')
    print('2) Περίμετρος κύκλου')
    print('3) Τερματισμός\n')

    # Επιλογή χρήστη
    choice = int(input('Δώσε την επιλογή σου: '))

    # Εμβαδόν κύκλου
    if choice == AREA_CIRCLE_CHOICE:
        print('\nΕμβαδόν κύκλου:')
        radius = float(input('Δώσε ακτίνα: '))
        print('Το εμβαδόν είναι ', math.pi * radius**2)

    # Περίμετρος κύκλου
    elif choice == CIRCUMFERENCE_CHOICE:
        print('\nΠερίμετρος κύκλου:')
        radius = float(input('Δώσε ακτίνα: '))
        print('Η περίμετρος είναι ', 2 * math.pi * radius)

    # Τερματισμός
    elif choice == QUIT_CHOICE:
        print('\nΤερματισμός...')

    # Λάθος είσοδος από χρήστη
    else:
        print('\nΣφάλμα: μη έγκυρη επιλογή!')
```

Επαναλαμβάνονται

Παράδειγμα

```
# module με τις βασικές συναρτήσεις
# Αποθήκευση ως geometry.py

import math

def menu():
    # Εμφάνιση μενού
    print('\n      ΜΕΝΟΥ')
    print('1) Εμβαδόν κύκλου')
    print('2) Περίμετρος κύκλου')
    print('3) Τερματισμός\n')

def get_choice():
    # Είσοδος επιλογή χρήστη
    return int(input('Δώσε την επιλογή σου: '))

def get_radius():
    # Είσοδος ακτίνας
    return float(input('Δώσε ακτίνα: '))

def compute_area():
    # Υπολογισμός εμβαδού
    print('\nΕμβαδόν κύκλου:')
    radius = get_radius()
    print('Το εμβαδόν είναι ', math.pi * radius**2)

def compute_circ():
    # Υπολογισμός περιμέτρου
    print('\nΠερίμετρος κύκλου:')
    radius = get_radius()
    print('Η περίμετρος είναι ', 2 * math.pi * radius)

def quit():
    # Τερματισμός
    print('\nΤερματισμός...')

def wrong_answer():
    # Μήνυμα μη έγκυρης επιλογής
    print('\nΣφάλμα: μη έγκυρη επιλογή!')
```

Παράδειγμα

```
import geometry

# Επιλογές μενού
AREA_CIRCLE_CHOICE = 1
CIRCUMFERENCE_CHOICE = 2
QUIT_CHOICE = 3

choice = 0
while choice != QUIT_CHOICE:
    # Εμφάνιση μενού
    geometry.menu()

    # Επιλογή χρήστη
    choice = geometry.get_choice()

    if choice == AREA_CIRCLE_CHOICE:
        # Υπολογισμός εμβαδού
        geometry.compute_area()

    elif choice == CIRCUMFERENCE_CHOICE:
        # Υπολογισμός περιμέτρου
        geometry.compute_circ()

    elif choice == QUIT_CHOICE:
        # Τερματισμός
        geometry.quit()

    else:
        # Μήνυμα μη έγκυρης επιλογής
        geometry.wrong_answer()
```

Λίστες και Πλειάδες

- Δημιουργία Λίστας
- Προσπέλαση Στοιχείων Λίστας
- Πράξεις Πάνω σε Λίστες
- Μέθοδοι Λιστών
- Δισδιάστατες Λίστες
- Πλειάδες

Δημιουργία Λίστας

Σε μία λίστα μπορούμε να αποθηκεύσουμε πολλά δεδομένα (στοιχεία λίστας).

- Κενή λίστα:

λίστα = []

```
>>> L = []  
>>> type(L)  
<class 'list'>
```

- Άμεση δημιουργία λίστας:

λίστα = [στοιχείο_1, στοιχείο_2, ...]

```
>>> L = [1,2,3]  
>>> L  
[1,2,3]
```

- Με τον τελεστή επανάληψης *:

λίστα = [στοιχείο_1, στοιχείο_2, ...] * πλήθος επαναλήψεων

```
>>> L = [1,2,3] * 3  
>>> L  
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Διαφορές με Πίνακες (arrays)

Σε μία λίστα μπορούμε να αποθηκεύσουμε δεδομένα διαφορετικού τύπου:

```
>>> L = [1, 2.0 , 'τρία']  
>>> L  
[1, 2.0, 'τρία']
```

Παράδειγμα:

```
name = input('Όνομα φοιτητή: ')  
id_number = int(input('Αριθμός μητρώου: '))  
grade = float(input('Βαθμός: '))  
  
student = [name, id_number, grade]  
  
print(student)
```

```
Όνομα φοιτητή: Γιάννης  
Αριθμός μητρώου: 111220220000  
Βαθμός: 4.5  
['Γιάννης', 111220220000, 4.5]
```

Χρειαζόμαστε πίνακες;

Η αλήθεια είναι πως όχι... **Αλλά:**

- Οι λίστες δεν υποστηρίζουν αριθμητικές πράξεις.
- Οι πίνακες είναι πολύ πιο «γρήγοροι» (καθώς αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη).

Μετατροπή σε Λίστα

Μπορούμε να μετατρέψουμε αντικείμενα που ακολουθούν το **πρωτόκολλο επανάληψης** σε λίστες με τη συνάρτηση `list()`:

```
>>> # Λίστα  
>>> L = [1, 2, 3]  
>>> L_1 = list(L)  
>>> L_1  
[1, 2, 3]
```

Δημιουργεί αντίγραφο της λίστας...

```
>>> # Αλληλουχίες  
>>> L = list(range(1, 4))  
>>> L  
[1, 2, 3]
```

```
>>> # Συμβολοσειρές  
>>> L = list('123')  
>>> L  
['1', '2', '3']
```

```
>>> # Κάτι άλλο;  
>>> L = list(123)  
Traceback (most recent call last):  
  File "<pyshell#22>", line 1, in <module>  
    L = list(123)  
TypeError: 'int' object is not iterable
```

Προσπέλαση Στοιχείων Λίστας

- Με for:

```
>>> L = [1,2,3]
>>> for i in L:
        print(i, end = ' ')
1 2 3
```

- Με δείκτες:

Κάθε στοιχείο της λίστας έχει ένα δείκτη, ανάλογα με τη θέση του στη λίστα (η αρίθμηση των δεικτών **ξεκινάει από το 0**). Μπορούμε να προσπελάσουμε κάποιο στοιχείο της λίστας γράφοντας:

λίστα[δείκτης_στοιχείου]

```
>>> print(L[0], L[1], L[2])
1 2 3
```

Αρνητικοί δείκτες;

```
>>> print(L[-1], L[-2], L[-3])
3 2 1
```

Η συνάρτηση `len()` επιστρέφει το πλήθος στοιχείων της λίστας (μήκος λίστας):

```
>>> n = len(L)
>>> print(L[n-1], L[n-2], L[n-3])
3 2 1
```


Προσπέλαση Στοιχείων Λίστας

Μη έγκυρη τιμή δείκτη:

```
>>> n = len(L)
>>> print(L[n+1])
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    print(L[n+1])
IndexError: list index out of range
```

- Με δείκτες και `for`: (αν για κάποιον λόγο χρειαζόμαστε και τον δείκτη)

```
>>> L = [1,2,3]
>>> for i in L:
    print(i, end = ' ')
1 2 3
>>> # Εναλλακτικά με δείκτες:
>>> for i in range(len(L)):
    print(i, ': ', L[i], end = '\t')
0 : 1    1 : 2    2 : 3
>>> # ή με while:
>>> i = 0
>>> while i < len(L):
    print(i, ': ', L[i], end = '\t')
    i += 1
0 : 1    1 : 2    2 : 3
```


Προσπέλαση Στοιχείων Λίστας

Παράδειγμα:

```
def get_info():  
    name = input('Όνομα φοιτητή: ')  
    id_number = int(input('Αριθμός μητρώου: '))  
    grade = float(input('Βαθμός: '))  
    student = [name, id_number, grade]  
  
    return student # Επιστρέφει λίστα  
  
def print_result(student): # Δέχεται σαν όρισμα λίστα  
    passed = (student[2] >= 4.5)  
  
    print('O', student[0], end = ' ')  
  
    if passed:  
        print('πέρασε το μάθημα!')  
    else:  
        print('δεν πέρασε το μάθημα...')  
  
def main():  
    student = get_info()  
    print_result(student)  
  
main()
```

Όνομα φοιτητή: **Γιάννης**
Αριθμός μητρώου: **111220220000**
Βαθμός: **4.5**
Ο Γιάννης πέρασε το μάθημα!

Εκχώρηση Τιμών σε Λίστα

Οι λίστες είναι **mutable** τύπος δεδομένων, δηλαδή οι τιμές τους μπορούν να αλλάξουν:

$$\text{λίστα}[\text{δείκτης_στοιχείου}] = \text{τιμή}$$

```
>>> L = [1, 2, 3]
>>> L[0] = 'Ένα'
>>> L
['Ένα', 2, 3]
```

Μπορούμε να αρχικοποιήσουμε τα στοιχεία μιας λίστας με τους ακόλουθους τρόπους:

- Δίνοντας τα στοιχεία απευθείας:

```
>>> L = [1, 2, 3]
```

- Δημιουργώντας λίστα (με προκαθορισμένες τιμές) και έπειτα δίνοντας τα στοιχεία ένα ένα:

```
>>> L = [0]*3 # Προσοχή: []*3 == []
>>> L[0] = 1
>>> L[1] = 2
>>> L[2] = 3
```

Εκχώρηση Τιμών σε Λίστα

Ο δεύτερος τρόπος είναι χρήσιμος αν δεν ξέρουμε εξαρχής το πλήθος στοιχείων της λίστας:

```
# Συνάρτηση που δημιουργεί μια λίστα για ψώνια
def main():
    no_of_items = int(input('Πόσα αντικείμενα; '))
    L = create_list(no_of_items)
    L = get_items(L)
    print_list(L)

# Δημιουργία λίστας
def create_list(no_of_items):
    return [0] * no_of_items

# Είσοδος αντικειμένων
def get_items(L):
    for i in range(len(L)):
        L[i] = input('Επόμενο αντικείμενο: ')
    return L

# Εκτύπωση λίστας
def print_list(L):
    for i in L:
        print(i)

main()
```

Εκχώρηση Τιμών σε Λίστα

Ο δεύτερος τρόπος είναι χρήσιμος αν δεν ξέρουμε εξ αρχής το πλήθος στοιχείων της λίστας:

```
# Συνάρτηση που δημιουργεί μια λίστα για ψώνια
def main():
    no_of_items = int(input('Πόσα αντικείμενα; '))
    L = create_list(no_of_items)
    L = get_items(L)
    print_list(L)

# Δημιουργία λίστας
def create_list(no_of_items):
    return list(range(no_of_items)) # Άλλος τρόπος

# Είσοδος αντικειμένων
def get_items(L):
    for i in range(len(L)):
        L[i] = input('Επόμενο αντικείμενο: ')
    return L

# Εκτύπωση λίστας
def print_list(L):
    for i in L:
        print(i)

main()
```

Πράξεις Πάνω σε Λίστες

- Παράθεση λιστών +:

λίστα = λίστα_1 + λίστα_2

```
>>> L_1 = [1, 2, 3]
>>> L_2 = [3, 4, 5]
>>> L = L_1 + L_2
>>> L
[1, 2, 3, 3, 4, 5]
```

```
>>> L_1 = L_1 + L_2
>>> L_1
[1, 2, 3, 3, 4, 5]
```



Δημιουργεί καινούργια λίστα
(με το ίδιο όνομα)

ή:

```
>>> L_1 += L_2
>>> L_1
[1, 2, 3, 3, 4, 5]
```



Η παράθεση γίνεται στην
υπάρχουσα λίστα

Το ίδιο ισχύει και για τον τελεστή επανάληψης *:

```
>>> L_1 = L_1 * 2 # καινούργια λίστα
>>> L_1 *= 2      # στην ίδια λίστα
```

Πράξεις Πάνω σε Λίστες

- Πρόσθεση στοιχείου με παράθεση:

λίστα = λίστα + [καινούργιο_στοιχείο]

```
>>> L = [1, 2, 3]
>>> L += [4]
>>> L
[1, 2, 3, 4]
```

Όχι:

```
>>> L += 4
Traceback (most recent call last):
  File "<pyshell#44>", line 1, in <module>
    L+=4
TypeError: 'int' object is not iterable
```

```
>>> L = [1, 2, 3]
>>> L += '456'
>>> L
[1, 2, 3, '4', '5', '6']
```

Πρέπει και τα δύο μέρη
να ακολουθούν το
πρωτόκολλο της επανάληψης

Πράξεις Πάνω σε Λίστες

Παράδειγμα:

```
# Συνάρτηση που δημιουργεί μια λίστα για ψώνια
def main():
    L = create_list()
    answer = 'Y'
    while answer == 'Y':
        new_item = get_item()
        L += [new_item]
        answer = input('Θες να συνεχίσεις; Y/N ')
    else:
        print_list(L)

# Δημιουργία λίστας
def create_list():
    return []

# Είσοδος αντικειμένων
def get_item():
    return input('Επόμενο αντικείμενο: ')

# Εκτύπωση λίστας
def print_list(L):
    for i in L:
        print(i)

main()
```


Πράξεις Πάνω σε Λίστες

Παράδειγμα:

```
# Συνάρτηση που δημιουργεί μια λίστα για ψώνια
def main():
    L = create_list()
    answer = 'Y'
    while answer == 'Y' or answer == 'y': # Το πρώτο είναι
        # λατινικός χαρακτήρας και το δεύτερο ελληνικός
        new_item = get_item()
        L += [new_item]
        answer = input('Θες να συνεχίσεις; Y/N ')
    else:
        print_list(L)

# Δημιουργία λίστας
def create_list():
    return []

# Είσοδος αντικειμένων
def get_item():
    return input('Επόμενο αντικείμενο: ')

# Εκτύπωση λίστας
def print_list(L):
    for i in L:
        print(i)

main()
```


Πράξεις Πάνω σε Λίστες

- Τεμαχισμός λίστας ::

λίστα[αρχή:τέλος]

Επιστρέφει **λίστα** με τα στοιχεία από τον δείκτη αρχή μέχρι τον τέλος-1:

```
      0 1 2 3 4
>>> L = [1, 2, 3, 4, 5]
>>> L[1:4]
[2, 3, 4]
```

λίστα[αρχή:τέλος:βήμα]

Επιστρέφει **λίστα** με τα στοιχεία από τον δείκτη αρχή μέχρι τον τέλος-1 που απέχουν βήμα-1 θέσεις μεταξύ τους:

```
>>> L[1:4:2]
[2, 4]
```

λίστα[αρχή:] **και** λίστα[:τέλος]

```
>>> L[1:]
[2, 3, 4, 5]
>>> L[:3]
[1, 2, 3]
```

Πράξεις Πάνω σε Λίστες

Ακραίες περιπτώσεις:

```
      0 1 2 3 4
>>> L = [1, 2, 3, 4, 5]
>>> L[1:-1]  ← len(L) - 1
[2, 3, 4]
```

```
>>> L[-1:]
[5]
```

```
>>> L[len(L)-1:0:-1] # Από len(L)-1 μέχρι και 0-βήμα
[5, 4, 3, 2]
```

```
>>> L[:] # Ολόκληρη τη λίστα
[1, 2, 3, 4, 5]
```

```
>>> L[::-1] # Αντίστροφη λίστα
[5, 4, 3, 2, 1]
```

```
>>> L[:len(L)+10] # Δείκτες εκτός ορίων
[1, 2, 3, 4, 5]
```

```
>>> L[2:1] # Δείκτες που δεν σχηματίζουν υπο-λίστα
[]
```

Πράξεις Πάνω σε Λίστες

- Αντικατάσταση τεμαχίων λίστας:

λίστα[αρχή:τέλος] = άλλη_λίστα

Αντικαθιστά το τεμάχιο της λίστας από τον δείκτη αρχή μέχρι τον τέλος-1 με την άλλη_λίστα:

0 1 2 3 4

```
>>> L = [1,2,3,4,5]
>>> L[1:3] = [6,7] # Αντικατάσταση
>>> L
[1, 6, 7, 4, 5]
```

```
>>> L = [1,2,3,4,5]
>>> L[1:1] = [6,7] # Παρεμβολή
>>> L
[1, 6, 7, 2, 3, 4, 5]
```

```
>>> L = [1,2,3,4,5]
>>> L[1:3] = [] # Διαγραφή
>>> L
[1, 4, 5]
```

```
>>> L = [1,2,3,4,5]
>>> L[1:2] = L[3:] # Πρώτα δημιουργείται η λίστα δεξιά
>>> L
[1, 4, 5, 3, 4, 5]
```

Ο Τελεστής `in`

στοιχείο `in` λίστα

True αν το στοιχείο
εμφανίζεται στη λίστα

False αν το στοιχείο **δεν**
εμφανίζεται στη λίστα

```
>>> L = [1,2,3,4,5]
>>> 1 in L
True
>>> 6 not in L # Το ίδιο με: not(6 in L)
True
```

Παράδειγμα:

```
codes = ['U135', 'X979', 'A123', 'S888']

item = input('Δώσε κωδικό προϊόντος: ')

if item in codes:
    print('To', item, 'βρέθηκε στη λίστα.')
else:
    print('To', item, 'δε βρέθηκε στη λίστα.')
```

Συναρτήσεις Πάνω σε Λίστες

Συνάρτηση	Λειτουργία	Επιστρέφει
<code>len()</code>	Μήκος λίστας	<code>int</code>
<code>min()/max()</code>	Ελάχιστο / μέγιστο στοιχείο λίστας (εφόσον τα στοιχεία είναι του ίδιου τύπου και μπορούν να συγκριθούν)	ανάλογα με τον τύπο των στοιχείων
<code>sum()</code>	Άθροισμα στοιχείων (εφόσον τα στοιχεία είναι αριθμητικού τύπου)	<code>int</code> ή <code>float</code>
<code>sorted()</code>	Ταξινόμηση στοιχείων (εφόσον τα στοιχεία είναι του ίδιου τύπου και μπορούν να συγκριθούν)	<code>list</code>
Τελεστής	Λειτουργία	Επιστρέφει
<code>del</code>	Διαγραφή στοιχείου από συγκεκριμένη θέση	<code>list</code>

```
>>> min([1,2,3,4,5])
1
>>> max(['αβγό', 'τυρί', 'γάλα'])
'τυρί'
>>> sum([1,2.0,3,4,5])
15.0
>>> sorted([7.2, 1, 4.5, 6, 3.2])
[1, 3.2, 4.5, 6, 7.2]
>>> sorted([7.2, 1, 4.5, 6, 3.2], reverse=True) #Αντίστροφη σειρά
[7.2, 6, 4.5, 3.2, 1]
>>> L = [1,2,3,4]
>>> del L[2]
>>> L
[1, 2, 4]
```

Συναρτήσεις Πάνω σε Λίστες

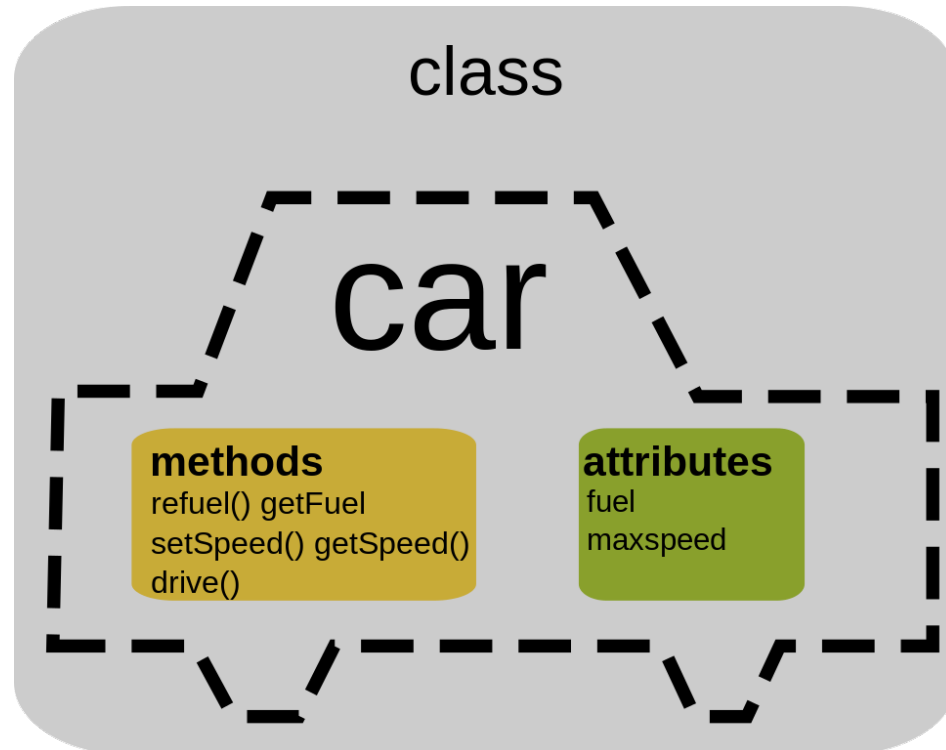
Λειτουργούν και πάνω σε άλλα αντικείμενα που **ακολουθούν το πρωτόκολλο επανάληψης**:

```
>>> len('αβγό'), len(range(100))
(4, 100)
>>> min('αβγό')
'α'
>>> sum(range(100))
4950
>>> sorted('Τυρί') # Επιστρέφει λίστα
['Τ', 'ί', 'ρ', 'υ']
>>> sorted(range(5), reverse = True)
[4, 3, 2, 1, 0]
>>> S = 'αβγό'
>>> S[2]
'γ'
>>> del S[2]
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    del S[2]
TypeError: 'str' object doesn't support item deletion
```

Τα string (και τα range) είναι immutable!

Μέθοδοι Λιστών

- Στην Python **όλα** είναι αντικείμενα (`int`, `float`, `str`, `list` κλπ.).
- Τα αντικείμενα «ανήκουν» σε μία κλάση.
- Στα αντικείμενα έχουμε τη λεγόμενη **ενθυλάκωση**: Περιέχουν τα δεδομένα τους (**ιδιοχαρακτηριστικά**) αλλά και συναρτήσεις (**μέθοδοι**) που δρουν πάνω σε αυτό



Αν δημιουργήσουμε ένα αντικείμενο:

```
fiat = Car()
```

Επικαλούμαστε τα ιδιοχαρακτηριστικά ή καλούμε τις μεθόδους ως εξής:

```
fiat.refuel()
```

```
speed = fiat.maxspeed
```


Μέθοδοι Λιστών

- Εύρεση θέσης-δείκτη στοιχείου: `index()`

`λίστα.index(στοιχείο_λίστας)`

```
>>> L = [1,2,3,4,5]
>>> L.index(3)
2
>>> L.index(6)
Traceback (most recent call last):
  File "<pyshell#47>", line 1, in <module>
    L.index(6)
ValueError: 6 is not in list
```

```
>>> # Εύρεση θέσης μεγαλύτερου στοιχείου
>>> L = [7.2, 1, 4.5, 6, 3.2]
>>> L.index(max(L))
0
```

- Πλήθος εμφανίσεων στοιχείου: `count()`

`λίστα.count(στοιχείο)`

```
>>> L = [1,2,3,3,4,5]
>>> L.count(3)
2
>>> L.count(6)
0
```


Μέθοδοι Λιστών

- Αντιστροφή στοιχείων λίστας: `reverse()`

```
>>> L = [1,2,3,4,5]
>>> L.reverse()
>>> L
[5, 4, 3, 2, 1]
```

- Ταξινόμηση στοιχείων λίστας: `sort()`

```
>>> L = [7.2, 1, 4.5, 6, 3.2]
>>> L.sort()
>>> L
[1, 3.2, 4.5, 6, 7.2]
```

Διαφορά `sorted()` και `sort()`:

```
>>> L = [7.2, 1, 4.5, 6, 3.2]
>>> id(L)
4393181696
>>> L.sort()
>>> id(L)
4393181696
>>> L = sorted(L)
>>> id(L)
4393182400
```

Δρα πάνω στο ίδιο το αντικείμενο

Δημιουργεί το αντικείμενο δεξιά του = και το εκχωρεί στην μεταβλητή αριστερά

Μέθοδοι Λιστών

Προσοχή: Αφού η `sort()` δεν δημιουργεί αντικείμενο είναι **λάθος** να εκχωρήσουμε το «αποτέλεσμά» της σε μεταβλητή.

```
>>> L = [7.2, 1, 4.5, 6, 3.2]
>>> L = L.sort()
>>> print(L)
None
>>> type(L)
<class 'NoneType'>
```

- Προσάρτηση στοιχείου στο τέλος: `append()`

λίστα.append(στοιχείο)

```
>>> L = [1, 2, 3]
>>> L.append(4)
>>> L
[1, 2, 3, 4]
>>> L = L.append(4)
>>> print(L)
None
```

Προσοχή:

```
>>> L = [1, 2, 3]
>>> L.append([4])
>>> L
[1, 2, 3, [4]]
```

(Οι λίστες μπορούν να έχουν για στοιχεία λίστες!)

Μέθοδοι Λιστών

Διαφορά `append()` και `+`:

```
>>> L = [1,2,3]
>>> id(L)
4393182656
>>> L.append(4)
>>> id(L)
4393182656
>>> L += [5] # Περισσότερα σε λίγο...
>>> id(L)
4393182656
>>> L = L + [6] # Αλλά:
>>> id(L)
4393225152
```

- Εισαγωγή στοιχείου **πριν** από κάποιο δείκτη: `insert()`

`λίστα.insert(δείκτης, στοιχείο)`

```
>>> L = [1,2,3]
>>> L.insert(0, 'μηδέν')
>>> L
['μηδέν', 1, 2, 3]
```

```
>>> L = [1,2,3]
>>> L.insert(-2, 'spam')
>>> L
[1, 'spam', 2, 3]
```

Μέθοδοι Λιστών

Ακραίες περιπτώσεις:

```
>>> L = [1,2,3]
>>> L.insert(0,'start') # Εισαγωγή στην αρχή
>>> L.insert(len(L)+10,'end') # Εισαγωγή στο τέλος
>>> L
['start', 1, 2, 3, 'end']
```

- Επέκταση λίστας με αντικείμενο που ακολουθεί το **πρωτόκολλο επανάληψης** : `extend()`

```
>>> L = [1,2,3]
>>> L.extend([4,5,6])
>>> L
[1, 2, 3, 4, 5, 6]
>>> L.extend(range(7,11))
>>> L
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> L.extend('spam')
>>> L
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 's', 'p', 'a', 'm']
>>> L.extend(54)
Traceback (most recent call last):
  File "<pyshell#116>", line 1, in <module>
    L.extend(54)
TypeError: 'int' object is not iterable
```

Ο τελεστής επαυξημένης εκχώρησης `+=` στην πραγματικότητα καλεί την `extend` (γι' αυτό και **δεν** δημιουργεί καινούργιο αντικείμενο).

Μέθοδοι Λιστών

- Διαγραφή όλων των στοιχείων : `clear()`

```
>>> L = [1,2,3]
>>> L.clear()
>>> L
[]
```

- Εξαγωγή στοιχείου από συγκεκριμένη **θέση**: `pop()`

Μεταβλητή = λίστα.`pop`(δείκτης)

```
>>> L = [1,2,3]
>>> x = L.pop(1)
>>> x, L
(2, [1, 3])
```

```
>>> L = [1,2,3]
>>> x = L.pop() # Χωρίς όρισμα
>>> x, L
(3, [1, 2])
```

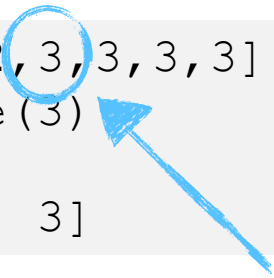
- Διαγραφή **στοιχείου**: `remove()`

λίστα.`remove`(στοιχείο)

```
>>> L = [1,2,3]
>>> L.remove(3) # Δεν χρειάζεται να ξέρουμε τη θέση του όπως με το del
>>> L
[1, 2]
```

Μέθοδοι Λιστών

```
>>> L = [1, 2, 3, 3, 3, 3]
>>> L.remove(3)
>>> L
[1, 2, 3, 3, 3]
```



Μόνο την πρώτη εμφάνιση!

Παράδειγμα: (αποθήκευση ως my_module.py)

```
# Συνάρτηση που διαγράφει όλες τις εμφανίσεις στοιχείου
def remove_all(List, element):
    while element in List:
        List.remove(element)
    return List
```

```
>>> import my_module as mm # Δίνω ένα πιο σύντομο όνομα, το mm
>>> L = [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
>>> mm.remove_all(L, 0)
[1, 1, 1, 1, 1]
```

Αντιγραφή

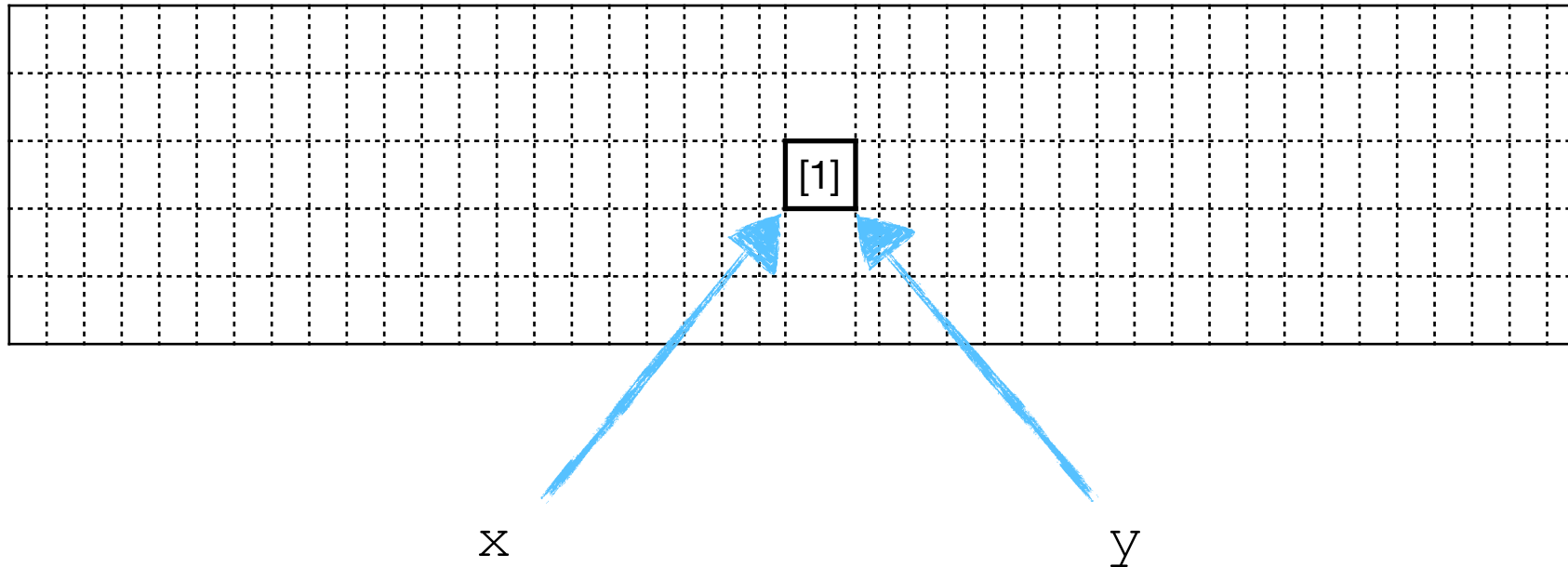
Η εκχώρηση ενός αντικειμένου σε ένα άλλο δεν δημιουργεί δύο αντικείμενα:

```
>>> x = 1
>>> id(x)
4300439792
>>> y = x
>>> id(y)
4300439792
```

```
>>> x = [1]
>>> id(x)
4349215680
>>> y = x
>>> id(y)
4349215680
```

απλά το αντικείμενο πλέον έχει δύο ονόματα (x και y).

RAM:



```
>>> x = [0]
>>> id(x)
4349216768
>>> id(y)
4349215680
```


Αντιγραφή

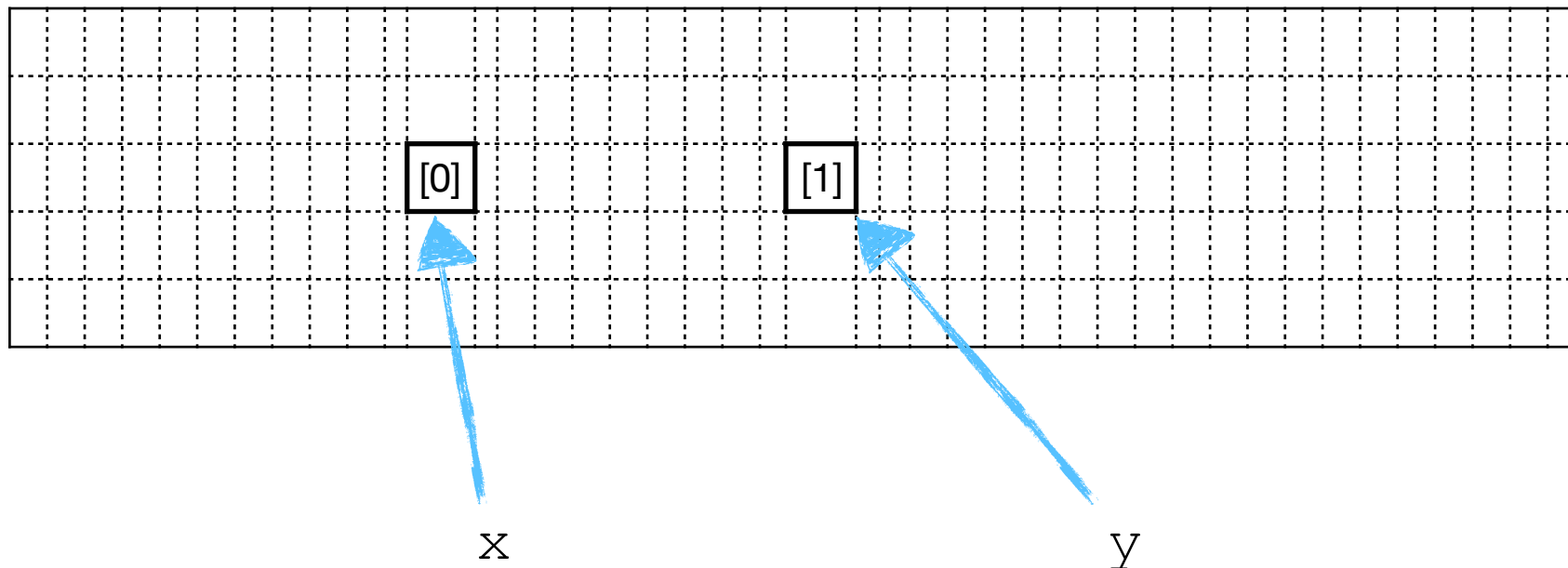
Η εκχώρηση ενός αντικειμένου σε ένα άλλο δεν δημιουργεί δύο αντικείμενα:

```
>>> x = 1
>>> id(x)
4300439792
>>> y = x
>>> id(y)
4300439792
```

```
>>> x = [1]
>>> id(x)
4349215680
>>> y = x
>>> id(y)
4349215680
```

απλά το αντικείμενο πλέον έχει δύο ονόματα (x και y).

RAM:



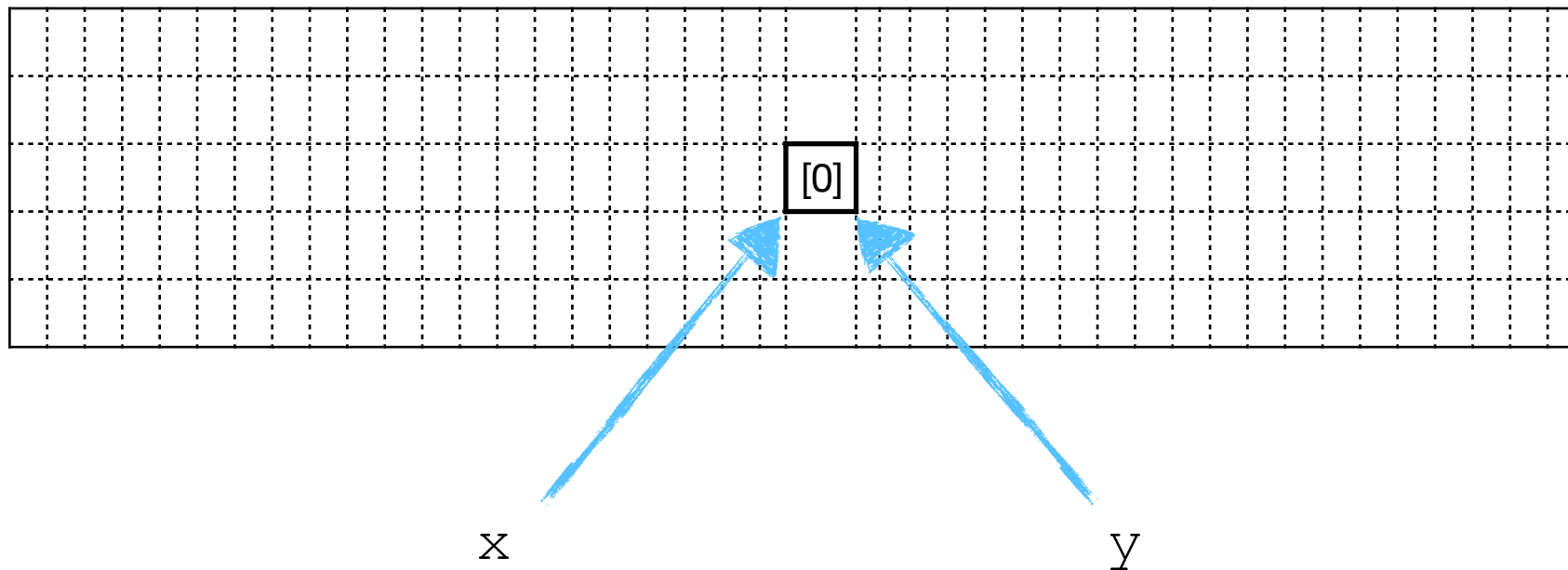
```
>>> x = [0] # Στην x εκχωρείτε καινούργια λίστα
>>> id(x)
4349216768
>>> id(y)
4349215680
```


Αντιγραφή

Όμως:

```
>>> x = [1]
>>> id(x)
4349215680
>>> y = x
>>> id(y)
4349215680
>>> x[0] = 0 # Η αλλαγή γίνεται πάνω στην ίδια λίστα (mutable)
>>> id(x)
4349215680
```

RAM:



Οι περισσότεροι τελεστές και συναρτήσεις κάνουν τις αλλαγές **πάνω στην ίδια** τη λίστα (δεν δημιουργούν καινούργια).

Υπάρχουν φορές που αυτό **δεν** είναι επιθυμητό και χρειαζόμαστε ένα αντίγραφο της λίστας.

Αντιγραφή

- Χειροκίνητα (append σε for):

```
List_2 = [] # Αρχικοποίηση λίστας  
for element in List_1:  
    List_2.append(element)
```

- Με επαυξημένη εκχώρηση σε κενή λίστα (+):

```
List_2 = [] # Αρχικοποίηση λίστας  
List_2 += List_1
```

- Με παράθεση σε κενή λίστα (+):

```
List_2 = [] + List_1
```

- Με τεμαχισμό:

```
List_2 = List_1[:]
```

- Με τη συνάρτηση list:

```
List_2 = list(List_1)
```

- Με τη μέθοδο copy:

```
List_2 = List_1.copy()
```

Αντιγραφή

Παράδειγμα: (my_module.py)

```
def get_username():  
    return input('Όνομα χρήστη: ')  
  
def get_password():  
    return input('Κωδικός: ')  
  
def register():  
    name = get_username()  
    password = get_password()  
    return [name, password]  
  
def change_password(user):  
    print('-- Αλλαγή κωδικού --')  
    temp_user = user.copy() # Αντιγραφή λίστας  
    temp_user[1] = get_password()  
    answer = input('Είσαι σίγουρος; (Y/N) ')  
    if answer == 'Y':  
        return temp_user  
    else:  
        return user
```

Παράδειγμα: (script.py)

```
import my_module  
user = my_module.register()  
user = my_module.change_password(user)  
print('Username:', user[0])  
print('Password:', user[1])
```

```
Όνομα χρήστη: dzoros  
Κωδικός: 1234  
-- Αλλαγή κωδικού --  
Κωδικός: 0000  
Είσαι σίγουρος; (Y/N) Y  
Username: dzoros  
Password: 0000
```

Αντιγραφή

Παράδειγμα: (my_module.py)

```
def get_username():  
    return input('Όνομα χρήστη: ')  
  
def get_password():  
    return input('Κωδικός: ')  
  
def register():  
    name = get_username()  
    password = get_password()  
    return [name, password]  
  
def change_password(user):  
    print('-- Αλλαγή κωδικού --')  
    temp_user = user.copy() # Αντιγραφή λίστας  
    temp_user[1] = get_password()  
    answer = input('Είσαι σίγουρος; (Y/N) ')  
    if answer == 'Y':  
        return temp_user  
    else:  
        return user
```

Παράδειγμα: (script.py)

```
import my_module  
user = my_module.register()  
user = my_module.change_password(user)  
print('Username:', user[0])  
print('Password:', user[1])
```

```
Όνομα χρήστη: dzoros  
Κωδικός: 1234  
-- Αλλαγή κωδικού --  
Κωδικός: 0000  
Είσαι σίγουρος; (Y/N) N  
Username: dzoros  
Password: 1234
```

Δισδιάστατες Λίστες

Λίστες με στοιχεία λίστες:

```
>>> L = [[1,2,3], [4,5,6], [7,8,9]]
```

Προσπέλαση «γραμμών» *:

```
>>> L[0]  
[1, 2, 3]  
>>> L[1]  
[4, 5, 6]  
>>> L[2]  
[7, 8, 9]
```

Προσπέλαση στοιχείου:

```
>>> L[0][2] # L[0] λίστα, άρα L[0][2] στοιχείο  
3
```

Προσπέλαση «στηλών»:

```
>>> L[0][1], L[1][1], L[2][1] #Τα στοιχεία της δεύτερης στήλης  
(2, 5, 8)
```

* Κάλλιστα μπορούμε να τις θεωρήσουμε και στήλες!

Δισδιάστατες Λίστες

Εκτύπωση των στοιχείων της σε μορφή πίνακα:

```
>>> L = [[1,2,3], [4,5,6], [7,8,9]]
>>> for i in range(len(L)):
        for j in range(len(L[i])): # L[i] λίστα
            print(L[i][j], end = '\t')
        print()
...
...
1   2   3
4   5   6
7   8   9
```

Δημιουργία και αρχικοποίηση με for: **(μονοδιάστατη λίστα)**

λίστα = [τιμή for i in iterable]

```
>>> L = [0 for i in range(3)]
>>> L = [0 for i in '123']
>>> L = [0 for i in [1,2,3]]
>>> L
[0, 0, 0]
```

Δισδιάστατη λίστα:

```
>>> L = [[0 for i in range(3)] for j in range(4)]
>>> L
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Δισδιάστατες Λίστες

Το πλήθος στοιχείων στις «γραμμές» και στις «στήλες» **δεν** χρειάζεται να είναι ίδιο!

```
>>> L = [[1,2], [1,2,3], [1,2,3,4]]
>>> for i in range(len(L)):
    for j in range(len(L[i])):
        print(L[i][j], end = '\t')
    print()
...
...
1    2
1    2    3
1    2    3    4
```

Μπορούμε να χειριστούμε τις λίστες με αντικείμενα που ακολουθούν το πρότυπο της επανάληψης ως δισδιάσταες λίστες:

```
>>> L = ['αβγό', 'τυρί', 'γάλα']
>>> L[1][2]
'ρ'
>>> L[0][1] = 'υ'
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    L[0][1] = 'υ'
TypeError: 'str' object does not support item assignment
```



Ta `str` είναι immutable!

Δισδιάστατες Λίστες

Παράδειγμα:

```
def create_results_list():
    return [[], [], [], []] # Ονόματα, Βαθμοί ασκήσεων, Βαθμοί εξέτασης,
                             # Τελικοί βαθμοί

def add_grade(List):
    print('----Επόμενος/η φοιτητής/τρια---')
    student_name = input('Όνομα φοιτητή/τριας: ')
    student_exercises_grade = float(input('Βαθμός ασκήσεων: '))
    student_exam_grade = float(input('Βαθμός εξέτασης: '))

    List[0].append(student_name)
    List[1].append(student_exercises_grade)
    List[2].append(student_exam_grade)
    List[3].append(min(10, round(student_exercises_grade + \
                                student_exam_grade)))

    return List

def print_final_results(List):
    print('-----')
    for i in range(len(List[0])):
        print('Ο/η', List[0][i], 'πήρε', List[3][i])
    print('\n\n', List, sep = '') # Εμφανίζουμε και τη λίστα

def main():
    List = create_results_list()
    number_of_exams = int(input('Πόσα γραπτά; '))
    for i in range(number_of_exams):
        List = add_grade(List)
    print_final_results(List)

main()
```


Δισδιάστατες Λίστες

Παράδειγμα:

```
Πόσα γραπτά; 3
----Επόμενος/η φοιτητής/τρια---
Όνομα φοιτητή/τριας: Μαρία
Βαθμός ασκήσεων: 2
Βαθμός εξέτασης: 7.5
----Επόμενος/η φοιτητής/τρια---
Όνομα φοιτητή/τριας: Ελένη
Βαθμός ασκήσεων: 1.5
Βαθμός εξέτασης: 2.1
----Επόμενος/η φοιτητής/τρια---
Όνομα φοιτητή/τριας: Γιάννης
Βαθμός ασκήσεων: 1.5
Βαθμός εξέτασης: 9.2
-----
Ο/η Μαρία πήρε 10
Ο/η Ελένη πήρε 4
Ο/η Γιάννης πήρε 10

[['Μαρία', 'Ελένη', 'Γιάννης'], [2.0, 1.5, 1.5], [7.5, 2.1, 9.2], [10, 4, 10]]
```

Ρηχά αντίγραφα

Η αντιγραφή παράγει «ρηχά» αντίγραφα:

```
>>> L = [[1,2,3], [4,5,6], [7,8,9]] # Λίστα με mutable στοιχεία
>>> M = L.copy()
>>> L[0][1] = 'spam'
>>> L
[[1, 'spam', 3], [4, 5, 6], [7, 8, 9]]
>>> M
[[1, 'spam', 3], [4, 5, 6], [7, 8, 9]]
>>> id(L) == id(M) # ή L is M
False
>>> id(L[0]) == id(M[0])
True
```

- Κάθε στοιχείο μιας λίστας είναι μία αναφορά προς αυτό το στοιχείο (οι τιμές των αντικειμένων είναι αναφορές).
- Κατά την αντιγραφή δημιουργείται μια λίστα και έπειτα αντιγράφονται σε αυτήν οι **αναφορές** των αντικειμένων που βρίσκονται στην πρωτότυπη λίστα

```
>>> import copy
>>> L = [[1,2,3], [4,5,6], [7,8,9]]
>>> M = copy.deepcopy(L) # Παράγει «βαθύ» αντίγραφο
>>> id(L) == id(M)
False
>>> id(L[0]) == id(M[0])
False
```

Ξεπακετάρισμα τιμών

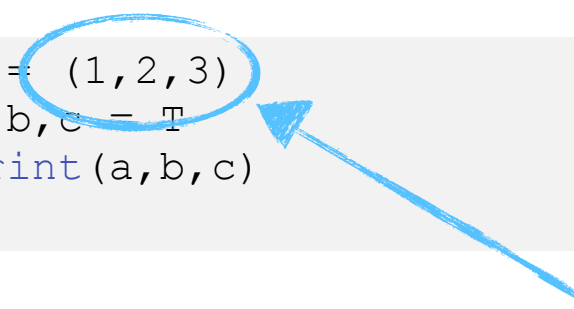
Στη Python αν έχουμε ένα αντικείμενο που ακολουθεί το πρωτόκολλο επανάληψης μπορούμε να «ξεπακετάρουμε» τα στοιχεία του ως εξής:

```
>>> L = [1,2,3]
>>> a,b,c = L
>>> print(a,b,c)
1 2 3
```

```
>>> R = range(1,4)
>>> a,b,c = R
>>> print(a,b,c)
1 2 3
```

```
>>> S = '123'
>>> a,b,c = S
>>> print(a,b,c)
1 2 3
```

```
>>> T = (1,2,3)
>>> a,b,c = T
>>> print(a,b,c)
1 2 3
```



Πλειάδα

Δημιουργία Πλειάδας

Πλειάδα είναι μια **διατεταγμένη** αλληλουχία αντικειμένων (όπως και οι λίστες):

- Κενή πλειάδα:

`πλειάδα = ()`

```
>>> T = ()
>>> type(T)
<class 'tuple'>
```

- Άμεση δημιουργία πλειάδας:

`πλειάδα = (στοιχείο_1, στοιχείο_2, ...)`

```
>>> T = (1, 2, 3)
>>> T
(1, 2, 3)
```

Προσοχή: δημιουργία πλειάδας με ένα στοιχείο:

```
>>> T = (1)
>>> type(T)
<class 'int'>
>>> T = (1,)
>>> type(T)
<class 'tuple'>
```

Δημιουργία Πλειάδας

- Με τον τελεστή επανάληψης *:

πλειάδα = (στοιχείο_1, στοιχείο_2, ...) * πλήθος επαναλήψεων

```
>>> T = (1,2,3) * 3
>>> T
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

- Χωρίς παρενθέσεις: (μόνο αν αυτό δεν προκαλεί σύγχυση)

πλειάδα = στοιχείο_1, στοιχείο_2, ...

```
>>> 1,2,3 # Πακετάρισμα τιμών
(1,2,3)
```

Παράδειγμα: (Τουμπάρισμα δύο τιμών στην Python)

```
>>> a = 1
>>> b = 2
>>> a,b = b,a # Πακετάρισμα και μετά ξεπακετάρισμα
>>> print(a,b)
2 1
```

Περισσότερα για πλειάδες

- Μπορούμε να μετατρέψουμε αντικείμενα που ακολουθούν το **πρωτόκολλο επανάληψης** σε πλειάδες με τη συνάρτηση `tuple`.
- Μπορούμε να προσπελάσουμε τα στοιχεία τους όπως και στις λίστες (`for`, δείκτες κλπ.).
- Μπορούμε να παραθέσουμε (+) σε αυτές κάποια άλλη πλειάδα (**μόνο** πλειάδα).
- Μπορούμε να τις τεμαχίσουμε (:).
- Μπορούμε να χρησιμοποιήσουμε τον τελεστή `in`.

Δεν μπορούμε να αλλάξουμε τα στοιχεία τους (είναι `immutable`)!

```
>>> T = (1, 2, 3)
>>> T[1] = 4
Traceback (most recent call last):
  File "<pyshell#67>", line 1, in <module>
    T[1] = 4
TypeError: 'tuple' object does not support item assignment
```

Συναρτήσεις Πάνω σε Πλειάδες

Συνάρτηση	Λειτουργία	Επιστρέφει
<code>len()</code>	Μήκος πλειάδας	<code>int</code>
<code>min()/max()</code>	Ελάχιστο / μέγιστο στοιχείο λίστας (εφόσον τα στοιχεία είναι του ίδιου τύπου και μπορούν να συγκριθούν)	ανάλογα με τον τύπο των στοιχείων
<code>sum()</code>	Άθροισμα στοιχείων (εφόσον τα στοιχεία είναι αριθμητικού τύπου)	<code>int</code> ή <code>float</code>
<code>sorted()</code>	Ταξινόμηση στοιχείων (εφόσον τα στοιχεία είναι του ίδιου τύπου και μπορούν να συγκριθούν)	<code>list</code>

```
>>> min(1,2,3,4,5) # Δεν χρειάζονται οι παρενθέσεις...
```

```
1
```

```
>>> T = 1,2,3,4,5
```

```
>>> min(T) # Εδώ χρειάζονται!
```

```
1
```

```
>>> sum((1,2,3,4,5)) # Όπως και εδώ!
```

```
15
```

```
>>> L = sorted((7.2, 6, 4.5, 3.2, 1)) # Και εδώ!
```

```
>>> tuple(L)
```

```
(1, 3.2, 4.5, 6, 7.2)
```

Επιστρέφει λίστα!

Μέθοδοι Πλειάδων

- Εύρεση θέσης-δείκτη στοιχείου: `index()`

`πλειάδα.index(στοιχείο_πλειάδας)`

```
>>> T = (1,2,3,4,5)
>>> T.index(3)
2
>>> T.index(6)
Traceback (most recent call last):
  File "<pyshell#87>", line 1, in <module>
    T.index(6)
ValueError: tuple.index(x): x not in tuple
```

- Πλήθος εμφανίσεων στοιχείου: `count()`

`πλειάδα.count(στοιχείο)`

```
>>> T = (1,2,3,3,4,5)
>>> T.count(3)
2
>>> T.count(6)
0
```

(Τροποποιητικές μέθοδοι προφανώς δεν έχουν νόημα...)

Γιατί πλειάδες;

- Μας θυμίζουν μαθηματικά αντικείμενα (διανύσματα κλπ.).
- Αποθηκεύονται σε ένα μπλοκ θέσεων μνήμης (σε αντίθεση με τις λίστες), πράγμα που τις κάνει πιο «γρήγορες».
- Χρειάζονται λιγότερη μνήμη από τις λίστες.
- Παρέχουν ασφάλεια δεδομένων.

Παράδειγμα:

```
>>> # Αποθήκευση τελικών βαθμών μαθήματος σε λίστα
>>> L = [['Μαρία', 'Ελένη', 'Γιάννης'], [10, 4, 10]]
>>> L[1][1] = 10
[['Μαρία', 'Ελένη', 'Γιάννης'], [10, 10, 10]]
```

```
>>> # Αποθήκευση τελικών βαθμών μαθήματος σε πλειάδα λιστών
>>> L = (['Μαρία', 'Ελένη', 'Γιάννης'], [10, 4, 10])
>>> L[1][1] = 10
(['Μαρία', 'Ελένη', 'Γιάννης'], [10, 10, 10])
```

```
>>> # Αποθήκευση τελικών βαθμών αποκλειστικά με πλειάδες
>>> L = (('Μαρία', 'Ελένη', 'Γιάννης'), (10, 4, 10))
>>> L[1][1] = 10
Traceback (most recent call last):
  File "<pyshell#97>", line 1, in <module>
    L[1][1] = 10
TypeError: 'tuple' object does not support item assignment
```

Συμβολοσειρές

- Προσπέλαση Χαρακτήρων
- Πράξεις με Συμβολοσειρές
- Τεμαχισμός Συμβολοσειράς
- Διαχωρισμός Συμβολοσειράς
- Μέθοδοι Συμβολοσειρών

Ανακεφαλαίωση

- Αλληλουχίες χαρακτήρων
- Διατάσσονται
- Αμετάβλητος τύπος (Immutable)
- Ακολουθούν το πρωτόκολλο επανάληψης

Δημιουργία συμβολοσειράς

'συμβολοσειρά'

"συμβολοσειρά"

'''συμβολοσειρά'''

"""συμβολοσειρά"""

```
>>> S = ' ' # Η κενή συμβολοσειρά
>>> menu = '''
Options
=====
1) Run
2) Quit
'''
>>> menu
'\nOptions\n=====\n1) Run\n2) Quit \n'
```

Ακολουθίες διαφυγής: \n, \t, \', \", \\

```
>>> str(1234.5) # Μετατροπή σε συμβολοσειρά
'1234.5'
```

Προσπέλαση Χαρακτήρων

- Με for:

```
>>> S = 'abcd'
>>> for ch in S:
        print(ch, end = ' ')
a b c d
```

- Με δείκτες:

```
>>> print(S[1], S[0], S[-1]) # Και αρνητικούς δείκτες
b a d
```

Η συνάρτηση `len()` επιστρέφει το πλήθος χαρακτήρων:

```
>>> len(S)
4
```

Παράδειγμα:

```
# Συνάρτηση που υπολογίζει το πλήθος εμφανίσεων χαρακτήρα
def main():
    text = input('Πληκτρολόγησε κείμενο: ')
    t = input('Δώσε χαρακτήρα: ')
    count = 0
    for ch in text:
        if ch==t:
            count += 1
    print('Ο χαρακτήρας', t, 'εμφανίζεται', count, 'φορές.')
```

Πράξεις με Συμβολοσειρές

- Παράθεση +:

συμβολοσειρά_1 + συμβολοσειρά_2

```
>>> 'abcd' + '1234'
'abcd1234'
>>> 'abcd' + 1234
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    'abcd' + 1234
TypeError: can only concatenate str (not "int") to str
```

```
>>> S = 'abcd'
>>> S += '1234'    # Επαυξημένη εκχώρηση
>>> S
'abcd1234'
```

Προσοχή: Δημιουργεί καινούργιο αντικείμενο (τα `str` είναι immutable)

```
>>> S[2] = 'e'
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    S[2] = 'e'
TypeError: 'str' object does not support item assignment
```

Πράξεις με Συμβολοσειρές

- Επανάληψη *:

συμβολοσειρά * αριθμός_επαναλήψεων

```
>>> 'abcd' * 3
'abcdabcdabcd'
>>> 'abcd' * '3'
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    'abcd' * '3'
TypeError: can't multiply sequence by non-int of type 'str'
```

- Σύγκριση συμβολοσειρών <, <=:

```
>>> 'aa' < 'ab'
True
>>> ord('a'), ord('b')
(97, 98)
>>> ord('A')
65
>>> 'a' < 'A'
False
>>> 'a' < 'aa'
True
```

Τεμαχισμός Συμβολοσειράς (:)

- Τεμαχισμός ::

συμβολοσειρά [αρχή : τέλος]

Επιστρέφει την **υπο-συμβολοσειρά** με στοιχεία από τον δείκτη αρχή μέχρι τον τέλος-1:

```
>>> S = 'Python'
>>> S[1:4]
'yth'
```

συμβολοσειρά [αρχή : τέλος : βήμα]

```
>>> S[1:4:2]
'yh'
```

λίστα [αρχή:] **και** λίστα [:τέλος]

```
>>> S[1:]
'ython'
>>> S[:3]
'Pyt'
```


Τεμαχισμός Συμβολοσειράς (:)

Ακραίες περιπτώσεις:

```
>>> S = 'Python'  
>>> S[1:-1]  
'ytho'
```

```
>>> S[-1:]  
'n'
```

```
>>> S[len(S):0:-1]  
'nohty'
```

```
>>> S[:]  
'Python'
```

```
>>> S[::-1]  
'nohtyP'
```

```
>>> S[:len(S)+10]  
'Python'
```

```
>>> S[2:1]  
''
```


Τεμαχισμός Συμβολοσειράς (:)

Παράδειγμα:

```
# Δημιουργία email χρήστη

def get_email(first, last, id_number):
    set_1 = first[0:3] # 3 πρώτα γράμματα μικρού ονόματος
    set_2 = last[0:3] # 3 πρώτα γράμματα επιθέτου
    set_3 = id_number[-3:] # 3 τελευταία ψηφία του Α.Μ.

    return set_1+set_2+set_3+'@math.uoa.gr'

def main():
    S_1 = input('Μικρό ονομα: ')
    S_2 = input('Επίθετο: ')
    S_3 = input('Αριθμός Μητρώου: ')

    email = get_email(last=S_2, first=S_1, id_number=S_3)
    print('Το email σου είναι: ', email.lower()) #Τα κάνει όλα μικρά

main()
```

```
Μικρό ονομα: Dimitris
Επίθετο: Zoros
Αριθμός Μητρώου: 1112202200014
Το email σου είναι: dimzor014@math.uoa.gr
```

Τεμαχισμός Συμβολοσειράς (:)

Παράδειγμα:

```
# Συνάρτηση που ελέγχει αν μια συμβολοσειρά είναι παλίνδρομη*  
  
def palindrome(word):  
    tmp = ''  
    for i in word: # «Διαγράφουμε» τα κενά  
        if i == ' ':  
            continue  
        else:  
            tmp += i  
  
    if tmp == tmp[::-1]: # Ελέγχουμε αν είναι παλίνδρομη  
        return True  
    else:  
        return False
```

```
>>> palindrome('aa ba a')  
True  
>>> palindrome('Dimitris')  
False
```

* Μια συμβολοσειρά είναι **παλίνδρομη** αν διαβάζεται το ίδιο και προς τα πίσω, π.χ. 'never odd or even'.

Άλλες Συναρτήσεις

Συναρτήσεις που λειτουργούν με αντικείμενα που ακολουθούν το πρωτόκολλο επανάληψης:

Συνάρτηση	Λειτουργία	Επιστρέφει
<code>len()</code>	Μήκος συμβολοσειράς	<code>int</code>
<code>min()/max()</code>	«Ελάχιστο» / «μέγιστο» στοιχείο συμβολοσειράς	<code>str</code>
<code>sorted()</code>	Ταξινόμηση χαρακτήρων συμβολοσειράς	<code>list</code>

```
>>> len('aa bb')
6
>>> min('αβγό')
'α'
>>> max('αβγό')
'ό'
>>> sorted('Τυρί') # Επιστρέφει λίστα
['Τ', 'ί', 'ρ', 'υ']
>>> L = sorted('Τυρί', reverse = True)
['υ', 'ρ', 'ί', 'Τ']
>>> str(L)
"['υ', 'ρ', 'ί', 'Τ']"
```

Έλεγχος Συμβολοσειράς (`in` και `not in`)

συμβολοσειρά_1 `in` συμβολοσειρά_2

True αν η συμβολοσειρά_1 εμφανίζεται στη συμβολοσειρά_2

False αν η συμβολοσειρά_1 **δεν** εμφανίζεται στη συμβολοσειρά_2

```
>>> text = 'abcdefg'
>>> 'd' in text
True
>>> 'h' not in text
True
```

```
>>> import random
>>> number = str(random.random()) # Οι float δεν είναι iterable...
>>> number
'0.8279943155075711'
>>> '6' in number
False
```

```
>>> DNA = '''
GTATACGTCTAGGCGTCACCCGCCATCTGTGTGCAGGTGGGCCGACGCACAAGCCAGTCTCGCAGCAAC
GCTCGTCAAGACACTGTCCCTGATTTCTCCGCTTCTAATAGCACACACGGGGCAATACCAGAAAGCTAC
'''
>>> 'CACAAGCCAGTCTCGCAGCAACGCTCGTCA' in DNA
True
```

Μέθοδοι Συμβολοσειρών

(Έλεγχος περιεχομένου συμβολοσειράς)

- Έλεγχος αν περιέχει μόνο **γράμματα ή αριθμούς**: `isalnum()`

```
>>> S = 'abdk34395dgkdhg8375'  
>>> S.isalnum()  
True  
>>> S = 'ad lklsdf 3354'  
>>> S.isalnum()  
False
```

- Έλεγχος αν περιέχει μόνο **γράμματα**: `isalpha()`

```
>>> S = 'abdk34395dgkdhg8375'  
>>> S.isalpha()  
False  
>>> S = 'ad lklsdf *$@!'  
>>> S.isalpha()  
False  
>>> S = 'adjskdjaa'  
>>> S.isalpha()  
True
```

- Έλεγχος αν περιέχει μόνο **αριθμούς**: `isdigit()`

```
>>> S = '0123456789'  
>>> S.isdigit()  
True
```

Μέθοδοι Συμβολοσειρών

(Έλεγχος περιεχομένου συμβολοσειράς)

- Έλεγχος αν περιέχει μόνο **κενά και** `\n` **ή** `\t`: `isspace()`

```
>>> S = ' \n \t '  
>>> S.isspace()  
True
```

- Έλεγχος αν περιέχει μόνο **πεζά** γράμματα: `islower()`

```
>>> S = 'Abcd'  
>>> S.islower()  
False
```

- Έλεγχος αν περιέχει μόνο **κεφαλαία** γράμματα: `isupper()`

```
>>> S = 'ABCD'  
>>> S.isupper()  
True
```

- Έλεγχος αν **ξεκινάει/τελειώνει** με συμβολοσειρά:

`startswith()` / `endswith()`

```
>>> S = 'ABCD'  
>>> S.startswith('AB') and S.endswith('CD')  
True
```

Μέθοδοι Συμβολοσειρών

(Έλεγχος περιεχομένου συμβολοσειράς)

Παράδειγμα:

```
while True:
    password = input('Δώσε νέο κωδικό: ')
    has_upper_letter = False
    has_lower_letter = False
    has_number = False

    if len(password) < 8:
        print('Πρέπει να έχει πάνω από 8 ψηφία.')
    elif not(password.isalnum()):
        print('Πρέπει να περιέχει μόνο γράμματα και αριθμούς.')
    elif not(password[0].isalpha()):
        print('Πρέπει να ξεκινάει με γράμμα.')
    else:
        for ch in password:
            if ch.isupper():
                has_upper_letter = True
            elif ch.islower():
                has_lower_letter = True
            elif ch.isdigit():
                has_number = True
        if has_upper_letter == False:
            print('Πρέπει να περιέχει τουλάχιστον ένα κεφαλαίο γράμμα.')
        elif has_lower_letter == False:
            print('Πρέπει να περιέχει τουλάχιστον ένα πεζό γράμμα.')
        elif has_number == False:
            print('Πρέπει να περιέχει τουλάχιστον έναν αριθμό.')
        else:
            print('Ο κωδικός έγινε δεκτός.')
            break
```


Μέθοδοι Συμβολοσειρών

(Έλεγχος περιεχομένου συμβολοσειράς)

- Πλήθος εμφανίσεων υπο-συμβολοσειράς: `count` (συμβολοσειρά)

```
>>> S = 'ουρουγουάη'  
>>> S.count('ου')  
3
```

- Πρώτη εμφάνιση υπο-συμβολοσειράς: `find` (συμβολοσειρά)

```
>>> S = 'ουρουγουάη'  
>>> S.find('ου')  
0  
>>> S[0]+S[1]  
'ου'
```

Αν η υπο-συμβολοσειρά **δεν** εμφανίζεται τότε επιστρέφει -1:

```
>>> S = 'ουρουγουάη'  
>>> S.find('δεν')  
-1
```

`in` και `find`:

```
>>> S = 'ουρουγουάη'  
>>> 'γου' in S  
True
```

```
>>> S = 'ουρουγουάη'  
>>> S.find('γου')  
5
```


«Τροποποιητικές» Μέθοδοι

(Επιστρέφουν τροποποιημένο αντίγραφο της συμβολοσειράς)

- Μετατροπή όλων των γραμμάτων **σε πεζά**: `lower()`

```
>>> S = 'LaTeX'
>>> S = S.lower()
>>> #Επιστρέφει κάτι (σε αντίθεση με τις τροποποιητικές των λιστών)
>>> S
'latex'
```

- Μετατροπή όλων των γραμμάτων **σε κεφαλαία**: `upper()`

```
>>> S = 'LaTeX'
>>> S.upper()
'LATEX'
```

Παράδειγμα:

```
while True:
    S = input('Θες να συνεχίσεις; ')
    if S.lower() != 'ναι': # Αν δεν πληκτρολογήσουμε ναι σταματάει
        print('Αντίο')
        break
```

```
Θες να συνεχίσεις; ΝΑΙ
Θες να συνεχίσεις; ΝΑΙ
Θες να συνεχίσεις; ναι
Θες να συνεχίσεις; ναι
Θες να συνεχίσεις; Μπα
Αντίο
```

«Τροποποιητικές» Μέθοδοι

(Επιστρέφουν τροποποιημένο αντίγραφο της συμβολοσειράς)

- Αφαίρεση **αρχικών** εμφανίσεων χαρακτήρα: `lstrip` (χαρακτήρας)

```
>>> S = 'aabcc'
>>> S.lstrip('a')
'bcc'
>>> S = '  a  '
>>> S.lstrip() # Αφαίρεση αρχικών κενών
'a  '
```

- Αφαίρεση **τελικών** εμφανίσεων χαρακτήρα: `rstrip` (χαρακτήρας)

```
>>> S = 'aabcc'
>>> S.rstrip('c')
'aab'
>>> S = '  a  '
>>> S.rstrip() # Αφαίρεση τελικών κενών
'  a'
```

- Αφαίρεση **αρχικών και τελικών** εμφανίσεων χαρακτήρα: `strip` (χαρακτήρας)

```
>>> S = 'aabaa'
>>> S.strip('a')
'b'
>>> S = '  a  '
>>> S.strip() # Αφαίρεση αρχικών και τελικών κενών
'a'
```

```
>>> S = 'aabaa'
>>> S.strip('c') # Δεν υπάρχει αλλαγή άρα όχι καινούργιο αντικείμενο!
'aabaa'
```

«Τροποποιητικές» Μέθοδοι

(Επιστρέφουν τροποποιημένο αντίγραφο της συμβολοσειράς)

Οι `lstrip`, `rstrip`, `strip` λειτουργούν και με ολόκληρες συμβολοσειρές:

```
>>> S = 'abcd123cdbaabbcb'
>>> S.strip('abc')
'd123cd'
```

Διαγράφει τους χαρακτήρες a, b, c από την αρχή και το τέλος

- Αντικατάσταση **όλων** των εμφανίσεων συμβολοσειράς με συμβολοσειρά:
`replace(συμβολοσειρά_1, συμβολοσειρά_2)`

```
>>> S = 'ουρουγούάη'
>>> S.replace('ou', 'OY')
'OYρOYγOYάη'
>>> S.replace('ou', '')
'ργάη'
>>> S.replace('OY', '')
'ουρουγούάη'
```

Παράδειγμα:

```
# Συνάρτηση που ελέγχει αν μια συμβολοσειρά είναι παλίνδρομη

def palindrome(word):
    word = word.replace(' ', '') # Διαγραφή κενών
    return word == word[::-1]
```

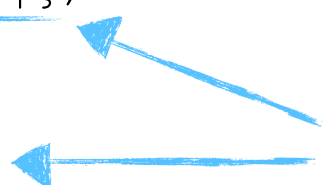
(sorted και sort ως προς συνάρτηση)

Μπορούν να διατάξουν μια λίστα με βάση την τιμή μιας συνάρτησης:

```
sorted(λίστα, key = όνομα_συνάρτησης)
```

```
λίστα.sort(key = όνομα_συνάρτησης)
```

Χωρίς παρενθέσεις!



```
>>> L = ['bab', 'aba', 'Abb', 'bAc']
>>> sorted(L)
['Abb', 'aba', 'bAc', 'bab']
>>> sorted(L, key = str.lower) # Αγνοούμε μικρά και κεφαλαία
['aba', 'Abb', 'bab', 'bAc']
```

```
>>> L # Η sorted δημιουργεί καινούργιο αντικείμενο
['bab', 'aba', 'Abb', 'bAc']
>>> L.sort(key = str.lower)
>>> L # Η sort κάνει τις αλλαγές πάνω στο ίδιο αντικείμενο
['aba', 'Abb', 'bab', 'bAc']
```

```
>>> L = ['Python', 'C', 'Java']
>>> L.sort(key = len) # Ταξινόμηση ως προς το μήκος
>>> L
['C', 'Java', 'Python']
```

Διαχωρισμός Συμβολοσειράς (split)

- Διαχωρισμός συμβολοσειράς στα κενά και επιστροφή **λίστας** με τις λέξεις που την απαρτίζουν : `split()`

```
>>> S = 'Των φρονίμων τα παιδιά πριν πεινάσουν παραγγέλνουν!'  
>>> S.split()  
['Των', 'φρονίμων', 'τα', 'παιδιά', 'πριν', 'πεινάσουν', 'παραγγέλνουν!']
```

Μπορούμε να επιλέξουμε εμείς σε ποια σύμβολα θα διαχωριστεί:

```
>>> S = 'a-b-c-d-e'  
>>> S.split('-')  
['a', 'b', 'c', 'd', 'e']
```

Μπορούμε να τη διαχωρίσουμε και ως προς κάποια συμβολοσειρά:

```
>>> S = 'άσπρο και μαύρο και κόκκινο και μπλε'  
>>> S.split(' και '  
['άσπρο', 'μαύρο', 'κόκκινο', 'μπλε']
```

Διαχωρισμός Συμβολοσειράς (split)

Παράδειγμα:

```
# Μετατροπή ημερομηνίας από Month/Day/Year σε Day/Month/Year

date_mdy = input('Δώσε ημερομηνία στη μορφή M/D/Y: ')
L = date_mdy.split('/')
date_dmy = L[1]+'/' + L[0]+'/' + L[2]

print('Η ημερομηνία στη μορφή D/M/Y είναι', date_dmy)
```

Δώσε ημερομηνία στη μορφή M/D/Y: **1/24/2022**
Η ημερομηνία στη μορφή D/M/Y είναι 24/1/2022

```
# Ως συνάρτηση

def date_converter(date_mdy):
    L = date_mdy.split('/')
    date_dmy = L[1]+'/' + L[0]+'/' + L[2]
    return date_dmy
```

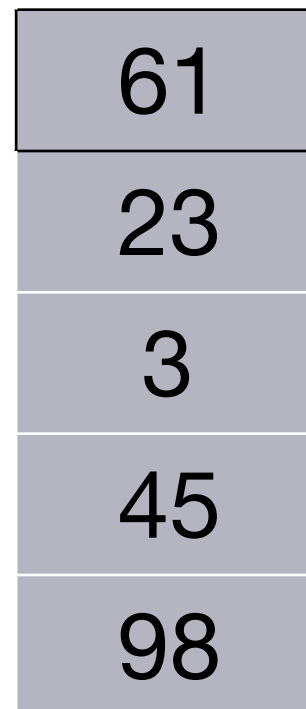
Δομές Δεδομένων

- Στοίβες (με βάση τις λίστες)
- Ουρές (με βάση τις λίστες)
- Ουρές προτεραιότητας (με βάση τις λίστες)
- Σύνολα
- Λεξικά

Στοίβες

- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Ακολουθούν τη **LIFO** λογική (Last In First Out)

Έχουμε πρόσβαση μόνο στην
κορυφή της στοίβας

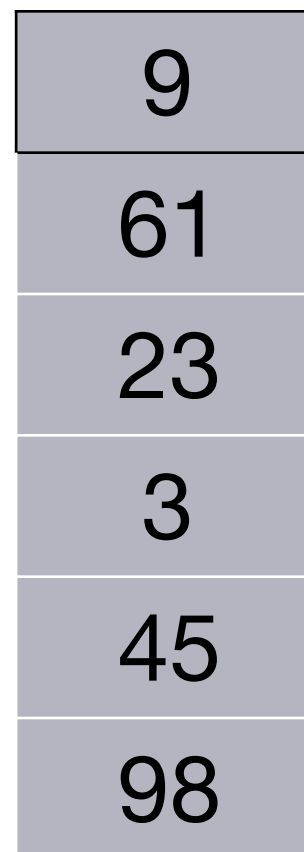


Προσθήκη στοιχείου (push)

Στοίβες

- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Ακολουθούν τη **LIFO** λογική (Last In First Out)

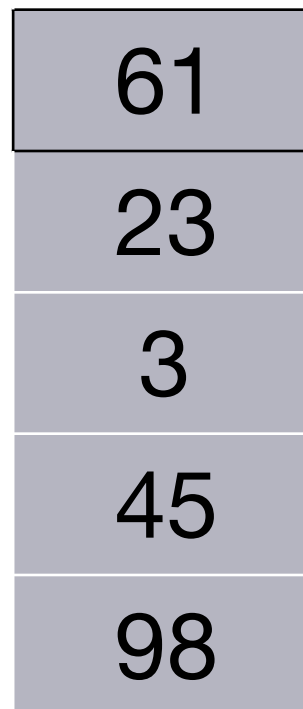
Έχουμε πρόσβαση μόνο στην
κορυφή της στοίβας



Αφαίρεση στοιχείου (pop)

Στοιίβες

- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Ακολουθούν τη **LIFO** λογική (Last In First Out)



Στοίβες

Μπορούμε να υλοποιήσουμε μία στοίβα χρησιμοποιώντας λίστες και τις μεθόδους που υποστηρίζουν:

- Αρχικοποίηση στοίβας:

```
>>> stack = []
```

- Εισαγωγή στοιχείου (push): `append()`

```
>>> stack.append(1)
>>> stack.append(2)
>>> stack.append(3)
>>> stack
[1, 2, 3]
```

- Εξαγωγή στοιχείου (pop): `pop()`

```
>>> x = stack.pop() #Χωρίς όρισμα εξάγει το τελευταίο στοιχείο
>>> stack
[1, 2]
```

Εφαρμογή

```
text = input('Δώσε την αριθμητική παράσταση:\n')

left = ['(', '[', '{']
right = [')', ']', '}']

balanced = True

stack = [] # Αρχικοποίηση στοίβας

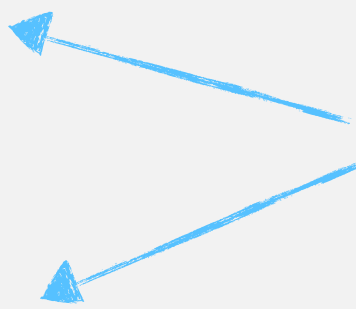
count_left, count_right = 0, 0

for ch in text: # Μέτρηση αριστερών και δεξιών παρενθέσεων
    if ch in left:
        count_left += 1
    elif ch in right:
        count_right += 1

if count_left != count_right:
    balanced = False
else:
    for ch in text: # Έλεγχος για την ορθότητα τοποθέτησης των παρενθέσεων
        if ch in left:
            stack.append(ch)
        elif ch in right and len(stack) == 0: # Δεξιά παρένθεση πριν από αριστερή
            balanced = False
            break
        elif ch in right and len(stack) > 0: # Το len(stack) > 0 δεν χρειάζεται
            tmp = stack.pop()
            i = left.index(tmp)
            if ch != right[i]: # Διαφορετικού τύπου δεξιά παρένθεση
                balanced = False
                break

if balanced:
    print('Σωστές παρενθέσεις')
else:
    print('Λάθος παρενθέσεις')
```

Δύο επαναλήψεις πάνω
στην συμβολοσειρά



Εφαρμογή

```
text = input('Δώσε την αριθμητική παράσταση:\n')

left = ['(', '[', '{']
right = [')', ']', '}']

balanced = True

stack = [] # Αρχικοποίηση στοίβας

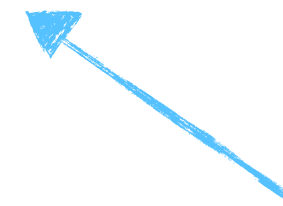
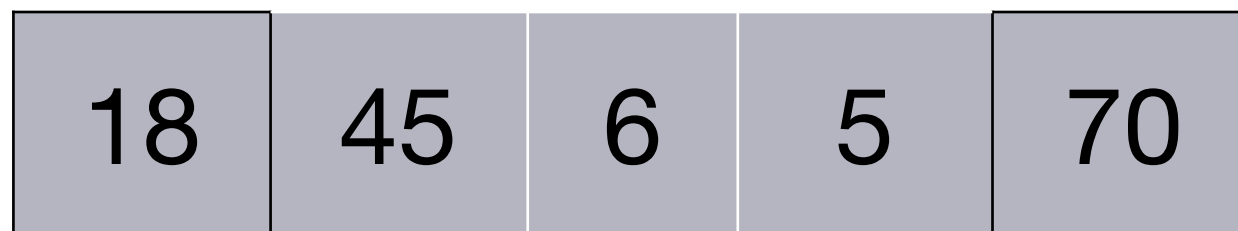
for ch in text:
    if ch in left:
        stack.append(ch)
    elif ch in right and len(stack)==0: # Δεξιά παρένθεση πριν από αριστερή
        balanced = False
        break
    elif ch in right and len(stack)>0: # Το len(stack)>0 δεν χρειάζεται
        tmp = stack.pop()
        i = left.index(tmp)
        if ch != right[i]: # Διαφορετικού τύπου δεξιά παρένθεση
            balanced = False
            break

if len(stack)>0: # Αριστερές παρενθέσεις πιο πολλές από τις δεξιές
    balanced = False

if balanced:
    print('Σωστές παρενθέσεις')
else:
    print('Λάθος παρενθέσεις')
```

Ουρές

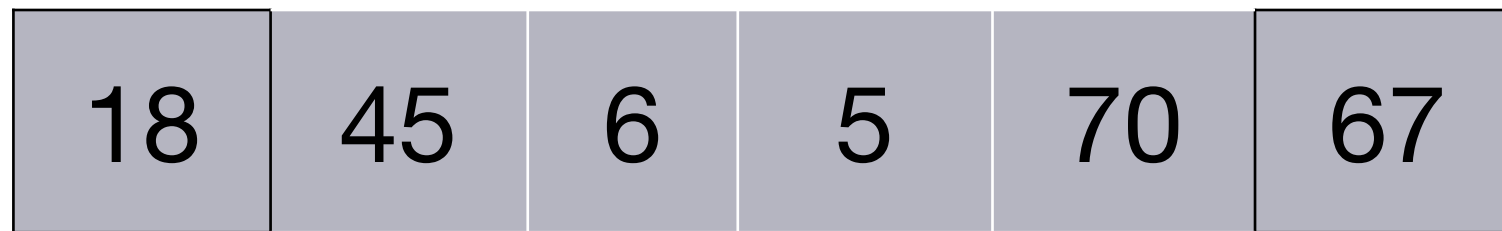
- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Ακολουθούν τη **FIFO** λογική (First In First Out)



Προσθήκη στοιχείου
στο τέλος της ουράς
(insert)

Ουρές

- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Ακολουθούν τη **FIFO** λογική (First In First Out)



Εξαγωγή στοιχείου
από την αρχή της ουράς
(eject)

Ουρές

- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Ακολουθούν τη **FIFO** λογική (First In First Out)



Ουρές

Μπορούμε να υλοποιήσουμε μία ουρά χρησιμοποιώντας λίστες και τις μεθόδους που υποστηρίζουν:

- Αρχικοποίηση ουράς:

```
>>> queue = []
```

- Εισαγωγή στοιχείου (insert): `append()`

```
>>> queue.append(1)
>>> queue.append(2)
>>> queue.append(3)
>>> queue
[1, 2, 3]
```

- Εξαγωγή στοιχείου (eject): `pop()`

```
>>> x = queue.pop(0) # Εξαγωγή στοιχείου με δείκτη 0
>>> queue
[2, 3]
```

Εφαρμογή

```
# Διατήρηση μιας ουράς πελατών

def new_appointment(queue):
    queue.append(input('Όνομα πελάτη: '))
    return queue

def next_appointment(queue):
    name = queue.pop(0)
    print('Επόμενος πελάτης: ', name)
    return queue

def new_queue():
    return []

# Χρήση συναρτήσεων

def main():
    queue = new_queue()
    for i in range(4):
        queue = new_appointment(queue)

    for i in range(len(queue)):
        queue = next_appointment(queue)

main()
```

```
Όνομα πελάτη: Γιάννης
Όνομα πελάτη: Μαρία
Όνομα πελάτη: Κώστας
Όνομα πελάτη: Ελένη
Επόμενος πελάτης: Γιάννης
Επόμενος πελάτης: Μαρία
Επόμενος πελάτης: Κώστας
Επόμενος πελάτης: Ελένη
```

Εφαρμογή

```
# Διατήρηση μιας ουράς πελατών
```

```
def new_appointment(queue):  
    queue.append(input('Όνομα πελάτη: '))
```

```
def next_appointment(queue):  
    name = queue.pop(0)  
    print('Επόμενος πελάτης: ', name)
```

```
def new_queue():  
    return []
```

```
# Χρήση συναρτήσεων
```

```
def main():  
    queue = new_queue()  
    for i in range(4):  
        new_appointment(queue)  
  
    for i in range(len(queue)):  
        next_appointment(queue)
```

```
main()
```

Σαν pass-by-reference
(mutable όρισμα)



Όνομα πελάτη: **Γιάννης**

Όνομα πελάτη: **Μαρία**

Όνομα πελάτη: **Κώστας**

Όνομα πελάτη: **Ελένη**

Επόμενος πελάτης: Γιάννης

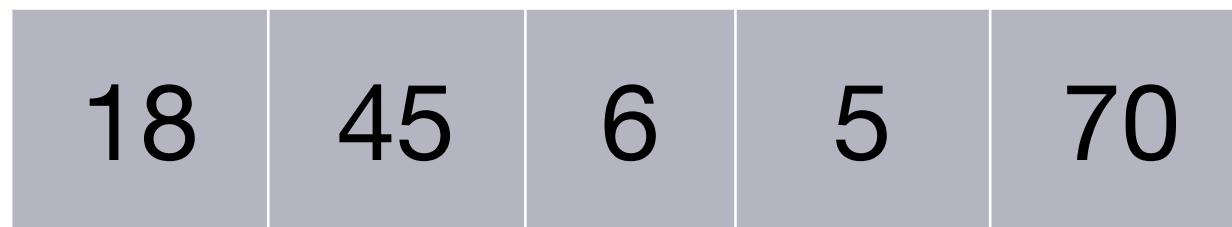
Επόμενος πελάτης: Μαρία

Επόμενος πελάτης: Κώστας

Επόμενος πελάτης: Ελένη

Ουρές Προτεραιότητας

- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Κάθε στοιχείο έχει και έναν **βαθμό προτεραιότητας** (όσο πιο μικρός είναι ο βαθμός τόσο πιο υψηλή είναι η προτεραιότητα του στοιχείου)
- Ακολουθούν τη **ΗΠΙΦΟ** λογική (Highest Priority In First Out)



Ουρές Προτεραιότητας

- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Κάθε στοιχείο έχει και έναν **βαθμό προτεραιότητας** (όσο πιο μικρός είναι ο βαθμός τόσο πιο υψηλή είναι η προτεραιότητα του στοιχείου)
- Ακολουθούν τη **HPIFO** λογική (Highest Priority In First Out)

3	1	2	1	5
18	45	6	5	70



Προσθήκη στοιχείου
στο τέλος της ουράς

Ουρές Προτεραιότητας

- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Κάθε στοιχείο έχει και έναν **βαθμό προτεραιότητας** (όσο πιο μικρός είναι ο βαθμός τόσο πιο υψηλή είναι η προτεραιότητα του στοιχείου)
- Ακολουθούν τη **ΗΡΙΦΟ** λογική (Highest Priority In First Out)

3	1	2	1	5	2
18	45	6	5	70	15

**Εξάγεται το στοιχείο με
την υψηλότερη προτεραιότητα**
(για τα στοιχεία με ίση προτεραιότητα έχουμε FIFO)

Ουρές Προτεραιότητας

- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Κάθε στοιχείο έχει και έναν **βαθμό προτεραιότητας** (όσο πιο μικρός είναι ο βαθμός τόσο πιο υψηλή είναι η προτεραιότητα του στοιχείου)
- Ακολουθούν τη **ΗΠΙΦΟ** λογική (Highest Priority In First Out)

3	1	2	1	5	2
18	45	6	5	70	15

**Εξάγεται το στοιχείο με
την υψηλότερη προτεραιότητα**
(για τα στοιχεία με ίση προτεραιότητα έχουμε FIFO)

Ουρές Προτεραιότητας

- Αποθηκεύουν δεδομένα του ίδιου τύπου
- Κάθε στοιχείο έχει και έναν **βαθμό προτεραιότητας** (όσο πιο μικρός είναι ο βαθμός τόσο πιο υψηλή είναι η προτεραιότητα του στοιχείου)
- Ακολουθούν τη **ΗΡΙΦΟ** λογική (Highest Priority In First Out)

3	2	1	5	2
18	6	5	70	15

**Εξάγεται το στοιχείο με
την υψηλότερη προτεραιότητα**
(για τα στοιχεία με ίση προτεραιότητα έχουμε FIFO)

Ουρές Προτεραιότητας

Χρησιμοποιούμε λίστες που περιέχουν πλειάδες (με δύο στοιχεία) ή και λίστες (αν θέλουμε να μπορούμε να αλλάξουμε τον βαθμό προτεραιότητας ενός) *:

- Αρχικοποίηση ουράς:

```
>>> priority_queue = []
```

- Εισαγωγή στοιχείου (insert): `append((βαθμός, στοιχείο))`

```
>>> priority_queue.append((4, 'Κώστας'))
>>> priority_queue.append((2, 'Ελένη'))
>>> priority_queue.append((3, 'Μαρία'))
>>> priority_queue
[(4, 'Κώστας'), (2, 'Ελένη'), (3, 'Μαρία')]
```

- Εξαγωγή στοιχείου (eject):

Μπορούμε να ταξινομήσουμε μία λίστα με στοιχεία πλειάδες (ή λίστες) χρησιμοποιώντας την μέθοδο `sort` (ή τη συνάρτηση `sorted`) :

```
>>> priority_queue.sort()
>>> priority_queue
[(2, 'Ελένη'), (3, 'Μαρία'), (4, 'Κώστας')]
```

* Όχι η καλύτερη υλοποίηση...



Ταξινόμηση ως προς
πρώτη συντεταγμένη

Ουρές Προτεραιότητας

Αν τα πρώτα στοιχεία είναι ίσα τότε η `sort` ταξινομεί τις πλειάδες με βάση το δεύτερο στοιχείο κ.ο.κ.:

```
>>> L = [ (2,2), (1,2), (2,1), (1,1) ]
>>> L.sort()
>>> L
[ (1, 1), (1, 2), (2, 1), (2, 2) ]
```

Εμείς θέλουμε να **μην** προχωράει η ταξινόμηση στο δεύτερο στοιχείο (γιατί θέλουμε στα στοιχεία με ίδια προτεραιότητα να έχουμε FIFO)!

```
>>> L = [ (2,2), (1,2), (2,1), (1,1) ]
>>> def first_element(T): # Επιστρέφει το πρώτο στοιχείο
...     return T[0]
...
>>> L.sort(key=first_element) # Ταξινόμηση ως προς συνάρτηση
>>> L
[ (1, 2), (1, 1), (2, 2), (2, 1) ]
```

Αν ταξινομήσουμε την ουρά με αυτόν τον τρόπο θα έχουμε ταξινόμηση **μόνο** ως προς τον βαθμό προτεραιότητας.

Συνεπώς το στοιχείο που πρέπει να εξαχθεί θα βρίσκεται στην **πρώτη θέση**.

Ουρές Προτεραιότητας

Παράδειγμα:

```
>>> priority_queue = [(3, 'Μαρία'), (2, 'Ελένη'), (3, 'Κώστας')]
>>> def first_element(T):
...     return T[0]
...
>>> priority_queue.sort(key=first_element)
>>> priority_queue
[(2, 'Ελένη'), (3, 'Μαρία'), (3, 'Κώστας')]
>>> x = priority_queue.pop(0)
>>> priority_queue
[(3, 'Μαρία'), (3, 'Κώστας')]
>>> x[1]
'Ελένη'
```

Επομένως για να κάνουμε εξαγωγή:

1. Ταξινομούμε την ουρά (με κλειδί τη συνάρτηση που επιστρέφει το πρώτο στοιχείο πλειάδας ή λίστας) .
2. Χρησιμοποιούμε τη μέθοδο `pop()` με δείκτη 0 .

Εφαρμογή

```
# Διατήρηση της ουράς στα επείγοντα περιστατικά
def first_element(T):
    return T[0]

def new_appointment(queue):
    name = input('Όνομα ασθενή: ')
    priority = int(input('Βαθμός προτεραιότητας: '))
    queue.append((priority, name))

def next_appointment(queue):
    queue.sort(key=first_element)
    patient = queue.pop(0)
    print('Επόμενος ασθενής: ', patient[1])

def new_queue():
    return []

# Χρήση συναρτήσεων
def main():
    queue = new_queue()
    for i in range(3):
        new_appointment(queue)
    for i in range(len(queue)):
        next_appointment(queue)

main()
```

Όνομα ασθενή: **Μαρία**
Βαθμός προτεραιότητας: **3**
Όνομα ασθενή: **Ελένη**
Βαθμός προτεραιότητας: **2**
Όνομα ασθενή: **Κώστας**
Βαθμός προτεραιότητας: **3**
Επόμενος ασθενής: Ελένη
Επόμενος ασθενής: Μαρία
Επόμενος ασθενής: Κώστας

Σύνολα

- Ξεχωριστός τύπος δεδομένων (`set`)
- Τα σύνολα είναι **mutable**
- Τα στοιχεία τους είναι **μοναδικά**
- **Δεν** υπάρχει διάταξη μεταξύ των στοιχείων
- Στοιχεία **μόνο** immutable αντικείμενα
- Δεδομένα ίδιου ή και διαφορετικού τύπου
- Ακολουθούν το **πρωτόκολλο επανάληψης**

Δημιουργία συνόλου

- Κενό σύνολο:

`σύνολο = set()`

```
>>> S = set()
>>> type(S)
<class 'set'>
>>> S = {} # Προσοχή: δημιουργεί λεξικό και όχι σύνολο
>>> type(S)
<class 'dict'>
```

Σύνολα

- Άμεση δημιουργία συνόλου:

σύνολο = {στοιχείο_1, στοιχείο_2, ...}

```
>>> S = {1,2,3}
>>> S
{1, 2, 3}
>>> S = {1,1,1,2,3} # Τα στοιχεία είναι μοναδικά!
>>> S
{1, 2, 3}
```

Μπορούμε να μετατρέψουμε αντικείμενα που ακολουθούν το **πρωτόκολλο επανάληψης** σε σύνολο με τη συνάρτηση `set()`:

```
>>> S = set([1,2,3]) # Λίστα
>>> S = set((1,2,3)) # Πλειάδα
>>> S = set(range(1,4)) # Αλληλουχία
>>> S
{1, 2, 3}
>>> S = set('123') # Συμβολοσειρά
>>> S
{'1', '3', '2'}
>>> S = set('aaabbbccc')
>>> S
{'a', 'b', 'c'}
```

Σύνολα

Τα στοιχεία του συνόλου μπορούν να είναι αντικείμενα διαφορετικού τύπου, αρκεί να είναι **immutable**:

```
>>> S = {1, (2, 3), 'τέσσερα'}
>>> S
{(2, 3), 1, 'τέσσερα'}
```

(Η αρχική «διάταξη» μπορεί να αλλάξει για λόγους αποτελεσματικότητας!)

```
>>> S = {1, [2, 3], 'τέσσερα'}
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    S = {1, [2, 3], 'τέσσερα'}
TypeError: unhashable type: 'list'
```

Παράδειγμα:

```
>>> # Δημιουργία συνόλου με τις λέξεις κειμένου
>>> text = 'Των φρονίμων τα παιδιά πριν πεινάσουν παραγγέλνουν'
>>> set(text.split())
{'παραγγέλνουν', 'παιδιά', 'Των', 'φρονίμων', 'πεινάσουν', 'τα', 'πριν'}
```


Προσπέλαση Στοιχείων

- Με `for`:

```
>>> S = {1,2,3}
>>> for i in S:
...     print(i, end = ' ')
...
1 2 3
```

(Προσπέλαση με δείκτες δεν θα είχε νόημα γιατί **δεν υπάρχει διάταξη!**)

- Η συνάρτηση `len` επιστρέφει τον πληθάριθμο του συνόλου:

```
>>> S = {1,2,3}
>>> len(S)
3
```

- `in` και `not in`:

```
>>> S = {1,2,3}
>>> 1 in S
True
>>> 4 not in S
True
```


Συναρτήσεις Πάνω σε Σύνολα

Συνάρτηση	Λειτουργία	Επιστρέφει
<code>len()</code>	Πληθάριθμος	<code>int</code>
<code>min()/max()</code>	Ελάχιστο / μέγιστο στοιχείο συνόλου (εφόσον τα στοιχεία είναι του ίδιου τύπου και μπορούν να συγκριθούν)	ανάλογα με τον τύπο των στοιχείων
<code>sum()</code>	Άθροισμα στοιχείων (εφόσον τα στοιχεία είναι αριθμητικού τύπου)	<code>int</code> ή <code>float</code>
<code>sorted()</code>	Ταξινόμηση στοιχείων (εφόσον τα στοιχεία είναι του ίδιου τύπου και μπορούν να συγκριθούν)	<code>list</code>

```
>>> min({1,2,3,4,5})
1
>>> max({'αβγό', 'τυρί', 'γάλα'})
'τυρί'
>>> sum({1,2.0,3,4,5})
15.0
>>> sorted({7.2, 1, 4.5, 6, 3.2}) # Επιστρέφει λίστα!
[1, 3.2, 4.5, 6, 7.2]
>>> sorted({7.2, 1, 4.5, 6, 3.2}, reverse = True)
[7.2, 6, 4.5, 3.2, 1]
```

(Στα σύνολα δεν υπάρχει μέθοδος `sort`!)

Μέθοδοι Συνόλων

(Προσθήκη Στοιχείων)

- Προσθήκη στοιχείου: `add()`

`σύνολο.add(στοιχείο)`

όπου το στοιχείο πρέπει να είναι **immutable**

```
>>> S = {1,2,3}
>>> S.add('τέσσερα')
>>> S
{1, 2, 3, 'τέσσερα'}
```

- Ενημέρωση συνόλου: `update()`

`σύνολο.update(αντικείμενο)`

όπου το αντικείμενο ακολουθεί το **πρωτόκολλο επανάληψης**.

```
>>> S = {1,2,3}
>>> S.update(range(4,7))
>>> S
{1, 2, 3, 4, 5, 6}
```

```
>>> S = {1,2,3}
>>> S.update('τέσσερα')
>>> S
{1, 2, 3, 'τ', 'σ', 'ρ', 'ε', 'έ', 'α'}
```

Μέθοδοι Συνόλων

(Διαγραφή Στοιχείων)

- Διαγραφή στοιχείου: `remove()` , `discard()`
`σύνολο.remove(στοιχείο)` / `σύνολο.discard(στοιχείο)`

```
>>> S = {1,2,3}
>>> S.remove(1)
>>> S
{2, 3}
```

```
>>> S = {1,2,3}
>>> S.discard(1)
>>> S
{2, 3}
```

Διαφορά `remove` και `discard`

```
>>> S = {1,2,3}
>>> S.discard(4) # Δεν κάνει τίποτα
>>> S.remove(4) # Προκαλεί εξαίρεση KeyError
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    S.remove(4)
KeyError: 4
```

- Εκκαθάριση συνόλου: `clear()`

```
>>> S = {1,2,3}
>>> S.clear()
>>> S
set()
```

Μέθοδοι Συνόλων

(Εξαγωγή Στοιχείων / Αντίγραφο Συνόλου)

- Εξαγωγή (αυθαίρετου) στοιχείου: `pop()`

```
>>> S = {1, 2, 3}
>>> S
{1, 2, 3}
>>> x = S.pop()
>>> x, S
(1, {2, 3})
```

(Στην πραγματικότητα εξάγει το «πρώτο» στοιχείο...)

- Δημιουργία αντίγραφου συνόλου: `copy()`

```
>>> S_1 = {1, 2, 3}
>>> S_2 = S_1.copy()
>>> S_1 == S_2 # Έλεγχος για ισότητα αντικειμένων
True
>>> S_1 is S_2 # Έλεγχος για ταύτιση αντικειμένων
False
>>> id(S_1), id(S_2)
(4486945856, 4486945408)
```

```
>>> S_1 = {1, 2, 3}
>>> S_2 = set(S_1)
```

(Παράγονται «ρηχά» αντίγραφα αλλά αυτό δεν δημιουργεί πρόβλημα καθώς τα σύνολα περιέχουν μόνο immutable αντικείμενα ως στοιχεία...)

Σύγκριση Συνόλων

Μαθηματικά	Τελεστής	Μέθοδος
$\in$	<code>in</code>	
$=$	<code>==</code>	
$\neq$	<code>!=</code>	
$\subseteq$	<code><, <=</code>	<code>σύνολο.issubset(iterable)</code>
$\supseteq$	<code>>, >=</code>	<code>σύνολο.issuperset(iterable)</code>
Ξένα σύνολα		<code>σύνολο.isdisjoint(iterable)</code>

```
>>> S_1 = {1,2,3}
>>> S_2 = {1,2,3,4}
>>> S_1 <= S_2, S_1.issubset(S_2)
(True, True)
>>> S_2 >= S_1, S_2.issuperset(S_1)
(True, True)
>>> S_1.isdisjoint(S_2)
False
>>> S_3 = {4,5,6}
>>> S_1.isdisjoint(S_3)
True
```

Οι μέθοδοι `issubset`, `issuperset`, `isdisjoint` λειτουργούν και με όρισμα αντικείμενο που ακολουθεί το **πρωτόκολλο επανάληψης**:

```
>>> {1,2,3}.isdisjoint([4,5,6])
True
```

Μετατρέπεται πρώτα σε σύνολο

Συνολοθεωρητικές πράξεις

Μαθηματικά	Τελεστής (χωρίς ενημέρωση)	Μέθοδος (χωρίς ενημέρωση)	Τελεστής (με ενημέρωση)	Μέθοδος (με ενημέρωση)
Ένωση, $\cup$		union	=	update
Τομή, $\cap$	&	intersection	&=	intersection_update
Διαφορά, $\setminus$	-	difference	-=	difference_update
Συμμετροδιαφορά	^	symmetric_difference	^=	symmetric_difference_update

```
>>> S_1 = {1,2,3}
>>> S_2 = {1,2,3,4}
>>> S_1 | S_2 # Επιστρέφει σύνολο
{1, 2, 3, 4}
>>> S_1 &= S_2 # Δεν επιστρέφει σύνολο!
>>> S_1
{1, 2, 3}
>>> S_3 = {2,3,4}
>>> S_1.difference(S_3) # Επιστρέφει σύνολο
{1}
>>> S_1.symmetric_difference_update(S_3) # Δεν επιστρέφει σύνολο!
>>> S_1
{1, 4}
```

Οι μέθοδοι λειτουργούν και με iterable αντικείμενα:

```
>>> {1,2,3}.intersection([2,3,4])
{2, 3}
```

Μη Συμβατική Χρήση Συνόλων

```
# Φιλτράρισμα επαναλαμβανόμενων τιμών
def remove_doubles(iterable):
    return list(set(iterable))

# Εύρεση κοινών στοιχείων
def find_common(iterable_1, iterable_2):
    return list(set(iterable_1) & set(iterable_2))

# Εύρεση διαφορών
def find_differences(iterable_1, iterable_2):
    return list(set(iterable_1) ^ set(iterable_2))

# Έλεγχος ισότητας αγνοώντας διάταξη
def is_equal(iterable_1, iterable_2):
    return set(iterable_1) == set(iterable_2)
```

```
>>> remove_doubles([1,1,2,2,3,3])
[1, 2, 3]
>>> find_common('abcd', 'bcde')
['c', 'b', 'd']
>>> find_differences((1,2,3), [2,3,4]) # Ορίσματα διαφορετικού τύπου!
[1, 4]
>>> is_equal('abc', ('c','b','a'))
True
```


Λεξικά

- Ξεχωριστός τύπος δεδομένων (`dict`)
- Σύνολα που περιέχουν ζεύγη της μορφής `κλειδί : τιμή` (στην πραγματικότητα τα λεξικά είναι **απεικονίσεις**)
- Το κλειδί χρησιμοποιείται για την ανάκτηση της τιμής
- Κλειδιά **μόνο** immutable αντικείμενα (τιμές οτιδήποτε)
- Τα κλειδιά είναι **μοναδικά** (οι τιμές όχι απαραίτητα)
- **Δεν** υπάρχει διάταξη μεταξύ των στοιχείων
- Είναι **mutable** (ακόμα και οι **τιμές** των στοιχείων μπορούν να αλλάξουν)
- Ακολουθούν το **πρωτόκολλο επανάληψης**

Δημιουργία Λεξικού

- Κενό λεξικό:

`λεξικό = {}` ή `λεξικό = dict()`

```
>>> D = dict()  
>>> type(D)  
<class 'dict'>
```


Λεξικά

- Άμεση δημιουργία λεξικού:

Λεξικό = {κλειδί_1:τιμή_1, Κλειδί_2:τιμή_2, ...}

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> D
{'Γιάννης': 6933333333, 'Μαρία': 6977777777}
>>> D = {'Γιάννης':6933333333, 'Γιάννης':6977777777}
>>> D # Τα κλειδιά είναι μοναδικά
{'Γιάννης': 6977777777}
```

```
>>> D = {'Γιάννης':6933333333} # Κλειδιά μόνο immutable
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    D = {'Γιάννης':6933333333} # Κλειδιά μόνο immutable
TypeError: unhashable type: 'list'
```

```
>>> D = {'Γιάννης':[6933333333,6977777777]} # Τιμές οτιδήποτε
>>> D
{'Γιάννης': [6933333333, 6977777777]}
```

Τα κλειδιά (μεταξύ τους) και οι τιμές (μεταξύ τους) μπορούν να έχουν διαφορετικό τύπο:

```
>>> D = {'A':1, 1:'A'}
>>> D
{'A': 1, 1: 'A'}
```

Λεξικά

- Μπορούμε να μετατρέψουμε **λίστες** ή **πλειάδες** με στοιχεία **ζεύγη** «κλειδιών-τιμών» με τη συνάρτηση `dict()`:

```
>>> D = dict([[1, 'A'], [2, 'B']]) # Λίστα με λίστες
>>> D = dict((1, 'A'), (2, 'B')) # Πλειάδα με Πλειάδες
>>> D = dict([(1, 'A'), (2, 'B')]) # Λίστα με Πλειάδες
>>> D = dict([1, 'A'], [2, 'B']) # Πλειάδα με λίστες
>>> D = dict([1, 'A'], (2, 'B')) # Ανακατεμένα!
>>> D
{1: 'A', 2: 'B'}
```

- Αν **όλα** τα κλειδιά είναι συμβολοσειρές (που δεν ξεκινάνε με αριθμό και δεν περιέχουν τελείες, *, # κτλ.) μπορούμε και ως εξής:

```
>>> dict(A=1, B=2, C=3) # Προσοχή: Χωρίς ''
{'A': 1, 'B': 2, 'C': 3}
```

- Με τη μέθοδο `fromkeys` με κλειδιά από αντικείμενο που ακολουθεί το **πρωτόκολλο επανάληψης** και προκαθορισμένη τιμή

`dict.fromkeys(iterable, προκαθορισμένη_τιμή)`

```
>>> dict.fromkeys(['A', 'B'], 0)
{'A': 0, 'B': 0}
>>> dict.fromkeys('ABC') # Χωρίς προκαθορισμένη τιμή
{'A': None, 'B': None, 'C': None}
```

Προσπέλαση Στοιχείων

- Με κλειδί: `λεξικό[κλειδί]` (Επιστρέφει την τιμή)

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> D['Μαρία']
6977777777
>>> D['Στέφανο']
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    D['Στέφανος']
KeyError: 'Στέφανο'
```

- Με `for` (ως προς κλειδί) :

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> for key in D:
...     print(key, D[key])
...
Γιάννης 6933333333
Μαρία 6977777777
```

Μεταβολή Στοιχείων

- Προσθήκη στοιχείου: `λεξικό[καινούργιο_κλειδί] = τιμή`

```
>>> D = {}
>>> D['Γιάννης'] = 6933333333
>>> D['Μαρία'] = 6977777777
>>> D
{'Γιάννης': 6933333333, 'Μαρία': 6977777777}
```

(Τα «διατάσσει» με τη σειρά που προστέθηκαν...)

- Τροποποίηση στοιχείου: `λεξικό[υπάρχον_κλειδί] = καινούργια_τιμή`
όπου το κλειδί **υπάρχει** ήδη στο λεξικό.

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> D['Γιάννης'] = 6944444444
>>> D
{'Γιάννης': 6944444444, 'Μαρία': 6977777777}
```

- Διαγραφή στοιχείου: `del λεξικό[κλειδί]`

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> del D['Μαρία']
>>> D
{'Γιάννης': 6944444444}
>>> del D['Στέφανο']
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    del D['Στέφανο']
KeyError: 'Στέφανο'
```

Συναρτήσεις και Τελεστές

Συνάρτηση	Λειτουργία	Επιστρέφει
<code>len()</code>	Πλήθος στοιχείων	<code>int</code>
<code>min()/max()</code>	Ελάχιστο / μέγιστο κλειδί (εφόσον είναι του ίδιου τύπου και μπορούν να συγκριθούν)	ανάλογα με τον τύπο των κλειδιών
<code>sum()</code>	Άθροισμα κλειδιών (εφόσον τα στοιχεία είναι αριθμητικού τύπου)	<code>int</code> ή <code>float</code>
<code>sorted()</code>	Ταξινόμηση κλειδιών (εφόσον είναι του ίδιου τύπου και μπορούν να συγκριθούν)	<code>list</code>

- Έλεγχος ισότητας (**ανεξάρτητη** από τη διάταξη): `==`, `!=`

```
>>> D_1 = {1:2, 3:4}
>>> D_2 = {3:4, 1:2}
>>> D_1 == D_2
True
```

- Έλεγχος μέλους **κλειδιού**: `in`, `not in`

```
>>> D = {1:2, 3:4}
>>> 1 in D
True
>>> 2 not in D # Προσοχή: Δεν δουλεύει με τιμές!
True
```

Μέθοδοι Λεξικών

(Προβολές Λεξικών)

Χρησιμοποιούνται για την **προβολή** του περιεχομένου ενός λεξικού:

- Προβολή κλειδιών: `keys()`
- Προβολή τιμών: `values()`
- Προβολή και των δύο (ως πλειάδα): `items()`

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> k = D.keys()
>>> v = D.values()
>>> i = D.items()
>>> type(k) # Επιστρέφουν αντικείμενα ειδικού τύπου:
<class 'dict_keys'>
>>> type(v)
<class 'dict_values'>
>>> type(i)
<class 'dict_items'>
>>> k # Δεν μπορούμε να τα χειριστούμε άμεσα
dict_keys(['Γιάννης', 'Μαρία'])
>>> v
dict_values([6933333333, 6977777777])
>>> i # Επιστρέφει τα ζεύγη ως πλειάδες
dict_items([('Γιάννης', 6933333333), ('Μαρία', 6977777777)])
```

Μέθοδοι Λεξικών

(Προβολές Λεξικών)

Οι προβολές λεξικού **ακολουθούν** το πρωτόκολλο επανάληψης:

```
>>> for key in k:
...     print(key, end = ' ')
...
...
Γιάννης Μαρία
>>> for value in v:
...     print(value, end = ' ')
...
...
6933333333 6977777777
>>> for items in i:
...     print(items, end = ' ')
...
...
('Γιάννης', 6933333333) ('Μαρία', 6977777777)
```

```
>>> list(k) # Δημιουργία λίστας από προβολή
['Γιάννης', 'Μαρία']
>>> list(v)
[6933333333, 6977777777]
>>> list(i)
[('Γιάννης', 6933333333), ('Μαρία', 6977777777)]
```


Μέθοδοι Λεξικών

(Προβολές Λεξικών)

Παράδειγμα:

```
# Έλεγχος αν κάποια τιμή υπάρχει στο λεξικό

def find_value(dictionary, value):
    if value in dictionary.values(): # in δουλεύει σε προβολές
        print('Βρέθηκε')
    else:
        print('Δεν βρέθηκε')
```

```
>>> find_value({1:2, 3:4},2)
Βρέθηκε
```

Μπορούμε μέσα στο `for` να ξεπακετάρουμε αντικείμενα:

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> for key, value in D.items():
...     print(key, value, end = '\t')
...
...
Γιάννης 6933333333      Μαρία 6977777777
```


Μέθοδοι Λεξικών

(Προσπέλαση / Εξαγωγή Στοιχείων)

- Προσπέλαση στοιχείου από κλειδί: `get()`
`λεξικό.get(κλειδί)`

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> D.get('Μαρία')
6977777777
>>> D['Μαρία'] # Κάνουν την ίδια δουλειά
6977777777
>>> D.get('Στέφανο') # Όμως με την get δεν προκαλείται εξαίρεση
>>> D.get('Στέφανο', 'Δεν έχει') # Επιστρέφει Τιμή αν δεν βρεθεί το κλειδί
'Δεν έχει'
```

- Εξαγωγή στοιχείου από κλειδί: `pop()`
`λεξικό.pop(κλειδί)`

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> D.pop('Μαρία')
6977777777
>>> D
{'Γιάννης': 6933333333}
>>> D.pop('Στέφανο') # Ίδια συμπεριφορά με τη del (μόνο που επιστρέφει τιμή)
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    D.pop('Στέφανο')
KeyError: 'Στέφανο'
```

```
>>> D.pop() # Οποσδήποτε με όρισμα!
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    D.pop()
TypeError: pop expected at least 1 argument, got 0
```

Μέθοδοι Λεξικών

(Εξαγωγή Στοιχείων / Εκκαθάριση Λεξικού)

- Εξαγωγή (αυθαίρετου) ζεύγους κλειδί/τιμή (ως πλειάδα): `popitem()`

`λεξικό.popitem()`

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> D.popitem()
('Μαρία', 6977777777)
>>> D
{'Γιάννης': 6933333333}
>>> D.popitem('Γιάννης') # Οποσδήποτε χωρίς όρισμα!
Traceback (most recent call last):
  File "<pyshell#54>", line 1, in <module>
    D.popitem('Γιάννης')
TypeError: dict.popitem() takes no arguments (1 given)
```

(Στην πραγματικότητα εξάγει το «τελευταίο» στοιχείο...)

- Εκκαθάριση λεξικού: `clear()`

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> D.clear()
>>> D
{}
```

Μέθοδοι Λεξικών

(Ενημέρωση / Αντιγραφή Λεξικού)

- Ενημέρωση λεξικού: `update()`

`λεξικό_1.update(λεξικό_2)`

(ή οτιδήποτε μπορεί να μετατραπεί σε λεξικό)

```
>>> D = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> D.update({'Μαρία':6977777777, 'Στέφανο':6944444444, 'Γιάννης':'-'})
>>> D
{'Γιάννης':'-', 'Μαρία': 6977777777, 'Στέφανο': 6944444444}
```

- Δημιουργία αντίγραφου λεξικού: `copy()`

```
>>> D_1 = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> D_2 = D_1.copy()
>>> D_1 == D_2
True
>>> D_1 is D_2
False
```

```
>>> D_1 = {'Γιάννης':6933333333, 'Μαρία':6977777777}
>>> D_2 = dict(D_1)
```

Ρηχά αντίγραφα:

```
>>> D_1 = {'a':[1,2], 'b':[3,4]}
>>> D_2 = D_1.copy()
>>> D_1['b'][0] = 2
>>> D_2
{'a': [1, 2], 'b': [2, 4]}
```

Εφαρμογές

Πρόγραμμα για την καταχώρηση γενεθλίων

```
# Το μενού
def menu():
    print('\n      ΜΕΝΟΥ')
    print('1) Αναζήτηση γενεθλίων (με όνομα)')
    print('2) Αναζήτηση γενεθλίων (με ημέρα)')
    print('3) Προσθήκη')
    print('4) Αλλαγή')
    print('5) Διαγραφή')
    print('6) Τερματισμός\n')

# Η επιλογή του χρήστη
def get_choice():
    return input('Δώσε την επιλογή σου: ')

# Επιλογή 1: Αναζήτηση γενεθλίων (με όνομα)
def look_up_by_name(birthdays):
    name = input('Δώσε όνομα: ')
    print(birthdays.get(name, 'Δεν βρέθηκε'))

# Επιλογή 2: Αναζήτηση γενεθλίων (με ημέρα)
def look_up_by_date(birthdays):
    date = input('Δώσε ημερομηνία (HH/MM/YYYY): ')
    L = list(birthdays.items()) # Λίστα με πλειάδες κλειδί/τιμή
    for element in L:
        if element[1] == date:
            print(element[0])
            break
    else:
        print('Δεν βρέθηκε')
```

Εφαρμογές

```
# Επιλογή 3: Προσθήκη
def add(birthdays):
    name = input('Δώσε όνομα: ')
    date = input('Δώσε ημερομηνία (HH/MM/YYYY): ')
    if name not in birthdays:
        birthdays[name] = date
    else:
        print('To', name, 'υπάρχει ήδη')

# Επιλογή 4: Αλλαγή
def change(birthdays):
    name = input('Δώσε όνομα: ')
    if name in birthdays:
        date = input('Δώσε νέα ημερομηνία (HH/MM/YYYY): ')
        birthdays[name] = date
    else:
        print('To', name, 'δεν βρέθηκε')

# Επιλογή 5: Διαγραφή
def delete(birthdays):
    name = input('Δώσε όνομα: ')
    if name in birthdays:
        del birthdays[name]
    else:
        print('To', name, 'δεν βρέθηκε')

# Επιλογή 6: Τερματισμός
def quit():
    print('\nΤερματισμός...')
```

Εφαρμογές

```
# Λάθος Επιλογή
def wrong_answer():
    print('\nΣφάλμα: μη έγκυρη επιλογή!')

# Η κύρια συνάρτηση
def main():
    birthdays = {} # Δημιουργία κενού λεξικού
    choice = 0
    while choice != '6':
        menu()
        choice = get_choice()
        if choice == '1':
            look_up_by_name(birthdays)
        elif choice == '2':
            look_up_by_date(birthdays)
        elif choice == '3':
            add(birthdays)
        elif choice == '4':
            change(birthdays)
        elif choice == '5':
            delete(birthdays)
        elif choice == '6':
            quit()
        else:
            wrong_answer()

main()
```

Εφαρμογές

Πρόγραμμα για μοίρασμα της τράπουλας

```
import random

deck = {1: '1 Καρό', 2: '2 Καρό', 3: '3 Καρό', 4: '4 Καρό', 5: '5 Καρό',
        6: '6 Καρό', 7: '7 Καρό', 8: '8 Καρό', 9: '9 Καρό', 10: '10 Καρό',
        11: 'K Καρό', 12: 'Q Καρό', 13: 'J Καρό',

        14: '1 Κούπα', 15: '2 Κούπα', 16: '3 Κούπα', 17: '4 Κούπα', 18: '5 Κούπα',
        19: '6 Κούπα', 20: '7 Κούπα', 21: '8 Κούπα', 22: '9 Κούπα', 23: '10 Κούπα',
        24: 'K Κούπα', 25: 'Q Κούπα', 26: 'J Κούπα',

        27: '1 Σπαθί', 28: '2 Σπαθί', 29: '3 Σπαθί', 30: '4 Σπαθί', 31: '5 Σπαθί',
        32: '6 Σπαθί', 33: '7 Σπαθί', 34: '8 Σπαθί', 35: '9 Σπαθί', 36: '10 Σπαθί',
        37: 'K Σπαθί', 38: 'Q Σπαθί', 39: 'J Σπαθί',

        40: '1 Μπαστούνι', 41: '2 Μπαστούνι', 42: '3 Μπαστούνι', 43: '4 Μπαστούνι',
        44: '5 Μπαστούνι', 45: '6 Μπαστούνι', 46: '7 Μπαστούνι', 47: '8 Μπαστούνι',
        48: '9 Μπαστούνι', 49: '10 Μπαστούνι', 50: 'K Μπαστούνι', 51: 'Q Μπαστούνι',
        52: 'J Μπαστούνι'}

no_of_players = int(input('Δώσε πλήθος παικτών: '))
no_of_cards = int(input('Δώσε πλήθος χαρτιών που πρέπει να μοιραστούν στον κάθε παίκτη: '))

if no_of_players*no_of_cards > 52:
    print('Χρειάζεται και δεύτερη τράπουλα!')
else:
    for i in range(1, no_of_players+1):
        print('Παίκτης', i)
        input('Πάτα enter για συνέχεια')
        hand = set() # Κενό σύνολο
        for j in range(no_of_cards):
            card_key = random.choice(list(deck.keys())) # Επιστρέφει ένα τυχαίο κλειδί
            card = deck.pop(card_key) # Εξάγουμε το χαρτί από την τράπουλα
            hand.add(card) # Προσθέτουμε το χαρτί στο χέρι του παίκτη
        print(hand)
        input('Πάτα enter για απόκρυψη')
        print('\n'*49) # Από 50 γραμμές και πάνω το IDLE «συμπιέζει» τις γραμμές
```

(Η random.choice(λίστα) επιστρέφει ένα τυχαίο στοιχείο από τη λίστα)

Εφαρμογές

Βάση δεδομένων (φοιτητών)

1ος τρόπος **λίστα** (με λεξικά):

```
def new_student():
    name = input('Όνομα: ')
    id_no = int(input('Αριθμός Μητρώου: '))
    year = int(input('Έτος Σπουδών: '))
    return {'name':name, 'info':{'id_no':id_no, 'year':year}}

def print_db(student_db):
    for element in student_db:
        print(element['name'], 'ΑΜ:'+str(element['info']['id_no']), \
              'Έτος:'+str(element['info']['year']), sep = ', ')

def main():
    student_db = [] # Λίστα
    answer = 'Y'
    while answer == 'Y':
        student_db.append(new_student())
        answer = input('Θέλεις να συνεχίσεις; (Y/N) ')
    print_db(student_db)

main()
```


Εφαρμογές

Βάση δεδομένων (φοιτητών)

2ος τρόπος **λεξικά** (με λεξικά):

```
def new_student():
    name = input('Όνομα: ')
    id_no = int(input('Αριθμός Μητρώου: '))
    year = int(input('Έτος Σπουδών:'))
    return {name:{'id_no':id_no, 'year':year}}

def print_db(student_db):
    for key in student_db:
        print(key, 'AM: '+str(student_db[key]['id_no']), \
              'Έτος: '+str(student_db[key]['year']), sep = ', ')

def main():
    student_db = {} # Λεξικό
    answer = 'Y'
    while answer == 'Y':
        student_db.update(new_student())
        answer = input('Θέλεις να συνεχίσεις; (Y/N) ')
    print_db(student_db)

main()
```

Λίστες ή λεξικά;

Λίστες

- Διάταξη
- Πολλαπλές εμφανίσεις στοιχείου
- Άμεσες αριθμητικές πράξεις

Λεξικά

- Τυχαίες θέσεις
- Αναζήτηση με κλειδί (πιο γρήγορη)
- Λιγότερος χώρος για «αραιές» δομές

Παράδειγμα:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \end{pmatrix}$$

Λίστα, δεσμεύουμε 30 θέσεις

```
>>> M = [[0,0,0,0,0,1],  
          [0,0,0,0,0,0],  
          [0,2,0,0,0,0],  
          [0,0,0,0,0,0],  
          [0,0,0,3,0,0]]
```

Λεξικό, δεσμεύουμε 3 θέσεις

```
>>> M = {(1,6):1, (3,2):2,  
         (5,4):3}
```

Αναδρομή

- Αναδρομικές Συναρτήσεις
- Αναδρομική Επίλυση Προβλημάτων
- Πως Δουλεύει;
- Αναδρομή ή Επαναληπτικός Αλγόριθμος;

Αναδρομικές Συναρτήσεις

Μια συνάρτηση μπορεί να κληθεί από **οποιοδήποτε** κομμάτι του προγράμματος (π.χ. και από κάποια άλλη συνάρτηση). Μπορεί ακόμα να κληθεί και από την **ίδια** τη συνάρτηση:

```
def recursive_function():
    print('Είμαι αναδρομική...')
    recursive_function()
```

[illegible]

Αναδρομικές Συναρτήσεις

Για να σταματήσουν κάποτε οι **αναδρομικές κλήσεις** πρέπει να εισάγουμε ένα κριτήριο που θα καθορίζει τον τερματισμό τους:

```
def recursive_function(count):  
    if count > 0: # count == 0 συνθήκη τερματισμού  
        print('Είμαι αναδρομική...')  
        recursive_function(count-1)
```

```
>>> recursive_function(5)  
Είμαι αναδρομική...  
Είμαι αναδρομική...  
Είμαι αναδρομική...  
Είμαι αναδρομική...  
Είμαι αναδρομική...
```



Σε κάθε αναδρομική κλήση το
όρισμα της συνάρτησης
μειώνεται κατά ένα

Αναδρομική Επίλυση Προβλημάτων

Πολλές φορές ερχόμαστε αντιμέτωποι με προβλήματα που μπορούν να χωριστούν σε **μικρότερα υπο-προβλήματα** με εντελώς **όμοια** δομή:

Υπολογισμός παραγοντικού: $n! = n \cdot \underline{(n-1)!}$  Μικρότερο υπο-πρόβλημα!

$0! = 1$  Βασική περίπτωση
(το πρόβλημα μπορεί να επιλυθεί **άμεσα**)

Τέτοιου είδους προβλήματα λύνονται εύκολα με **αναδρομή**:

```
def factorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n-1)
```

Παραδείγματα

Παράδειγμα:

```
def recursive_function(count, data):  
    if count == 1:  
        print(data)  
    else:  
        recursive_function(count-1, data**2)
```

```
>>> recursive_function(count=5, data=2)  
65536  
>>> ((2**2)**2)**2**2  
65536
```

- ▶ **Αρχική κλήση:** count=5, data=2
- ▶ **1η αναδρομική κλήση:** count=4, data=4
- ▶ **2η αναδρομική κλήση:** count=3, data=16
- ▶ **3η αναδρομική κλήση:** count=2, data=256
- ▶ **4η αναδρομική κλήση:** count=1, data=65536



Βάθος αναδρομής

Βασική περίπτωση
(σταματάνε οι αναδρομικές κλήσεις)

Παραδείγματα

Πάντα μπορούμε να σχεδιάσουμε και έναν **επαναληπτικό** αλγόριθμο!

Παράδειγμα: (Επαναληπτικός αλγόριθμος)

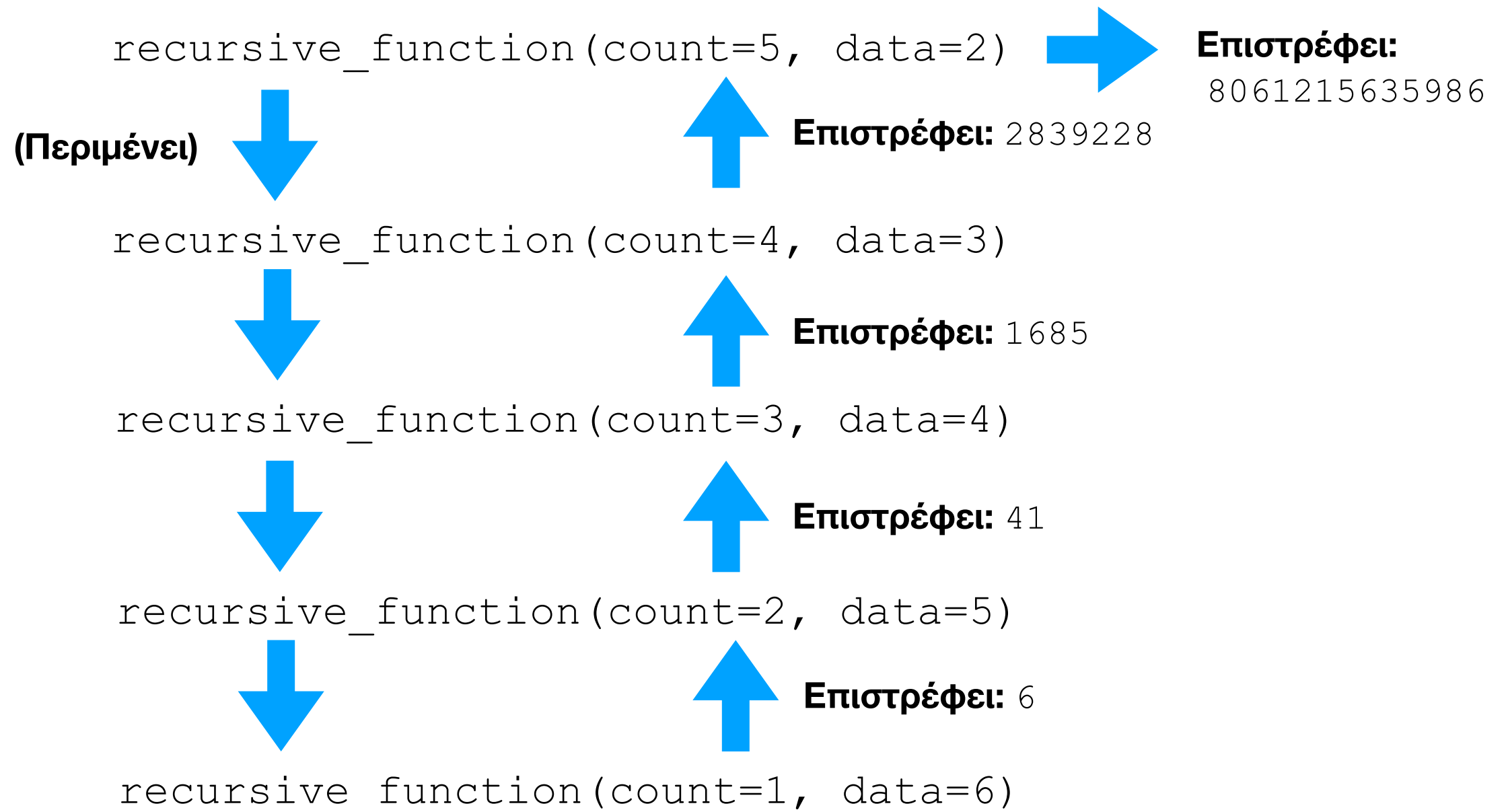
```
def iterative_function(count, data):  
    while count > 1:  
        data *= 2  
        count -= 1  
    print(data)
```

Παράδειγμα: (Αναδρομικός αλγόριθμος που επιστρέφει τιμή)

```
def recursive_function(count, data):  
    if count == 1:  
        result = data  
    else:  
        result = recursive_function(count-1, data+1)**2 + data  
    return result
```


Πως Δουλεύει;

```
def recursive_function(count, data):  
    if count == 1:  
        result = data  
    else:  
        result = recursive_function(count-1, data+1)**2 + data  
    return result
```



Πως Δουλεύει;

Για την αποθήκευση των δεδομένων των αναδρομικών κλήσεων χρησιμοποιείται μια **στοίβα**:

```
count=1, data=6: result=6
```

```
count=2, data=5 (result=?)
```

```
count=3, data=4 (result=?)
```

```
count=4, data=3 (result=?)
```

```
count=5, data=2 (result=?)
```

Πως Δουλεύει;

Για την αποθήκευση των δεδομένων των αναδρομικών κλήσεων χρησιμοποιείται μια **στοίβα**:

```
count=2, data=5: result=41
```

```
count=3, data=4 (result=?)
```

```
count=4, data=3 (result=?)
```

```
count=5, data=2 (result=?)
```

Πως Δουλεύει;

Για την αποθήκευση των δεδομένων των αναδρομικών κλήσεων χρησιμοποιείται μια **στοίβα**:

```
count=3, data=4: result=1685
```

```
count=4, data=3 (result=?)
```

```
count=5, data=2 (result=?)
```

Πως Δουλεύει;

Για την αποθήκευση των δεδομένων των αναδρομικών κλήσεων χρησιμοποιείται μια **στοίβα**:

```
count=4, data=3: result=2839228
```

```
count=5, data=2 (result=?)
```

Πως Δουλεύει;

Για την αποθήκευση των δεδομένων των αναδρομικών κλήσεων χρησιμοποιείται μια **στοίβα**:

```
count=5, data=2: result=8061215635986
```

Πως Δουλεύει;

Επαναληπτικός αλγόριθμος (με στοίβα):

```
def iterative_function(count, data):  
    # Αρχικοποιούμε μία «στοίβα»  
    stack = []  
  
    # Γεμίζουμε τη στοίβα με τα δεδομένα  
    while count > 1:  
        stack.append([count, data])  
        count -= 1  
        data += 1  
  
    # Υπολογισμός τιμής για την οριακή περίπτωση  
    result = data  
  
    # Υπολογισμός της τιμής για κάθε «αναδρομική κλήση»  
    while len(stack) > 0:  
        L = stack.pop() # Χρειαζόμαστε το data  
        result = result**2 + L[1]  
  
    return result
```


Αναδρομή ή Επαναληπτικός Αλγόριθμος;

Η ακολουθία Fibonacci:

$$F_0 = 1$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}, \quad n > 1$$

(Αποθήκευση σε Fibonacci.py)

```
# Αναδρομικός αλγόριθμος
def fib_1(n):
    if n in [0,1]:
        return 1
    else:
        return fib_1(n-1)+fib_1(n-2)
```

```
# Επαναληπτικός αλγόριθμος
def fib_2(n):
    if n in [0,1]:
        return 1
    else:
        F = [0] * (n+1) # Θα αποθηκεύσουμε τα F_i σε μία λίστα
        F[0] = 1
        F[1] = 1
        for i in range(2, n+1):
            F[i] = F[i-1] + F[i-2]
    return F[n]
```


Αναδρομή ή Επαναληπτικός Αλγόριθμος;

```
import Fibonacci
import time

print('Χρόνος αναδρομικού αλγόριθμου:')
start = time.time() # Πλήθος sec από την «αρχή του χρόνου»
x = Fibonacci.fib_1(40)
end = time.time()
print(format(end - start, '.6f'), 'sec')

print('Χρόνος επαναληπτικού αλγόριθμου:')
start = time.time()
x = Fibonacci.fib_2(40)
end = time.time()
print(format(end - start, '.6f'), 'sec')
```

```
Χρόνος αναδρομικού αλγόριθμου:
73.982229 sec
Χρόνος επαναληπτικού αλγόριθμου:
0.000034 sec
```

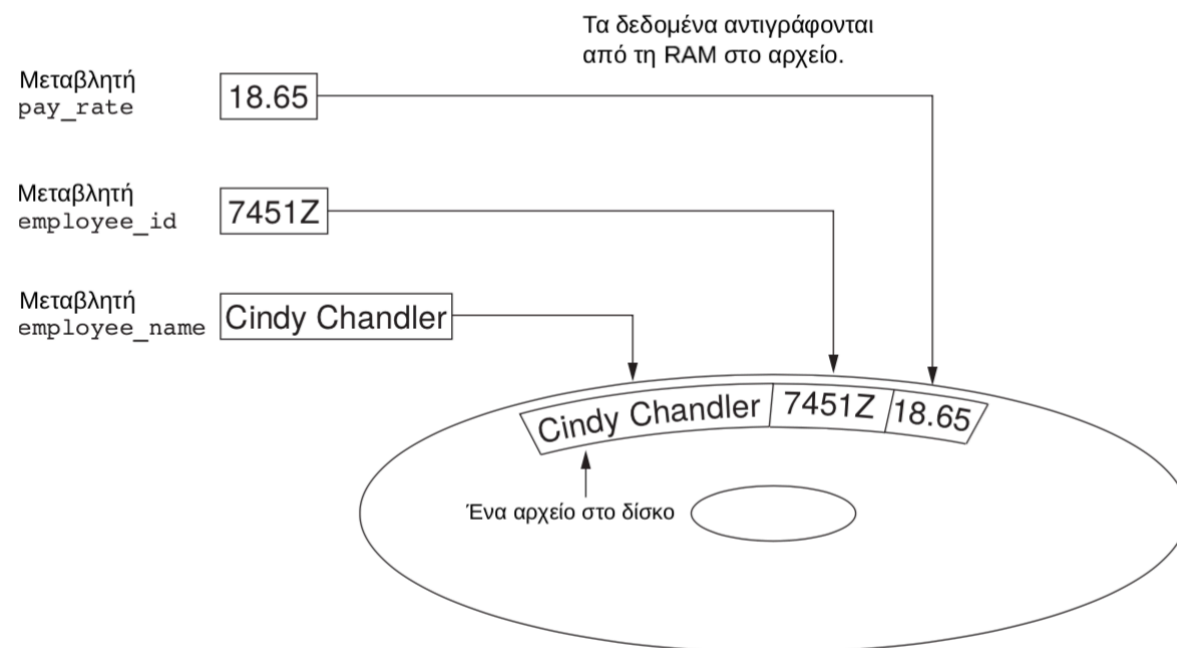
- Οι επαναληπτικοί αλγόριθμοι συνήθως είναι πιο **γρήγοροι**.
- Η αναδρομή κάνει πιο **εύκολη** τη σχεδίαση των Αλγορίθμων.

Αρχεία

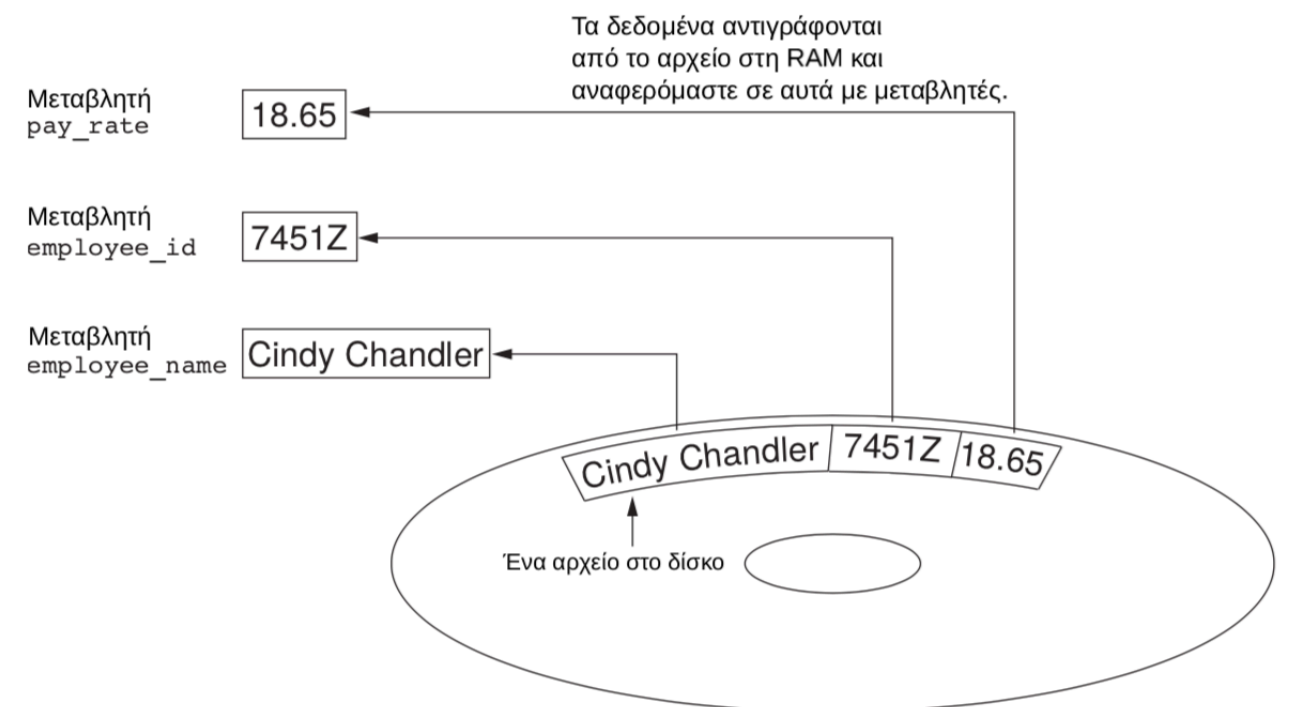
- Γιατί Αρχεία;
- Τύποι Αρχείων
- Δημιουργία και Άνοιγμα Αρχείου
- Μέθοδοι Αρχείων
- Αρχεία με Αριθμητικά Δεδομένα
- Παραδείγματα
- Αρχεία CSV
- Αποθήκευση Σύνθετων Δομών

Γιατί Αρχεία;

- Τα δεδομένα διατηρούνται και μετά την εκτέλεση του προγράμματος.
- Καταλληλότερα για την αποθήκευση μεγάλου όγκου δεδομένων.
- Επεξεργασία δεδομένων που έχουν δημιουργηθεί από άλλα προγράμματα.



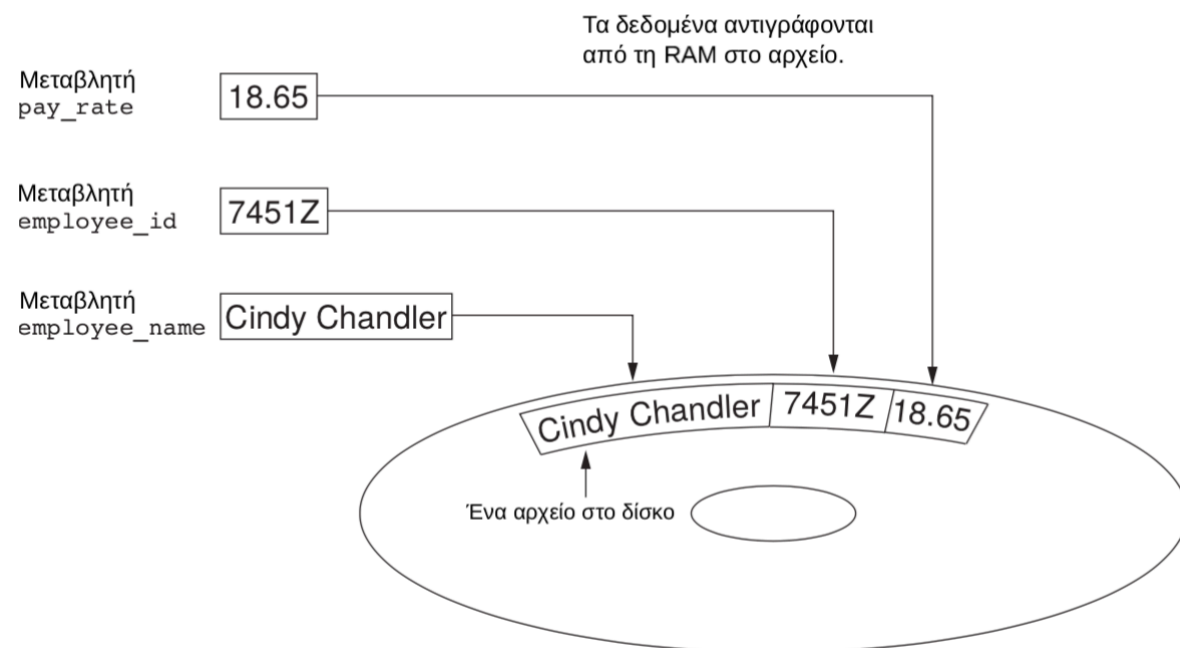
Αποθήκευση αρχείου στον δίσκο
(«Έξοδος»)



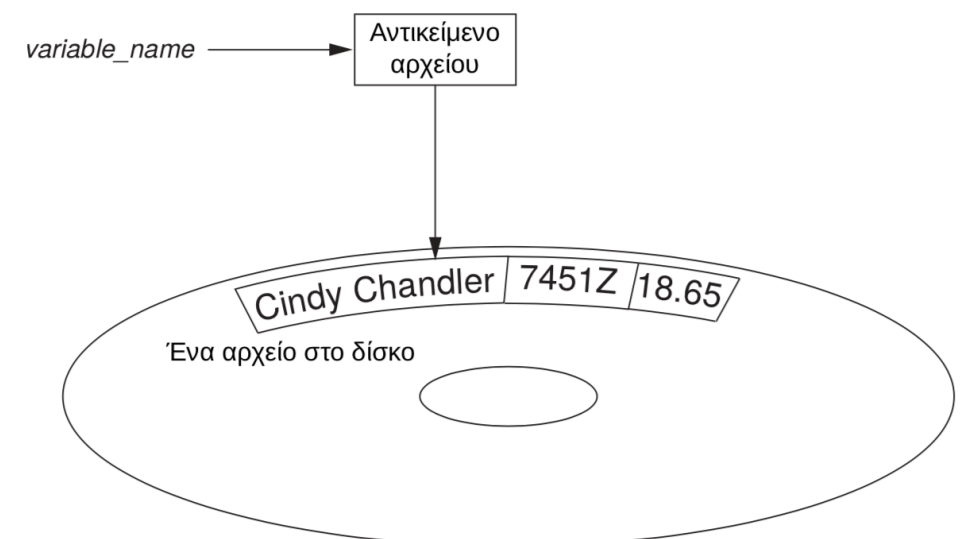
Φόρτωση αρχείου από τον δίσκο
(«Είσοδος»)

Γιατί Αρχεία;

- Τα δεδομένα διατηρούνται και μετά την εκτέλεση του προγράμματος.
- Καταλληλότερα για την αποθήκευση μεγάλου όγκου δεδομένων.
- Επεξεργασία δεδομένων που έχουν δημιουργηθεί από άλλα προγράμματα.



Αποθήκευση αρχείου στον δίσκο
(«Έξοδος»)



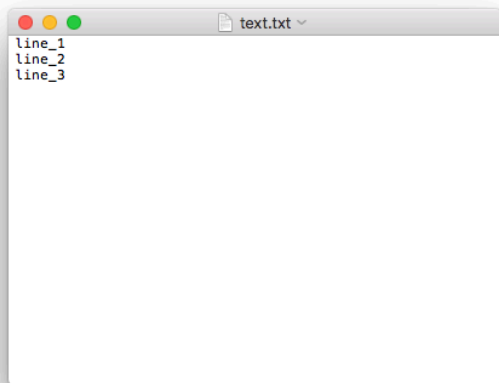
Φόρτωση αρχείου από τον δίσκο
(«Είσοδος»)

Η Python διαχειρίζεται τα αρχεία
ως **αντικείμενα**

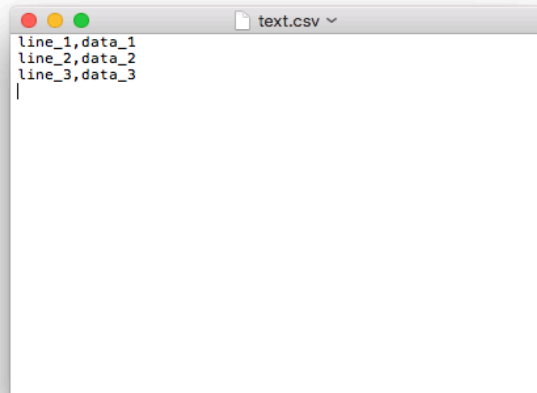
Τύποι Αρχείων

Αρχεία κειμένου

- Απλά αρχεία κειμένου (περιέχουν και ακολουθίες διαφυγής)
- Συνήθως κατάληξη `.txt`, `.csv`
- Άμεσως χειρισμός από την Python



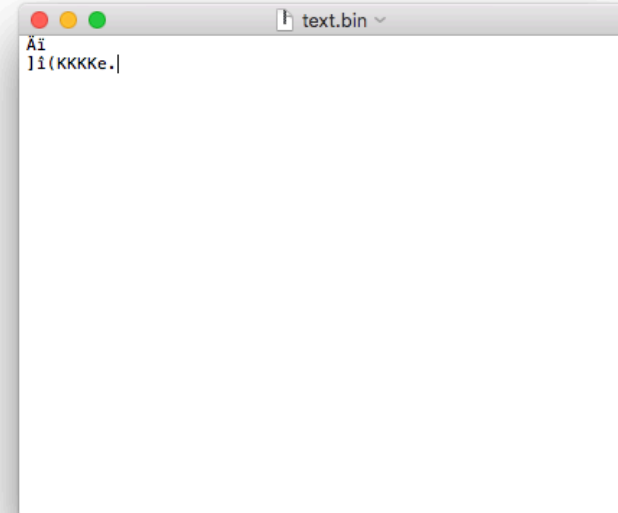
```
line_1
line_2
line_3
```



```
line_1,data_1
line_2,data_2
line_3,data_3
```

Δυαδικά αρχεία

- Οτιδήποτε (αρχεία ήχου, ψηφιακές φωτογραφίες κ.α.)
- Κατάλληλη `.bin`, `.dat`, `.mp3`, `.jpg` και άλλα
- Χρειάζονται ειδικά module για τον χειρισμό τους



```
A\x
11(KKKKe.|
```

Δημιουργία και Άνοιγμα Αρχείου

- Δημιουργούμε αντικείμενα αρχείου (file objects) με την εντολή `open`:

`μεταβλητή = open('όνομα_αρχείου', 'τύπος_προσπέλασης')`

όπου:

Προσπέλαση	Λειτουργία
<code>r</code>	read-only: Άνοιγμα μόνο για ανάγνωση.
<code>w</code>	write: Άνοιγμα για εγγραφή (το περιεχόμενο του αρχείου διαγράφεται). Αν το αρχείο δεν υπάρχει δημιουργείται αρχείο με όνομα το <code>όνομα_αρχείου</code> .
<code>a</code>	append-write: Άνοιγμα για εγγραφή (τα νέα δεδομένα προστίθενται στο τέλος του αρχείου). Αν το αρχείο δεν υπάρχει δημιουργείται αρχείο με όνομα το <code>όνομα_αρχείου</code> .

```
>>> f = open('text.txt', 'r')
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    f = open('text.txt', 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'text.txt'
```

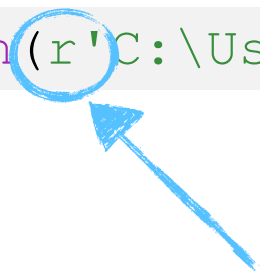
Δημιουργία και Άνοιγμα Αρχείου

- Αν το αρχείο βρίσκεται στον **ίδιο** υποκατάλογο με αυτόν του προγράμματος που το ανοίγει δεν χρειάζεται να προσθέσουμε τη διαδρομή του (path).
- Σε αντίθετη περίπτωση πρέπει να προσθέσουμε και τη διαδρομή (path):

```
>>> f = open('/Users/dzoros/Desktop/text.txt', 'r')
>>> type(f)
<class '_io.TextIOWrapper'>
```

ΠΡΟΣΟΧΗ: Σε περιβάλλον Microsoft Windows (όπου αντί για / χρησιμοποιείται \) θα πρέπει να γράψουμε τη διαδρομή του αρχείου ως εξής:

```
>>> f = open(r'C:\Users\dzoros\Desktop\text.txt', 'r')
```



Δηλώνει ότι η συμβολοσειρά δεν είναι επεξεργάσιμη, επομένως ο διερμηνέας ερμηνεύει το \ ως κανονικό χαρακτήρα (και όχι ακολουθία διαφυγής)

Μέθοδοι Αρχείων

(Ανάγνωση αρχείου)

ΠΡΟΣΟΧΗ: Πρέπει το αρχείο να έχει ανοιχθεί με τύπο προσπέλασης 'r' (αλλιώς θα δημιουργηθεί εξαίρεση)!

```
>>> f = open('polis.txt', 'r')
>>> print(f) # Δεν εμφανίζει το αρχείο!
<_io.TextIOWrapper name='polis.txt' mode='r' encoding='UTF-8'>
```

- Ανάγνωση ολόκληρου του αρχείου: `read()`

```
>>> f.read()
'Η πόλις του Κ. Καβάφη\n\nΕίπες· «Θα πάγω σ' άλλη γη, θα πάγω σ' άλλη
θάλασσα.\nΜια πόλις άλλη θα βρεθεί καλύτερη από αυτή.\nΚάθε
προσπάθεια μου μια καταδίκη είναι γραφή.\nκι είν' η καρδιά μου – σαν
νεκρός – θαμμένη.\nΟ νους μου ως τότε μες στον μαρασμόν αυτόν θα
μένει.\nΌπου το μάτι μου γυρίσω, όπου κι αν δω\nνερείπια μαύρα της
ζωής μου βλέπω εδώ,\nπου τόσα χρόνια πέρασα και ρήμαξα και
χάλασα.»\n\nΚαινούριους τόπους δεν θα βρεις, δεν θά βρεις άλλες
θάλασσες.\nΗ πόλις θα σε ακολουθεί. Στους δρόμους θα γυρνάς\nτους
ίδιους. Και στες γειτονιές τες ίδιες θα γερνάς.\nκαι μες στα ίδια
σπίτια αυτά θ' ασπρίζεις.\nΠάντα στην πόλι αυτή θα φθάνεις. Για τα
αλλού – μη ελπίζεις –\nδεν έχει πλοίο για σε, δεν έχει οδό.\nΈτσι που
τη ζωή σου ρήμαξες εδώ\nστην κόχη τούτη την μικρή, σ' όλην την γη την
χάλασες.'
```

Επιστρέφει το αρχείο ως **συμβολοσειρά** (στο τέλος κάθε γραμμής έχουμε την ακολουθία διαφυγής αλλαγής γραμμής `\n`).

Μέθοδοι Αρχείων

(Ανάγνωση αρχείου)

Θέση ανάγνωσης: Δείκτης που «δείχνει» σε ποιο σημείο βρισκόμαστε στην ανάγνωση του κειμένου.

```
>>> f.read()
'Η πόλις του Κ. Καβάφη\n\nΕίπες· «Θα πάγω σ' άλλη γη, θα πάγω σ' άλλη
θάλασσα.\nΜια πόλις άλλη θα βρεθεί καλύτερη από αυτή.\nΚάθε προσπάθεια μου
μια καταδίκη είναι γραφή·\nκι είν' η καρδιά μου – σαν νεκρός – θαμμένη.\nΟ
νους μου ως τότε μες στον μαρασμόν αυτόν θα μένει.\nΌπου το μάτι μου
γυρίσω, όπου κι αν δω\nνερείπια μαύρα της ζωής μου βλέπω εδώ,\nπου τόσα
χρόνια πέρασα και ρήμαξα και χάλασα.»\n\nΚαινούριους τόπους δεν θα βρεις,
δεν θά βρεις άλλες θάλασσες.\nΗ πόλις θα σε ακολουθεί. Στους δρόμους θα
γυρνάς\nτους ίδιους. Και στες γειτονιές τες ίδιες θα γερνάς·\nκαι μες στα
ίδια σπίτια αυτά θ' ασπρίζεις.\nΠάντα στην πόλι αυτή θα φθάνεις. Για τα
αλλού – μη ελπίζεις –\nδεν έχει πλοίο για σε, δεν έχει οδό.\nΈτσι που τη
ζωή σου ρήμαξες εδώ\nστην κόχη τούτη την μικρή, σ' όλην την γη την
χάλασες.'
```

```
>>> f.read() # Η θέση ανάγνωσης είναι στο τέλος του κειμένου
''
```

- Έυρεση θέσης ανάγνωσης: `tell()`

```
>>> f.tell()
1402
```

- Αλλαγή της θέσης ανάγνωσης: `seek(αριθμός_χαρακτήρα)`

```
>>> f.seek(0)
0
>>> S = f.read()
```

Μέθοδοι Αρχείων

(Ανάγνωση αρχείου)

```
>>> print(S)
```

Η πόλις του Κ. Καβάφη

Είπες· «Θα πάγω σ' άλλη γη, θα πάγω σ' άλλη θάλασσα.
Μια πόλις άλλη θα βρεθεί καλύτερη από αυτή.
Κάθε προσπάθεια μου μια καταδίκη είναι γραφτή·
κι είν' η καρδιά μου — σαν νεκρός — θαμμένη.
Ο νους μου ως πότε μες στον μαρασμόν αυτόν θα μένει.
Όπου το μάτι μου γυρίσω, όπου κι αν δω
ερείπια μαύρα της ζωής μου βλέπω εδώ,
που τόσα χρόνια πέρασα και ρήμαξα και χάλασα.»

Καινούριους τόπους δεν θα βρεις, δεν θά βρεις άλλες θάλασσες.
Η πόλις θα σε ακολουθεί. Στους δρόμους θα γυρνάς
τους ίδιους. Και στες γειτονιές τες ίδιες θα γερνάς·
και μες στα ίδια σπίτια αυτά θ' ασπρίζεις.
Πάντα στην πόλι αυτή θα φθάνεις. Για τα αλλού — μη ελπίζεις —
δεν έχει πλοίο για σε, δεν έχει οδό.
Έτσι που τη ζωή σου ρήμαξες εδώ
στην κόχη τούτη την μικρή, σ' όλην την γη την χάλασες.

Μέθοδοι Αρχείων

(Ανάγνωση αρχείου)

- Ανάγνωση γραμμής: `readline()`

```
>>> f.seek(0)
0
>>> f.readline()
'Η πόλις του Κ. Καβάφη\n'
>>> f.readline()
'\n'
>>> f.readline()
'Είπες· «Θα πάγω σ' άλλη γη, θα πάγω σ' άλλη θάλασσα.\n'
```

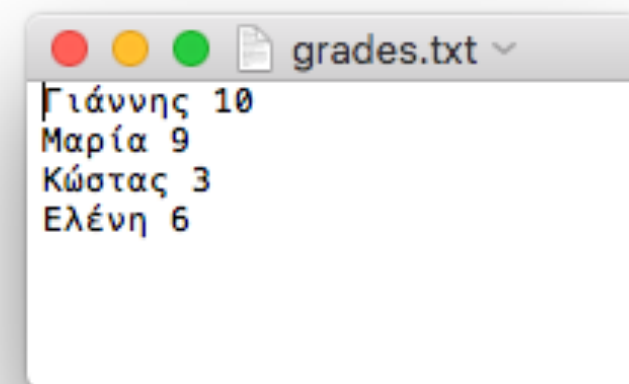
Επιστρέφει κάθε «γραμμή» (μέχρι και το `\n` δηλαδή) ως **συμβολοσειρά** (όταν η θέση ανάγνωση φτάσει το τέλος του αρχείου επιστρέφει την κενή συμβολοσειρά `' '`).

Παράδειγμα: (αρχείο `grades.txt` με βαθμούς)

```
f = open('grades.txt', 'r')

print(f.readline(), end = '')
print(f.readline(), end = '')
print(f.readline(), end = '')
```

```
Γιάννης 10
Μαρία 9
Κώστας 3
```



Αλλάζει η γραμμή γιατί οι γραμμές τελειώνουν με `\n`

Προσπέλαση Δεδομένων

Τα αρχεία ακολουθούν το πρωτόκολλο επανάληψης (η «επανάληψη» γίνεται ως προς τις **γραμμές**):

```
f = open('grades.txt', 'r')  
  
for line in f: # Από τη θέση ανάγνωσης και μετά  
    print(line, end = '')
```

```
Γιάννης 10  
Μαρία 9  
Κώστας 3  
Ελένη 6
```

Το τέλος του αρχείου σηματοδοτείται από την κενή γραμμή ' ':

```
f = open('grades.txt', 'r')  
  
line = 'κάτι'  
  
while line != '':  
    line = f.readline()  
    print(line, end = '')
```

```
Γιάννης 10  
Μαρία 9  
Κώστας 3  
Ελένη 6
```

Μέθοδοι Αρχείων

(Ανάγνωση αρχείου)

- Εξαγωγή γραμμών σε **λίστα**: `readlines()`

```
>>> f = open('grades.txt', 'r')
>>> L = f.readlines()
>>> L
['Γιάννης 10\n', 'Μαρία 9\n', 'Κώστας 3\n', 'Ελένη 6']
>>> for i in range(len(L)):
...     L[i] = L[i].rstrip('\n')
...
...
>>> L
['Γιάννης 10', 'Μαρία 9', 'Κώστας 3', 'Ελένη 6']
```

- Κλείσιμο αρχείου: `close()`

```
>>> f.close()
```

- Το αρχείο κλείνει (το αντικείμενο αρχείου συνεχίζει να υπάρχει αλλά πλέον είναι άχρηστο καθώς **δεν συνδέεται** με τα αρχεία)
- Όποιες αλλαγές έχουν γίνει μεταφέρονται (από τη μνήμη RAM) στο αρχείο (στον σκληρό δίσκο)

Μέθοδοι Αρχείων

(Επεξεργασία αρχείου)

ΠΡΟΣΟΧΗ: Πρέπει το αρχείο να έχει ανοιχθεί με τύπο προσπέλασης 'w' ή 'a' (αλλιώς θα δημιουργηθεί εξαίρεση)!

- Δημιουργία αρχείου:

```
>>> f = open('grades.txt', 'w')
```

Αν το αρχείο υπάρχει **διαγράφει** το περιεχόμενό του

```
>>> f = open('grades.txt', 'a')
```

Δεν διαγράφει το περιεχόμενο

- Εγγραφή δεδομένων: `write()`

μεταβλητή_αρχείου.write(συμβολοσειρά)

προσθέτει τη συμβολοσειρά στο **τέλος** του αρχείου:

```
>>> f.write('Γιάννης ') # Επιστρέφει το πλήθος χαρακτήρων  
8  
>>> f.write('10\n')  
3  
>>> f.close()
```

Μέθοδοι Αρχείων

(Επεξεργασία αρχείου)

```
>>> f = open('grades.txt') # Όχι προσπάθεια ισοδυναμεί με 'r'
>>> f.read()
'Γιάννης 10\n'
>>> f.close()
>>> f = open('grades.txt', 'a')
>>> f.write('Μαρία 9\nΚώστας 3\nΕλένη 6\n')
25
>>> f.close()
>>> f = open('grades.txt')
>>> f.read()
'Γιάννης 10\nΜαρία 9\nΚώστας 3\nΕλένη 6\n'
>>> f.close()
```

- Εγγραφή δεδομένων από **λίστα με συμβολοσειρές**: `writelines` (λίστα)

```
>>> f = open('grades.txt', 'a')
>>> f.writelines(['Πέτρος 4\n', 'Αλεξία 7\n', 'Δημήτρης 9\n'])
>>> f.close()
>>> f = open('grades.txt')
>>> f.read()
'Γιάννης 10\nΜαρία 9\nΚώστας 3\nΕλένη 6\nΠέτρος 4\nΑλεξία 7\nΔημήτρης 9\n'
>>> f.close()
```


Αρχεία με Αριθμητικά Δεδομένα

```
def get_prices():
    tmp_file = open('prices.txt', 'w')
    for i in range(1,13):
        price = input(f'Τιμή πώλησης {i}ου μήνα: ') # f-string
        tmp_file.write(price+'\n') # Πρέπει να προσθέσουμε το \n !!
    tmp_file.close()

def show_prices():
    tmp_file = open('prices.txt', 'r')
    i = 1
    for line in tmp_file:
        print(str(i)+'ος μήνας:', end = ' ')
        print(line.rstrip('\n')) # Πρέπει να αφαιρέσουμε το \n !!
        i += 1
    tmp_file.close()

def get_average_price():
    tmp_file = open('prices.txt', 'r')
    L = tmp_file.readlines()
    tmp_file.close()
    for i in range(len(L)):
        L[i] = float(L[i]) # int() και float() αγνοούν το \n !!
    return sum(L)/12

def main():
    get_prices()
    print('Οι τιμές τους τελευταίους 12 μήνες ήταν: ' )
    show_prices()
    print('Ο μέσος όρος τους είναι: ')
    print(format(get_average_price(), '.2f'))

main()
```


Παραδείγματα

```
# Πλήθος γραμμών σε αρχείο κειμένου
def line_count(file_name):
    tmp_file = open(file_name, 'r')
    count = 0
    for line in tmp_file:
        count += 1
    tmp_file.close()
    return count

# Πλήθος λέξεων σε αρχείο κειμένου
def word_count(file_name):
    tmp_file = open(file_name, 'r')
    count = 0
    for line in tmp_file:
        count += len(line.split())
    tmp_file.close()
    return count

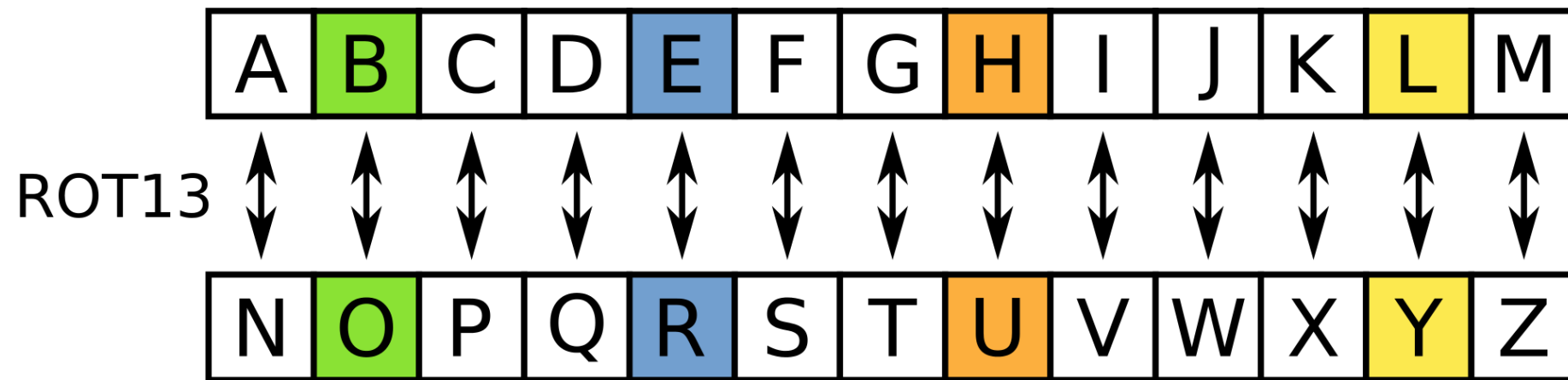
# Πλήθος χαρακτήρων σε αρχείο κειμένου (μετράμε και τα κενά)
def character_count(file_name):
    tmp_file = open(file_name, 'r')
    count = 0
    for line in tmp_file:
        count += len(line.rstrip('\n')) # Δεν μετράμε τα \n στο τέλος των γραμμών
    tmp_file.close()
    return count
```

```
>>> print(line_count('polis.txt'))
19
>>> print(word_count('polis.txt'))
151
>>> print(character_count('polis.txt'))
764
```

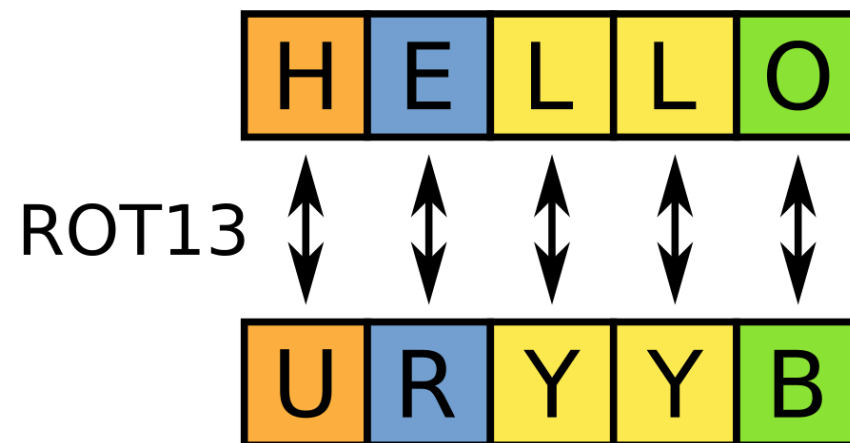
Παραδείγματα

Κωδικοποίηση ROT 13:

`ord('A')` == 65 == 78 - 13



`ord('N')` == 78 == 65 + 13



Παραδείγματα

```
# Κωδικοποίηση ROT13 συμβολοσειράς
def ROT13(S):
    encoded_string = ''
    for ch in S:
        letter_order = ord(ch) # Αριθμός ASCII χαρακτήρα
        if 'A' <= ch.upper() <= 'M':
            encoded_string += chr(letter_order+13) # Χαρακτήρας από ASCII
        elif 'N' <= ch.upper() <= 'Z':
            encoded_string += chr(letter_order-13)
        else:
            encoded_string += ch # Τα υπόλοιπα σύμβολα δεν τα αλλάζουμε!!
    return encoded_string

# Κωδικοποίηση ROT13 αρχείου
def file_encode():
    file_name = input('Όνομα αρχείου που θα κωδικοποιηθεί: ')
    tmp_file = open(file_name)
    S = tmp_file.read()
    tmp_file.close()
    S = ROT13(S)
    encoded_text_name = input('Όνομα κωδικοποιημένου αρχείου: ')
    tmp_file = open(encoded_text_name, 'w') # Δημιουργούμε καινούργιο αρχείο
    tmp_file.write(S)
    tmp_file.close()
```

```
>>> ROT13('Dimitris')
'Qvzvgevf'
>>> ROT13('Qvzvgevf')
'Dimitris'
```

Παραδείγματα

```
>>> file_encode()  
'Όνομα αρχείου που θα κωδικοποιηθεί: mirror.txt  
'Όνομα κωδικοποιημένου αρχείου: mirror_ROT13.txt  
>>> print(open('mirror_ROT13.txt').read())  
-- Zveebe --
```

V nz fvyire naq rknpq. V unir ab cerpbaprcgvbaf.
Jungrire V frr V fjnyybj vzzrqvngryl
Whfg nf vg vf, hazvfgrq ol ybir be qvfyvyr.
V nz abg pehry, bayl gehgushy,
Gur rlr bs n yvggyr tbq, sbhe-pbearerq.
Zbfg bs gur gvzr V zrqvgngv ba gur bccbfvgr jnyy.
Vg vf cvax, jvgu fcrpxyrf. V unir ybbxrg ng vg fb ybat
V guvax vg vf cneg bs zl urneg. Ohg vg syvpxref.
Snprf naq qnexarff frcnengr hf bire naq bire.

Abj V nz n ynrx. N jbzna oraqf bire zr,
Frnepuvat zl ernpurf sbe jung fur ernyyf vf.
Gura fur gheaf gb gubfr yvnef, gur pnaqyrf be gur zbba.
V frr ure onpx, naq ersyrpg vg snvgushyyf.
Fur erjneqf zr jvgu grnef naq na ntvgnvgvba bs unaqf.
V nz vzcbegnag gb ure. Fur pbzrf naq tbrf.
Rnpv zbeavat vg vf ure snpr gung ercynprf gur qnexarff.
Va zr fur unf qebjarq n lbhat tvey, naq va zr na byq jbzna
Evfrf gbjneq ure qnl nsgre qnl, yvyr n greevoyr svfu.

-- Flyivn Cyngu, 1961 --

Αρχεία CSV

(Comma Separated Values)

- Περιέχουν **οργανωμένα** δεδομένα σε στήλες και γραμμές.
- Τα στοιχεία κάθε στήλης χωρίζονται με ένα σύμβολο, π.χ. , ή ;
- Πολλές εφαρμογές εξάγουν τα δεδομένα σε αρχεία csv.

```
>>> f = open('grades.csv', 'w')
>>> f.write('Όνομα,Βαθμός\n') # Επικεφαλίδα
13
>>> f.write('Γιάννης,10\nΜαρία,9\nΚώστας,3\nΕλένη,6\nΠέτρος,4\nΑλεξία,7\nΔημήτρης,9\n')
65
>>> f.close()
>>> f = open('grades.csv')
>>> print(f.read())
Όνομα,Βαθμός
Γιάννης,10
Μαρία,9
Κώστας,3
Ελένη,6
Πέτρος,4
Αλεξία,7
Δημήτρης,9
>>> f.close()
```

Αρχεία CSV

(Comma Separated Values)

```
# Μέσος όρος βαθμών
```

```
def average_grade(file_name):  
    f = open(file_name)  
    sum_of_grades = 0  
    header = f.readline() # Προσπερνάμε την επικεφαλίδα  
    for line in f:  
        L = line.split(',')  
        grade = int(L[1]) # Δεν χρειάζεται rstrip('\n')  
        sum_of_grades += grade  
    f.seek(0) # Επιστρέφουμε στην αρχή του αρχείου  
    no_of_grades = len(f.readlines())-1 # Λόγω της επικεφαλίδας  
    f.close()  
    return sum_of_grades/no_of_grades
```

```
>>> print(format(average_grade('grades.csv'), '.2f'))  
6.86
```

Αρχεία CSV

(Comma Separated Values)

Παράδειγμα: (αρχείο `eclass_users.csv` με τους εγγεγραμμένους χρήστες στην e-class του μαθήματος)

```
# Έλεγχος αν κάποιος έχει εγγραφεί στην eclass
def find_user(name, surname):
    f = open('/Users/dzoros/Dropbox/Files/Work/Πληροφορική II/
eclass_users.csv')
    L = f.readlines()
    f.close()
    for i in range(len(L)):
        L[i]=L[i].rstrip('\n')

    for i in range(len(L)):
        L[i]=L[i].split(',') #Τα στοιχεία της L είναι πλέον λίστες

    for i in range(len(L)):
        if surname.upper() == L[i][0].upper() and\
            name.upper() == L[i][1].upper():
            print('Τον βρήκα!')
            break
    else:
        print('Δεν είναι πουθενά...')
```

```
>>> find_user(surname='Ζώρος', name='Δημήτρης')
Τον βρήκα!
```


Αρχεία CSV

(Comma Separated Values)

Παράδειγμα: (αρχείο `eclass_users.csv` με τους εγγεγραμμένους χρήστες στην e-class του μαθήματος)

```
# Έλεγχος αν κάποιος έχει εγγραφεί στην eclass
def find_user(name, surname):
    f = open('/Users/dzoros/Dropbox/Files/Work/Πληροφορική II/
eclass_users.csv')
    L = f.readlines()
    f.close()
    for element in L:
        element = element.rstrip('\n')

    for element in L:
        element = element.split(',')

    for element in L:
        if surname.upper() == element[0].upper() and\
           name.upper() == element[1].upper():
            print('Τον βρήκα!')
            break
    else:
        print('Δεν είναι πουθενά...')
```

```
>>> find_user(surname='Ζώρος', name='Δημήτρης')
Τον βρήκα!
```


Αρχεία CSV

(Comma Separated Values)

Παράδειγμα: (αρχείο `eclass_users.csv` με τους εγγεγραμμένους χρήστες στην e-class του μαθήματος)

```
# Έλεγχος αν κάποιος έχει εγγραφεί στην eclass
def find_user(name, surname):
    f = open('/Users/dzoros/Dropbox/Files/Work/Πληροφορική II/
eclass_users.csv')
    L = f.readlines()
    f.close()
    for element in L: # Αν η L έχει διακεκριμένα στοιχεία...
        L[L.index(element)] = element.rstrip('\n')

    for element in L: #Αν η L έχει ακόμα διακεκριμένα στοιχεία...
        L[L.index(element)] = element.split(',')

    for element in L:
        if surname.upper() == element[0].upper() and\
           name.upper() == element[1].upper():
            print('Τον βρήκα!')
            break
    else:
        print('Δεν είναι πουθενά...')
```

```
>>> find_user(surname='Ζώρος', name='Δημήτρης')
Τον βρήκα!
```

Αποθήκευση Σύνθετων Δομών

(Σε δυαδικά αρχεία)

Αν θέλουμε να αποθηκεύσουμε σε αρχείο κάποια σύνθετη δομή δεδομένων (λίστα, λεξικό κλπ.) χρησιμοποιούμε μία μέθοδο που ονομάζεται Pickling *.

To module `pickle`:

```
pickle.dump(δομή, αντικείμενο_δυναμικού_αρχείου)
```

```
όνομα_δομής = pickle.load(αντικείμενο_δυναμικού_αρχείου)
```

```
>>> import pickle
>>> f = open('pickle.bin', 'wb') # Δημιουργία δυναμικού αρχείου
>>> L = [1, 2, 3, 4]
>>> pickle.dump(L, f)
>>> f.close()
```

```
>>> import pickle
>>> f = open('pickle.bin', 'rb') # Άνοιγμα δυναμικού αρχείου
>>> L = pickle.load(f)
>>> L
[1, 2, 3, 4]
>>> f.close()
```

* Συντήρηση σε ελεύθερη μετάφραση. Η ακριβής μετάφραση του όρου *pickling* είναι *τουρσοποίηση*.

Αποθήκευση Σύνθετων Δομών

(Σε δυαδικά αρχεία)

Παράδειγμα:

```
# Βάση δεδομένων φοιτητών

import pickle

def new_student(student_db):
    name = input('Όνομα: ')
    id_no = int(input('Αριθμός Μητρώου: '))
    year = int(input('Έτος Σπουδών:'))
    return student_db.update({name:{'id_no':id_no, 'year':year}})

def print_db(student_db):
    for key in student_db:
        print(key, 'AM: '+str(student_db[key]['id_no']), \
              'Έτος: '+str(student_db[key]['year']), sep = ', ')

def save_db(student_db, file_name):
    tmp_file = open(file_name, 'wb')
    pickle.dump(student_db, tmp_file)
    tmp_file.close()

def load_db(file_name):
    tmp_file = open(file_name, 'rb')
    student_db = pickle.load(tmp_file)
    tmp_file.close()
    return student_db
```

Αποθήκευση Σύνθετων Δομών

(Σε δυαδικά αρχεία)

```
>>> db = {}
>>> new_student(db)
Όνομα: Μαρία
Αριθμός Μητρώου: 11122022000
Έτος Σπουδών: 1
>>> new_student(db)
Όνομα: Πέτρος
Αριθμός Μητρώου: 11122022001
Έτος Σπουδών: 1
>>> new_student(db)
Όνομα: Γιώργος
Αριθμός Μητρώου: 11122021000
Έτος Σπουδών: 2
>>> save_db(db, 'student_db.bin')
```

```
>>> db = load_db('student_db.bin')
>>> new_student(db)
Όνομα: Ελένη
Αριθμός Μητρώου: 11122020010
Έτος Σπουδών: 3
>>> print_db(db)
Μαρία, AM: 11122022000, Έτος: 1
Πέτρος, AM: 11122022001, Έτος: 1
Γιώργος, AM: 11122021000, Έτος: 2
Ελένη, AM: 11122020010, Έτος: 3
>>> save_db(db, 'student_db.bin')
```

Χειρισμός Εξαιρέσεων

- Αμυντικός Προγραμματισμός
- Παραδείγματα Εξαιρέσεων
- Χειρισμός Εξαιρέσεων
- `else` και `finally`
- Πρόκληση Εξαιρέσεων (`raise`)
- Μετάδοση Εξαιρέσεων

Αμυντικός Προγραμματισμός

Στον **αμυντικό προγραμματισμό** (defensive programming) **δεν** θεωρούμε δεδομένο ότι ο χρήστης θα ακολουθήσει τις προδιαγραφές που έχουμε θέσει:

```
print('Επέλεξε έναν ακέραιο αριθμό από το 1 έως και το 9.')  
print('Θα σε παρακαλέσω να μην δώσεις τον αριθμό ολογράφος.\n')  
x = int(input('Ποιον αριθμό θα επιλέξεις; '))
```

Επέλεξε έναν ακέραιο αριθμό από το 1 έως και το 9.
Θα σε παρακαλέσω να μην δώσεις τον αριθμό ολογράφος.

Ποιον αριθμό θα επιλέξεις; **Δύο**

Traceback (most recent call last):

File "/Users/dzoros/Desktop/test.py", line 3, in <module>

x = int(input('Ποιον αριθμό θα επιλέξεις; '))

ValueError: invalid literal for int() with base 10: 'Δύο'

Παρόλο που (ενδεχομένως) έχουμε ξεκαθαρίσει πλήρως τι περιμένουμε από τον χρήστη, πρέπει να αναπτύξουμε «αμυντικούς μηχανισμούς» για να αποφύγουμε τα λάθη.

Παραδείγματα Εξαιρέσεων

Εξαίρεση: Είναι ένα **σφάλμα** κατά την εκτέλεση του προγράμματος (run time error) που μπορούμε να το **προβλέψουμε** και να το «**χειριστούμε**».

Παράδειγμα εξαίρεσης:

```
y = 1/0 # Διαίρεση με μηδέν
```

```
Traceback (most recent call last):  
  File "/Users/dzoros/Desktop/test.py", line 1, in <module>  
    y = 1/0 # Διαίρεση με μηδέν  
ZeroDivisionError: division by zero
```

Όνομα εξαίρεσης

Περιγραφή εξαίρεσης

Εντολή που προκλήθηκε
η εξαίρεση

- Όταν προκληθεί εξαίρεση η εκτέλεση του προγράμματος **σταματάει απότομα**.
- Ο **χειρισμός** των εξαιρέσεων μας επιτρέπει (στη χειρότερη περίπτωση) να τερματίσουμε το πρόγραμμα με πιο ομαλό τρόπο (ολοκληρώνοντας τυχόν απαραίτητες ενέργειες).

Παραδείγματα Εξαιρέσεων

```
L = [1,2,3]
y = L[4] # Δείκτης εκτός λίστας
```

```
Traceback (most recent call last):
  File "/Users/dzoros/Desktop/test.py", line 2, in <module>
    y = L[4] # Δείκτης εκτός λίστας
IndexError: list index out of range
```

```
x = 3 + y # Ανύπαρκτό όνομα μεταβλητής
```

```
Traceback (most recent call last):
  File "/Users/dzoros/Desktop/test.py", line 1, in <module>
    x = 3+y # Ανύπαρκτό όνομα μεταβλητής
NameError: name 'y' is not defined
```

```
x = 2 + '3' # Εφαρμογή τελεστή σε λάθος τύπους
```

```
Traceback (most recent call last):
  File "/Users/dzoros/Desktop/test.py", line 1, in <module>
    x = 2 + '3' # Εφαρμογή τελεστή σε λάθος τύπους
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
x = int('ένα') # Αδυναμία μετατροπής
```

```
Traceback (most recent call last):
  File "/Users/dzoros/Desktop/test.py", line 1, in <module>
    x = int('ένα') # Αδυναμία μετατροπής
ValueError: invalid literal for int() with base 10: 'ένα'
```


Παραδείγματα Εξαιρέσεων

```
x = (1,2)
x[0] = 3 # Αλλαγή σε immutable αντικείμενο
```

```
Traceback (most recent call last):
  File "/Users/dzoros/Desktop/test.py", line 2, in <module>
    x[0] = 3 # Αλλαγή σε immutable αντικείμενο
TypeError: 'tuple' object does not support item assignment
```

```
x = open('nonexistent.txt', 'r') # Άνοιγμα ανύπαρκτου αρχείου
```

```
Traceback (most recent call last):
  File "/Users/dzoros/Desktop/test.py", line 1, in <module>
    x = open('nonexistent.txt', 'r') # Άνοιγμα ανύπαρκτου αρχείου
FileNotFoundError: [Errno 2] No such file or directory:
'nonexistent.txt'
```

```
x = open('existent.txt', 'r')
x.write('something') # Λάθος προσπάθεια αρχείου
```

```
Traceback (most recent call last):
  File "/Users/dzoros/Desktop/test.py", line 2, in <module>
    x.write('something') # Λάθος προσπάθεια αρχείου
io.UnsupportedOperation: not writable
```

Παραδείγματα Εξαιρέσεων

```
D = {}  
x = D['key'] # Ανύπαρκτο κλειδί
```

```
Traceback (most recent call last):  
  File "/Users/dzoros/Desktop/test.py", line 2, in <module>  
    x = D['key'] # Ανύπαρκτο κλειδί  
KeyError: 'key'
```

```
import nonexistent_module # Ανύπαρκτο module
```

```
Traceback (most recent call last):  
  File "/Users/dzoros/Desktop/test.py", line 1, in <module>  
    import nonexistent_module # Ανύπαρκτο module  
ModuleNotFoundError: No module named 'nonexistent_module'
```

```
x = 1  
def func():  
    print(x) # Αναφορά σε τοπική μεταβλητή πριν της δοθεί τιμή  
    x = 0  
func()
```

```
Traceback (most recent call last):  
  File "/Users/dzoros/Desktop/test.py", line 6, in <module>  
    func()  
  File "/Users/dzoros/Desktop/test.py", line 3, in func  
    print(x) # Αναφορά σε τοπική μεταβλητή πριν της δοθεί τιμή  
UnboundLocalError: local variable 'x' referenced before assignment
```

Παραδείγματα Εξαιρέσεων

```
L = [1,2,3]
it = iter(L)
x = next(it) # Παίρνουμε το επόμενο στοιχείο
print(x)
x = next(it)
print(x)
x = next(it)
print(x)
x = next(it) # Δεν υπάρχει επόμενο στοιχείο
print(x)
```

```
1
2
3
```

```
Traceback (most recent call last):
  File "/Users/dzoros/Desktop/test.py", line 9, in <module>
    x = next(it) # Δεν υπάρχει επόμενο στοιχείο
StopIteration
```

Παραδείγματα Εξαιρέσεων

Ιεραρχία κλάσεων εξαιρέσεων:

```
BaseException
• Exception
  - ArithmeticError
    FloatingPointError
    OverflowError
    ZeroDivisionError
  - AssertionError
  - AttributeError
  - BufferError
  - EOFError
  - ImportError
    ModuleNotFoundError
  - LookupError
    IndexError
    KeyError
  - MemoryError
  - NameError
    UnboundLocalError
  - OSError
    BlockingIOError
    ChildProcessError
    ConnectionError
      BrokenPipeError
      ConnectionAbortedError
      ConnectionRefusedError
      ConnectionResetError
    FileExistsError
    FileNotFoundError
    InterruptedError
    IsADirectoryError
    NotADirectoryError
    PermissionError
    ProcessLookupError
    TimeoutError
  - ReferenceError
  - RuntimeError
    NotImplementedError
    RecursionError
  - StopIteration
  - StopAsyncIteration
  - SyntaxError
    IndentationError
    TabError
  - SystemError
  - TypeError
  - ValueError
    UnicodeError
      UnicodeDecodeError
      UnicodeEncodeError
      UnicodeTranslateError
  - Warning
    BytesWarning
    DeprecationWarning
    FutureWarning
    ImportWarning
    PendingDeprecationWarning
    ResourceWarning
    RuntimeWarning
    SyntaxWarning
    UnicodeWarning
    UserWarning
• GeneratorExit
• KeyboardInterrupt
• SystemExit
```

Είδαμε:

ZeroDivisionError

IndexError

NameError

TypeError

ValueError

FileNotFoundError

io.UnsupportedOperation

KeyError

ModuleNotFoundError

UnboundLocalError

StopIteration

Χειρισμός Εξαιρέσεων

Κάποιες εξαιρέσεις μπορούμε να τις «χειριστούμε» και με τη δομή ελέγχου `if`:

```
# Αναλογίας φοιτητών που πέρασαν το μάθημα

passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))

ratio = passed/failed

print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))
```

Πλήθος φοιτητών που πέρασαν: **110**

Πλήθος φοιτητών που κόπηκαν: **0**

Traceback (most recent call last):

File `"/Users/dzoros/Desktop/test.py"`, line 6, in `<module>`

`ratio = passed/failed`

ZeroDivisionError: division by zero

Χειρισμός Εξαιρέσεων

Κάποιες εξαιρέσεις μπορούμε να τις «χειριστούμε» και με τη δομή ελέγχου `if`:

```
# Αναλογίας φοιτητών που πέρασαν το μάθημα

passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))

if failed == 0:
    print('Όλοι πέρασαν το μάθημα!')
else:
    ratio = passed/failed
    print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))
```

```
Πλήθος φοιτητών που πέρασαν: 110
Πλήθος φοιτητών που κόπηκαν: 0
Όλοι πέρασαν το μάθημα!
```

```
Πλήθος φοιτητών που πέρασαν: μπλε
Traceback (most recent call last):
  File "/Users/dzoros/Desktop/test.py", line 3, in <module>
    passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
ValueError: invalid literal for int() with base 10: 'μπλε'
```

Σε άλλες όμως δεν μπορούμε να χρησιμοποιήσουμε `if` (ή δεν είναι ιδιαίτερα βολικό).

Χειρισμός Εξαιρέσεων

Χειριστής εξαιρέσεων:

```
try:
    εντολή
    εντολή
    ...
except όνομα_εξαίρεσης:
    εντολή
    εντολή
    ...
```

} Οι εντολές που μπορούν να προκαλέσουν εξαίρεση

} Οι εντολές που θα εκτελεσθούν αν προκληθεί η εξαίρεση όνομα_εξαίρεσης

```
# Αναλογίας φοιτητών που πέρασαν το μάθημα
```

```
try:
```

```
    passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
```

```
    failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))
```

```
    ratio = passed/failed
```

```
    print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))
```

```
except ZeroDivisionError:
```

```
    print('Όλοι πέρασαν το μάθημα!')
```


Χειρισμός Εξαιρέσεων

Μπορούμε να χειριστούμε παραπάνω από μία εξαίρεση **ταυτόχρονα**:

```
# Αναλογίας φοιτητών που πέρασαν το μάθημα

try:
    passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
    failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))

    ratio = passed/failed

    print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))

except ValueError:
    print('Δεν έδωσες ακέραιο αριθμό!')

except ZeroDivisionError:
    print('Όλοι πέρασαν το μάθημα!')
```

Πλήθος φοιτητών που πέρασαν: **μπλε**
Δεν έδωσες ακέραιο αριθμό!

Πλήθος φοιτητών που πέρασαν: **110**
Πλήθος φοιτητών που κόπηκαν: **0**
Όλοι πέρασαν το μάθημα!

Χειρισμός Εξαιρέσεων

Οι εντολές που ακολουθούν τον χειριστή εξαιρέσεων εκτελούνται **πάντα**.
(Εκτός φυσικά αν στο μεσοδιάστημα προκύψει εξαίρεση που δεν έχουμε χειριστεί.)

```
# Αναλογίας φοιτητών που πέρασαν το μάθημα

try:
    passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
    failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))
except ValueError:
    print('Δεν έδωσες ακέραιο αριθμό!')

try:
    ratio = passed/failed
except ZeroDivisionError:
    print('Όλοι πέρασαν το μάθημα!')

print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))
```

Πλήθος φοιτητών που πέρασαν: **μπλε**

Δεν έδωσες ακέραιο αριθμό!

Traceback (most recent call last):

File "/Users/dzoros/Desktop/test.py", line 10, in <module>
 ratio = passed/failed

NameError: name 'passed' is not defined



Δεν τη χειριστήκαμε!

Χειρισμός Εξαιρέσεων

Οι εντολές που ακολουθούν τον χειριστή εξαιρέσεων εκτελούνται **πάντα**.
(Εκτός φυσικά αν στο μεσοδιάστημα προκύψει εξαίρεση που δεν έχουμε χειριστεί.)

```
# Αναλογίας φοιτητών που πέρασαν το μάθημα

try:
    passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
    failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))
except ValueError:
    print('Δεν έδωσες ακέραιο αριθμό!')

try:
    ratio = passed/failed
except ZeroDivisionError:
    print('Όλοι πέρασαν το μάθημα!')

print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))
```

Πλήθος φοιτητών που πέρασαν: **110**

Πλήθος φοιτητών που κόπηκαν: **0**

Όλοι πέρασαν το μάθημα!

Traceback (most recent call last):

File "/Users/dzoros/Desktop/test.py", line 14, in <module>

print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))

NameError: name 'ratio' is not defined



Δεν τη χειριστήκαμε!

Χειρισμός Εξαιρέσεων

- Όταν προκαλείται μια εξαίρεση δημιουργείται ένα «αντικείμενο εξαίρεσης».
- Η τιμή του συνήθως είναι ένα προκαθορισμένο **μήνυμα σφάλματος**.
- Για να μπορέσουμε να χρησιμοποιήσουμε αυτό το αντικείμενο πρέπει μέσα στην «ρήτρα» `except` να προσδιορίσουμε ρητά το όνομά του:

```
except όνομα_εξαίρεσης as όνομα_αντικειμένου:
```

```
try:  
    x = 1/0  
  
except ZeroDivisionError as error:  
    print(type(error)) # Τύπος αντικειμένου  
    print(error)      # Μήνυμα εξαίρεσης
```

```
<class 'ZeroDivisionError'>  
division by zero
```

Με αυτόν τον τρόπο μπορούμε να χρησιμοποιήσουμε τα προκαθορισμένα μηνύματα των αντικειμένων εξαίρεσης. (Πολλές φορές περιέχουν πιο πολλές πληροφορίες για την εξαίρεση που προκλήθηκε.)

Χειρισμός Εξαιρέσεων

```
# Αναλογίας φοιτητών που πέρασαν το μάθημα

try:
    passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
    failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))

    ratio = passed/failed

    print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))

except ValueError as error:
    print(error)

except ZeroDivisionError as error:
    print(error)
```

```
Πλήθος φοιτητών που πέρασαν: μπλε
invalid literal for int() with base 10: 'μπλε'
```

```
Πλήθος φοιτητών που πέρασαν: 110
Πλήθος φοιτητών που κόπηκαν: 0
division by zero
```

Χειρισμός Εξαιρέσεων

Μπορούμε να χειριστούμε όλες τις πιθανές εξαιρέσεις με **μία μόνο** ρήτρα `except`, χωρίς να αναφέρουμε το όνομα της εξαίρεσης που θέλουμε να χειριστούμε:

```
# Αναλογίας φοιτητών που πέρασαν το μάθημα

try:
    passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
    failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))

    ratio = passed/failed

    print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))

except:
    print('Πρόβλημα!')
```

```
Πλήθος φοιτητών που πέρασαν: μπλε
Πρόβλημα!
```

```
Πλήθος φοιτητών που πέρασαν: 110
Πλήθος φοιτητών που κόπηκαν: 0
Πρόβλημα!
```

Κακή πρακτική:

Με αυτόν τον τρόπο όμως **δεν** δημιουργείται το αντικείμενο της εξαίρεσης και **δεν** μπορούμε να γνωρίζουμε τι εξαίρεση προκλήθηκε τελικά!

Χειρισμός Εξαιρέσεων

- Όταν προκύψει μία εξαίρεση ο διερμηνέας εξετάζει τις ρήτρες με τη **σειρά** που έχουν γραφτεί.
- Όταν βρει τη ρήτρα με το όνομα της εξαίρεσης που προκλήθηκε εκτελεί τις εντολές χειρισμού.
- Βγαίνει από το `try`-μπλοκ.

Συνεπώς, σωστή πρακτική είναι να βάλουμε τις ρήτρες με τις πιο ειδικές εξαιρέσεις **πριν** από αυτές με τις πιο γενικές (και πιθανώς να τελειώνουμε με μια εξαίρεση χωρίς όνομα ως «τελευταία γραμμή άμυνας»):

```
# Λάθος σειρά
try:
    passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
    failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))

    ratio = passed/failed

    print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))

except ArithmeticError: # Γενικότερη εξαίρεση
    print('Αριθμητικό λάθος!')

except ZeroDivisionError: # Ειδικότερη εξαίρεση
    print('Διαίρεση με μηδέν!')
```

```
Πλήθος φοιτητών που πέρασαν: 110
Πλήθος φοιτητών που κόπηκαν: 0
Αριθμητικό λάθος!
```

else και finally

- `else:`

```
try:
    εντολή
    εντολή
    ...
except όνομα_εξαίρεσης:
    εντολή
    εντολή
    ...
else:
    εντολή
    εντολή
    ...
```

Εκτελούνται **μόνο αν δεν** προκληθεί εξαίρεση
(μετά από τις εντολές στο `try`-μπλοκ)

Χρησιμεύει ώστε να μην βάζουμε όλον τον κώδικα στο `try`-μπλοκ:

```
# Αναλογίας φοιτητών που πέρασαν το μάθημα
try:
    passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
    failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))

    ratio = passed/failed

except ValueError:
    print('Δεν έδωσες ακέραιο αριθμό!')

except ZeroDivisionError:
    print('Όλοι πέρασαν το μάθημα!')
else:
    print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))
```


else και finally

Χρησιμοποιεί επίσης για να **επαναλαμβάνουμε** τον κώδικα μέχρι ο χρήστης να δώσει σωστή είσοδο:

```
# Αναλογία φοιτητών που πέρασαν το μάθημα
while True:
    try:
        passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
        failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))

        ratio = passed/failed

    except ValueError:
        print('Δεν έδωσες ακέραιο αριθμό!')

    except ZeroDivisionError:
        print('Όλοι πέρασαν το μάθημα!')
        break # Σωστή είσοδος
    else:
        print('Η αναλογία πέρασαν/κόπηκαν είναι:', format(ratio, '.2f'))
        break
```


else **και** finally

- Finally:

```
try:
    εντολή
    εντολή
    ...
except όνομα_εξαίρεσης:
    εντολή
    εντολή
    ...
finally:
    εντολή
    εντολή
    ...
```

} Εκτελούνται **πάντα**, ανεξάρτητα από το αν προκληθεί εξαίρεση ή όχι

Χρησιμεύει για να κάνουμε κάποιες «απαραίτητες εργασίες» **ακόμα και αν** προκληθεί εξαίρεση:

```
try:
    f = open('file.txt', 'r')
    L = f.readlines()
    f.close()
    print(L[1].rstrip('\n'))

except IndexError as error: # Αν δεν υπάρχει δεύτερη γραμμή στο αρχείο
    print(error)

finally:
    print('Τερματισμός...')
```

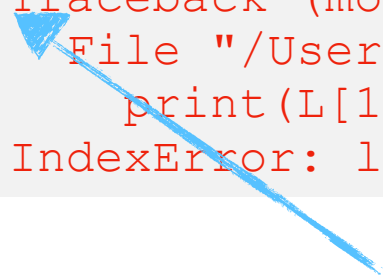
else και finally

Το μπλοκ `finally` εκτελείται ακόμα και αν προκληθεί εξαίρεση που **δεν** χειριστήκαμε:

```
try:
    f = open('file.txt', 'r')
    L = f.readlines()
    f.close()
    print(L[1].rstrip('\n'))

except FileNotFoundError as error: # Αν δεν υπάρχει το αρχείο
    print(error)

finally:
    print('Τερματισμός...')
```

1. Τερματισμός...
2. Traceback (most recent call last):
File "/Users/dzoros/Desktop/test.py", line 5, in <module>
 print(L[1].rstrip('\n'))
IndexError: list index out of range

Ολοκληρώνεται ο «χειρισμός» των εξαιρέσεων
(με τις εντολές του `finally`) και μετά
«προκαλείται» η εξαίρεση `IndexError`

Παραδείγματα

```
LAST_YEAR = 3.5
file_name = input('Όνομα αρχείου βαθμολογιών: ') # Αρχείο .csv με τις βαθμολογίες
sum_of_grades = 0
number_of_students = 0
try:
    f = open(file_name, 'r') # Πιθανό FileNotFoundError
except FileNotFoundError:
    print('-'*60)
    print('Σφάλμα ανάγνωσης αρχείου')
    print('-'*60)
else:
    try:
        first_line = f.readline() # Προσπερνάμε την επικεφαλίδα
        for line in f:
            student_data = line.split(',')
            grade = int(student_data[2]) # Πιθανό IndexError και ValueError
            sum_of_grades += grade
            number_of_students += 1
        average = sum_of_grades/number_of_students # Πιθανό ZeroDivisionError
    except IndexError:
        print('-'*60)
        print('Λιγότερα πεδία στο αρχείο')
        print('-'*60)
    except ValueError:
        print('-'*60)
        print('Όχι ακέραιος αριθμός στο τρίτο πεδίο')
        print('-'*60)
    except ZeroDivisionError:
        print('-'*60)
        print('Διαίρεση με το μηδέν')
        print('-'*60)
    else:
        print('Ο μέσος όρος είναι:', format(average, '.2f'))
    finally:
        f.close() # Όχι μέσα στο try-μπλοκ!
print('Ο περσινός μέσος όρος ήταν:', LAST_YEAR)
```

Παραδείγματα

Συνάρτηση που ελέγχει αν η είσοδος έχει τον σωστό τύπο:

```
def input_checker(input_type, input_message, error_message):  
    while True: # Επανάληψη μέχρι να φτάσουμε σε return  
        tmp = input(input_message)  
        try:  
            x = input_type(tmp) # Δοκιμάζουμε αν μπορούμε να  
                                # κάνουμε μετατροπή στον τύπο  
                                # που εισήγαγε ο χρήστης  
        except ValueError:  
            print(error_message)  
        else:  
            return x
```

```
>>> x = input_checker(int, 'Δώσε έναν ακέραιο: ', 'Αυτό δεν είναι  
ακέραιος!')
```

```
Δώσε έναν ακέραιο: Δύο  
Αυτό δεν είναι ακέραιος!  
Δώσε έναν ακέραιο: 4.5  
Αυτό δεν είναι ακέραιος!  
Δώσε έναν ακέραιο: 3  
>>> x  
3
```

Η πρώτη παράμετρος
είναι συνάρτηση

Πρόκληση Εξαιρέσεων (raise)

Μπορούμε να προκαλέσουμε και εμείς εξαίρεση με την εντολή `raise`:

```
>>> raise ZeroDivisionError
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    raise ZeroDivisionError
ZeroDivisionError
>>> raise ValueError
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    raise ValueError
ValueError
>>> raise IndexError
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    raise IndexError
IndexError
>>> raise IOError
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    raise IOError
OSError
```

Πρόκληση Εξαιρέσεων (raise)

Εφαρμογή:

```
try:
    month = int(input('Δώσε μου ένα μήνα (1-12): '))
except ValueError:
    print('Πρέπει να δώσεις ακέραιο αριθμό 1-12.')
else:
    if month in [12,1,2,3]:
        print('Κάνει κρύο')
    elif month in [4,5,10,11]:
        print('Έχει δροσιά')
    elif month in [6,7,8,9]:
        print('Έχει ζέστη')
```

Δώσε μου ένα μήνα (1-12): **4**
Έχει δροσιά

Δώσε μου ένα μήνα (1-12): **Απρίλιος**
Πρέπει να δώσεις ακέραιο αριθμό 1-12.

Δώσε μου ένα μήνα (1-12): **13**

Πρόκληση Εξαιρέσεων (raise)

Εφαρμογή:

```
try:
    month = int(input('Δώσε μου ένα μήνα (1-12): '))

    # Αν δεν δώσει ακέραιο 1-12 θα προκαλέσουμε εξαίρεση
    # τύπου ValueError
    if month < 1 or month > 12:
        raise ValueError

except ValueError:
    print('Πρέπει να δώσεις ακέραιο αριθμό 1-12.')
else:
    if month in [1,2,3]:
        print('Κάνει κρύο')
    elif month in [4,5,10,11]:
        print('Έχει δροσιά')
    elif month in [6,7,8,9]:
        print('Έχει ζέστη')
```

Δώσε μου ένα μήνα (1-12): **Απρίλιος**
Πρέπει να δώσεις ακέραιο αριθμό 1-12.

Δώσε μου ένα μήνα (1-12): **13**
Πρέπει να δώσεις ακέραιο αριθμό 1-12.

Πρόκληση Εξαιρέσεων (raise)

Εφαρμογή:

```
try:
    month = int(input('Δώσε μου ένα μήνα (1-12): '))

    # Αν θέλουμε διαφορετικό μήνυμα προκαλούμε εξαίρεση
    # άλλου τύπου...
    if month < 1 or month > 12:
        raise ZeroDivisionError

except ValueError:
    print('Πρέπει να δώσεις ακέραιο αριθμό 1-12')

# ...και τη χειριζόμαστε ξεχωριστά
except ZeroDivisionError:
    print('Έχουμε μόνο 12 μήνες!')

else:
    if month in [12, 1, 2, 3]:
        print('Κάνει κρύο')
    elif month in [4, 5, 10, 11]:
        print('Έχει δροσιά')
    elif month in [6, 7, 8, 9]:
        print('Έχει ζέστη')
```

Δώσε μου ένα μήνα (1-12): **13**
Έχουμε μόνο 12 μήνες!

Πρόκληση Εξαιρέσεων (raise)

Όταν προκαλούμε εξαίρεση με τη `raise` μπορούμε να προσθέσουμε και «περιγραφή» της εξαίρεσης:

```
>>> raise ZeroDivisionError('Διαίρεση με το μηδέν!')
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    raise ZeroDivisionError('Διαίρεση με το μηδέν!')
ZeroDivisionError: Διαίρεση με το μηδέν!
>>> raise ValueError('Αδυναμία μετατροπής!')
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    raise ValueError('Αδυναμία μετατροπής!')
ValueError: Αδυναμία μετατροπής!
>>> raise IndexError('Δείκτης εκτός λίστας!')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    raise IndexError('Δείκτης εκτός λίστας!')
IndexError: Δείκτης εκτός λίστας!
>>> raise IOError('Το αρχείο δεν βρέθηκε!')
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    raise IOError('Το αρχείο δεν βρέθηκε!')
OSError: Το αρχείο δεν βρέθηκε!
```

Πρόκληση Εξαιρέσεων (raise)

Εφαρμογή:

```
try:
    month = int(input('Δώσε μου ένα μήνα (1-12): '))

    # Αν θέλουμε διαφορετικό μήνυμα προκαλούμε εξαίρεση
    # άλλου τύπου
    if month < 1 or month > 12:
        raise ZeroDivisionError('Έχουμε μόνο 12 μήνες!')

except ValueError:
    print('Πρέπει να δώσεις ακέραιο αριθμό 1-12')

# και τη χειριζόμαστε ξεχωριστά
except ZeroDivisionError as error:
    print(error)

else:
    if month in [12,1,2,3]:
        print('Κάνει κρύο')
    elif month in [4,5,10,11]:
        print('Έχει δροσιά')
    elif month in [6,7,8,9]:
        print('Έχει ζέστη')
```

Δώσε μου ένα μήνα (1-12): **13**
Έχουμε μόνο 12 μήνες!

Πρόκληση Εξαιρέσεων (raise)

Εφαρμογή:

Ή εναλλακτικά:

```
try:
    month = int(input('Δώσε μου ένα μήνα (1-12): '))

    if month < 1 or month > 12:
        raise ValueError('Έχουμε μόνο 12 μήνες!')

except ValueError as error:
    print(error)

else:
    if month in [12,1,2,3]:
        print('Κάνει κρύο')
    elif month in [4,5,10,11]:
        print('Έχει δροσιά')
    elif month in [6,7,8,9]:
        print('Έχει ζέστη')
```

Δώσε μου ένα μήνα (1-12): **13**
Έχουμε μόνο 12 μήνες!

Δώσε μου ένα μήνα (1-12): **Απρίλιος**
invalid literal for int() with base 10: 'Απρίλιος'

Μετάδοση Εξαιρέσεων

- Οι εξαιρέσεις «μεταδίδονται» από μία συνάρτηση **στην συνάρτηση που την κάλεσε**.
- Αυτό θα συμβαίνει μέχρι **μέχρι** να βρεθεί μία συνάρτηση που τη χειρίζεται (ή μέχρι να φτάσουμε στην αρχική συνάρτηση και να διακοπεί απότομα το πρόγραμμα).

```
def A():  
    print('Αρχή A')  
    print(1/0) #  Διαιρέση με μηδέν!  
    print('Τέλος A')
```

```
def B():  
    print('Αρχή B')  
    A()  
    print('Τέλος B')
```

```
def C():  
    print('Αρχή C')  
    B()  
    print('Τέλος C')
```

```
>>> C()  
Αρχή C  
Αρχή B  
Αρχή A  
Traceback (most recent call last):  
  File "<pyshell#0>", line 1, in <module>  
    C()  
  File "/Users/dzoros/Desktop/test.py", line 13, in C  
    B()  
  File "/Users/dzoros/Desktop/test.py", line 8, in B  
    A()  
  File "/Users/dzoros/Desktop/test.py", line 3, in A  
    print(1/0) #  Διαιρέση με μηδέν!  
ZeroDivisionError: division by zero
```

Μετάδοση Εξαιρέσεων

```
def A():  
    print('Αρχή A')  
    try:  
        print(1/0) # Διαίρεση με μηδέν!  
    except ZeroDivisionError:  
        print('Τη χειρίστηκε η A')  
    print('Τέλος A')  
  
def B():  
    print('Αρχή B')  
    A()  
    print('Τέλος B')  
  
def C():  
    print('Αρχή C')  
    B()  
    print('Τέλος C')
```

```
>>> C()  
Αρχή C  
Αρχή B  
Αρχή A  
Τη χειρίστηκε η A  
Τέλος A  
Τέλος B  
Τέλος C
```

← Ολοκληρώθηκαν **όλες** οι συναρτήσεις

Μετάδοση Εξαιρέσεων

```
def A():  
    print('Αρχή A')  
    print(1/0) # Διαίρεση με μηδέν!  
    print('Τέλος A')  
  
def B():  
    print('Αρχή B')  
    try:  
        A()  
    except ZeroDivisionError:  
        print('Τη χειρίστηκε η B')  
    print('Τέλος B')  
  
def C():  
    print('Αρχή C')  
    B()  
    print('Τέλος C')
```

```
>>> C()  
Αρχή C  
Αρχή B  
Αρχή A  
Τη χειρίστηκε η B  
Τέλος B  
Τέλος C
```

← Η A() **δεν** ολοκληρώθηκε

Μετάδοση Εξαιρέσεων

```
def A():  
    print('Αρχή A')  
    print(1/0) # Διαίρεση με μηδέν!  
    print('Τέλος A')  
  
def B():  
    print('Αρχή B')  
    A()  
    print('Τέλος B')  
  
def C():  
    print('Αρχή C')  
    try:  
        B()  
    except ZeroDivisionError:  
        print('Τη χειρίστηκε η C')  
    print('Τέλος C')
```

```
>>> C()  
Αρχή C  
Αρχή B  
Αρχή A  
Τη χειρίστηκε η C  
Τέλος C
```

← Ούτε η B() **δεν** ολοκληρώθηκε

Μετάδοση Εξαιρέσεων

Παράδειγμα:

```
def get_inputs():
    passed = int(input('Πλήθος φοιτητών που πέρασαν: '))
    failed = int(input('Πλήθος φοιτητών που κόπηκαν: '))
    return passed, failed # Επιστρέφει πλειάδα

def get_ratio(passed, failed):
    ratio = passed/failed
    return ratio

def main():
    while True:
        try:
            passed, failed = get_inputs() # Επακετάρισμα πλειάδας
            ratio = get_ratio(passed, failed)
        except ValueError: # Εξαίρεση της get_inputs()
            print('Δεν έδωσες ακέραιους αριθμούς!')
        except ZeroDivisionError: # Εξαίρεση της get_ratio()
            print('Όλοι πέρασαν το μάθημα!')
            break
        else:
            print('Η αναλογία είναι:', format(ratio, '.2f'))
            break

main()
```


Κλάσεις και Αντικείμενα

- Αντικειμενοστραφής Προγραμματισμός
- Κλάσεις και Αντικείμενα
- Προστασία Ιδιοχαρακτηριστικών
- Η Μέθοδος `__str__`
- Υπερφόρτωση Τελεστών
- Παραδείγματα

Αντικειμενοστραφής Προγραμματισμός

Διαδικαστικός

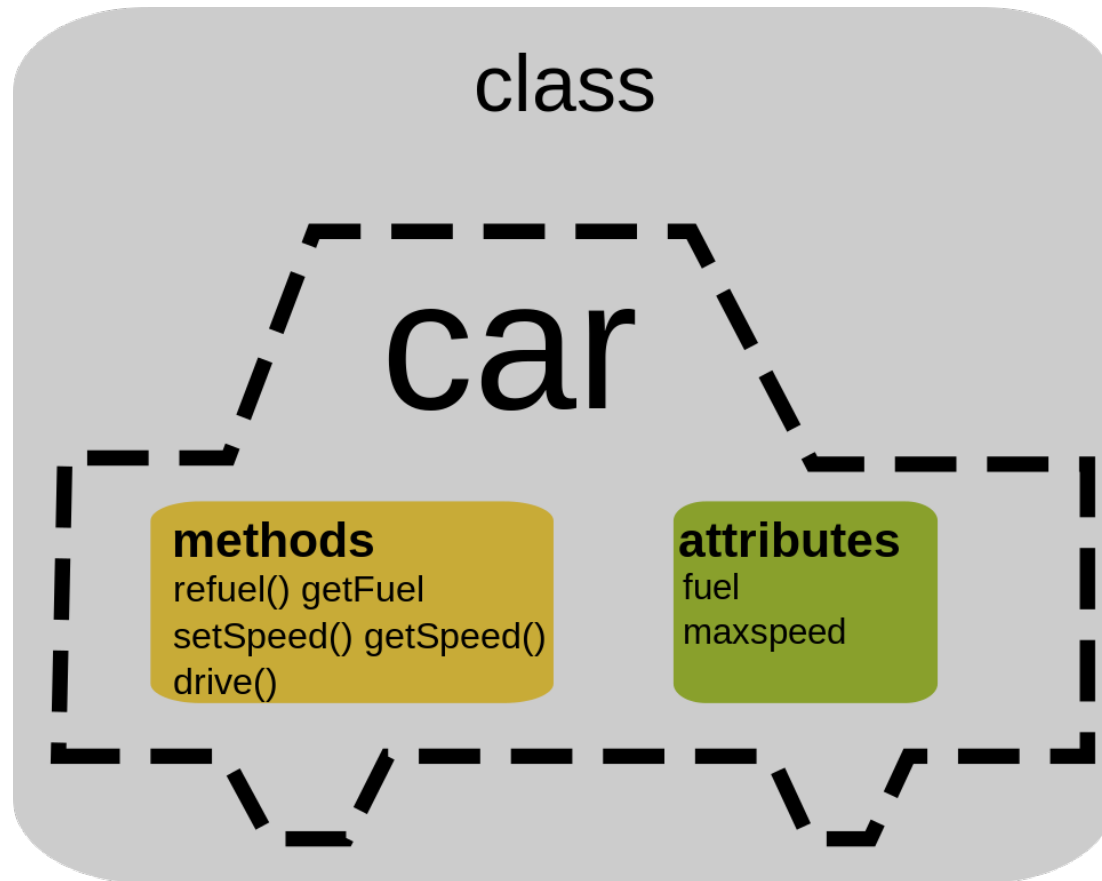
- Χωρισμός προβλήματος σε υποπροβλήματα (τα «λύνουμε» ξεχωριστά με συναρτήσεις)
- Το πρόγραμμα είναι μια αλληλουχία συναρτήσεων
- Τα δεδομένα είναι ανεξάρτητα από τις συναρτήσεις
- Χρήση μόνο των βασικών τύπων δεδομένων (`int`, `str`, `list`, κλπ.)
- ✓ Ενδεικτικός για καθαρά υπολογιστικά προβλήματα (που έχουν μορφή που δεν μεταβάλλεται ιδιαίτερα από εφαρμογή σε εφαρμογή)

Αντικειμενοστραφής

- Το πρόβλημα εκφράζεται μέσω **αντικειμένων** που προσομοιώνουν τον πραγματικό κόσμο
- Το πρόγραμμα αποτελείται από «ενέργειες» πάνω σε αυτά τα αντικείμενα και από **αλληλεπιδράσεις** μεταξύ τους
- Δεδομένα και συναρτήσεις **συνυπάρχουν** μέσα στα αντικείμενα (**ενθυλάκωση**)
- Τα αντικείμενα είναι **σύνθετοι** (αφηρημένοι) τύποι δεδομένων
- ✓ Χρήσιμος για την χρησιμοποίηση (του μεγαλύτερου τμήματος) του κώδικα στην επίλυση πολλών παραπλήσιων προβλημάτων

Αντικειμενοστραφής Προγραμματισμός

Γιατί χρειαζόμαστε τα αντικείμενα;

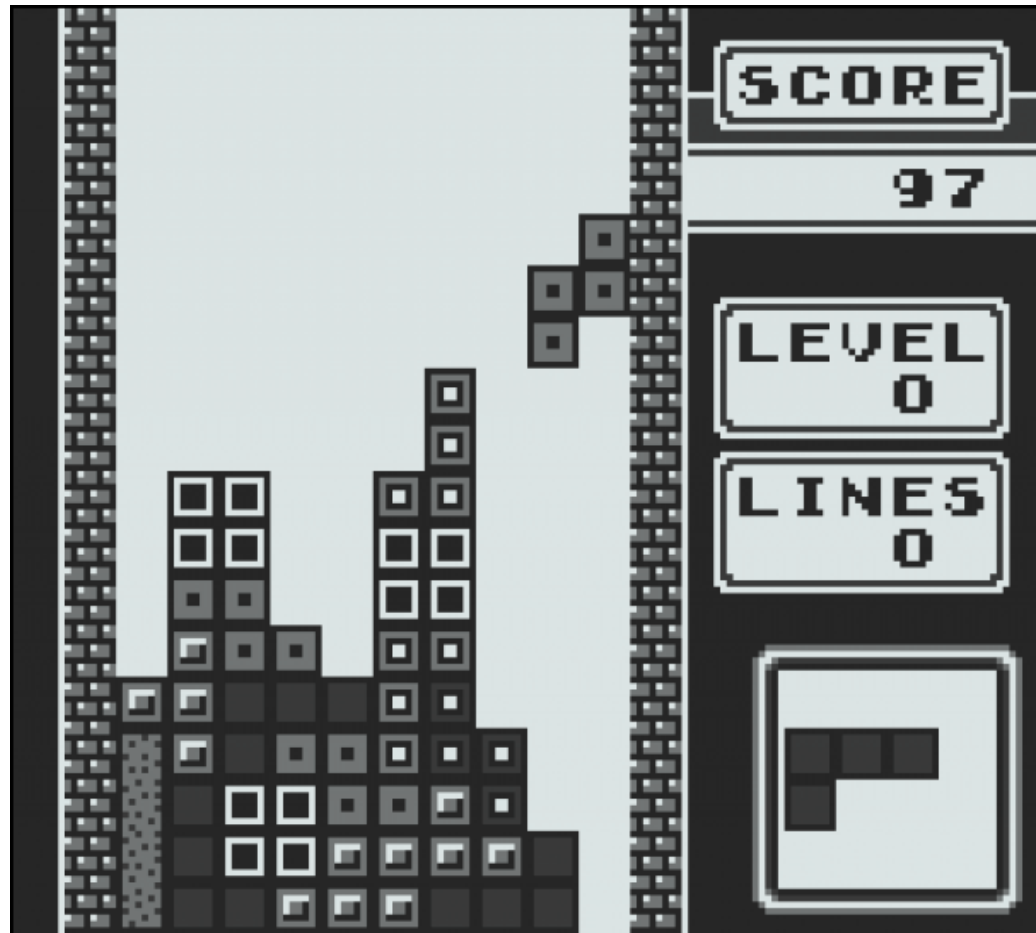


«Προσομοιώνουν» **πραγματικά αντικείμενα** και μας δίνουν τη δυνατότητα να έχουμε ειδικούς τύπους δεδομένων για αυτά.

Μπορούμε να ορίσουμε μία **κλάση** για το αυτοκίνητο που θα περιέχει τα βασικά χαρακτηριστικά ενός αυτοκινήτου (μάρκα, μοντέλο, χρώμα κ.λπ.) και βασικές λειτουργίες με αυτά (ή σε αυτά) τα χαρακτηριστικά.

Αντικειμενοστραφής Προγραμματισμός

Γιατί χρειαζόμαστε τα αντικείμενα;

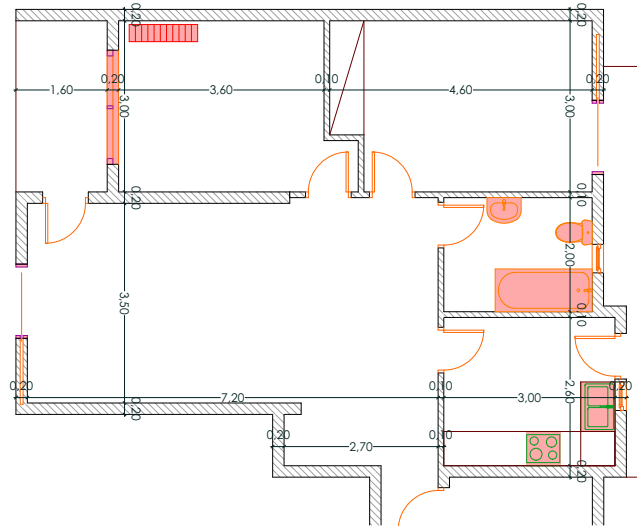


Έχουμε πολλά στιγμιότυπα της ίδιας οντότητας (τουβλάκια), που διαφέρουν μόνο στο σχήμα.

Μπορούμε να ορίσουμε μία **κλάση** (το «πρωτότυπο-γενικό» τουβλάκι) και έπειτα να δημιουργήσουμε «αντίγραφα-στιγμιότυπα» αυτής της κλάσης που θα έχουν και κοινά χαρακτηριστικά με το πρωτότυπο, αλλά και ξεχωριστά χαρακτηριστικά το καθένα.

Κλάσεις και Αντικείμενα

Κλάση: **Αφηρημένοι τύποι δεδομένων** που αποτελούν τα πρότυπα των αντικειμένων και «περιγράφουν» τα χαρακτηριστικά τους



«Αρχιτεκτονικό σχέδιο»

Αντικείμενα: «Στιγμιότυπα» δημιουργημένα στα πρότυπα της κλάσης που χρησιμοποιούνται στο πρόγραμμα μας



Κλάσεις και Αντικείμενα

Ορισμός κλάσης:

```
class Όνομα_κλάσης:  
    ορισμοί_πεδίων  
    ...  
    ορισμοί_μεθόδων  
    ...
```

```
class Client:  
    pass # Προσωρινό (για να μην έχουμε συντακτικό λάθος)
```

Δημιουργία αντικειμένου: `όνομα_αντικειμένου = Όνομα_κλάσης()`

```
>>> C = Client()  
>>> type(C)  
<class '__main__.Client'>  
>>> C  
<__main__.Client object at 0x106a1efe0>
```

```
>>> x = int() # 0  
>>> y = float() # 0.0  
>>> S = str() # ''  
>>> L = list() # []  
>>> T = tuple() # ()  
>>> S = set() # set()  
>>> D = dict() # {}
```

Στην Python όλα είναι αντικείμενα!

Κλάσεις και Αντικείμενα

Κανόνες ονομασίας κλάσεων

- Δε μπορούν να χρησιμοποιηθούν δεσμευμένες λέξεις (λέξεις-κλειδιά):
`class, for, if, else, while...`
- Ξεκινάνε με γράμμα ή κάτω παύλα (`_`)
- Περιέχουν γράμματα, αριθμούς, `_` (όχι τελείες, `*`, `#` κτλ.)
- Όχι κενά
- Διάκριση μεταξύ κεφαλαίων και πεζών (case sensitive)

Σύμβαση:

- Περιγραφικά ονόματα
- Χωρισμός λέξεων με `_`
- Ξεκινάνε με **κεφαλαίο** γράμμα

Κλάσεις και Αντικείμενα

Προσθήκη «ιδιοτήτων»:

Οι ιδιότητες που θα έχουν τα αντικείμενα συνήθως καλούνται **ιδιοχαρακτηριστικά** ή **πεδία**.

```
class Όνομα_κλάσης:  
    όνομα_πεδίου = τιμή_πεδίου  
    ...
```

Πρόσβαση σε πεδίο:

όνομα_αντικειμένου.όνομα_πεδίου

Παράδειγμα:

```
class Client:  
    name = 'Όνομα'  
    surname = 'Επώνυμο'
```

```
>>> C = Client()  
>>> C.name  
'Όνομα'  
>>> C.surname  
'Επώνυμο'  
>>> C.name = 'Δημήτρης'  
>>> C.name  
'Δημήτρης'
```


Κλάσεις και Αντικείμενα

- Η συνάρτηση `isinstance()`:

```
>>> C_1 = Client()
>>> C_2 = Client()
>>> isinstance(C_1, Client) # Ελέγχει αν είναι στιγμιότυπο της κλάσης
True
>>> isinstance(C_2, Client)
True
>>> C_1 is C_2
False
```

```
>>> C_1.name == C_2.name
True
>>> C_1.name is C_2.name
True
>>> C_1.name = 'Δημήτρης'
>>> C_2.name
'Όνομα'
>>> >>> C_1.name is C_2.name
False
```

Κάθε στιγμιότυπο της κλάσης έχει το δικό του **ανεξάρτητο** σύνολο ιδοχαρακτηριστικών...

Κλάσεις και Αντικείμενα

Προσθήκη «συμπεριφορών»:

Τη συμπεριφορά των αντικειμένων την ορίζουν οι **μέθοδοι** τους.

```
class Όνομα_κλάσης:  
    def όνομα_μεθόδου(self, παράμετροι) :  
        ...
```

- Όχι απαραίτητα `self` (συνήθης πρακτική)
- Αναφέρεται στο ίδιο το αντικείμενο (δεν μεταβιβάζουμε τιμή)
- Πάντα η πρώτη παράμετρος
- Είναι το «όνομα» του αντικειμένου

Κλήση μεθόδου:

`όνομα_αντικειμένου.όνομα_μεθόδου(παράμετροι)`

Παράδειγμα:

```
class Client:  
    name = 'Όνομα'  
    surname = 'Επώνυμο'  
  
    def print_client(self):  
        print(self.name, self.surname)
```

```
>>> C = Client()  
>>> C.print_client()  
Όνομα Επώνυμο  
>>> print(C.name, C.surname)  
Όνομα Επώνυμο
```

Κλάσεις και Αντικείμενα

- Αρχικοποίηση πεδίων: `def __init__(self, παράμετροι):`

```
class Client:
    def __init__(self):
        self.name = 'Όνομα'
        self.surname = 'Επώνυμο'
```

```
>>> C = Client() # 1. Δημιουργείται αντικείμενο 2. Καλείται η __init__
>>> C.name
'Όνομα'
```

ή

```
class Client:
    def __init__(self, first = 'Όνομα', last = 'Επώνυμο'):
        self.name = first
        self.surname = last
```

```
>>> C = Client('Δημήτρης') # Δίνουμε τιμές στα ιδιοχαρακτηριστικά
>>> print(C.name, C.surname)
Δημήτρης Επώνυμο
```

Προστασία Ιδιοχαρακτηριστικών

Παράδειγμα:

```
class Client:
    def __init__(self, first = ' ', last = ' ', balance = 0):
        self.name = first
        self.surname = last
        self.balance = balance
```

```
>>> x = Client('Δημήτρης', 'Ζώρος', 1000)
>>> x.balance
1000
>>> x.balance = 100000
>>> x.balance = 'εκατό χιλιάδες' # Μη προβλεπόμενη αλλαγή!
>>> x.balance - 50000
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    x.balance - 50000
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

Ο κώδικας έξω από το αντικείμενο **δεν** είναι απαραίτητο να γνωρίζει την εσωτερική δομή του αντικειμένου (όπως π.χ. την προτιθέμενη μορφή των ιδιοχαρακτηριστικών).

Το μόνο που είναι απαραίτητο να ξέρει είναι η «συμπεριφορά» των αντικειμένων (δηλαδή τις λειτουργίες που επιτελούν οι μέθοδοί του).

Προστασία Ιδιοχαρακτηριστικών

Συνήθως όταν ορίζουμε μία κλάση προσθέτουμε:

- μεθόδους που **τροποποιούν** τα ιδιοχαρακτηριστικά (τροποποιητικές, setters) και
- μεθόδους που μας **παρέχουν πρόσβαση** σε αυτά (getters).

```
class Client:
    def __init__(self, first = ' ', last = ' ', balance = 0):
        self.name = first
        self.surname = last
        self.balance = balance

    # setter:
    def new_balance(self, balance):
        try:
            self.balance = float(balance)
        except ValueError:
            print('Λάθος υπόλοιπο')

    #getter:
    def get_balance(self):
        print('Υπόλοιπο: €'+format(self.balance, '.2f'))
        return self.balance
```

Αλλαγή μόνο με setter σημαίνει αλλαγή **σύμφωνα με τις προδιαγραφές!**

Προστασία Ιδιοχαρακτηριστικών

```
>>> x = Client('Δημήτρης', 'Ζώρος', 1000)
>>> x.new_balance(100000)
>>> x.new_balance('εκατό χιλιάδες')
Λάθος υπόλοιπο
>>> y = x.get_balance()
Υπόλοιπο: €100000.00
>>> y - 50000
50000.0
```

Οι τροποποιητικές μέθοδοι από μόνες τους **δεν αρκούν** για να «προστατέψουμε» τα ιδιοχαρακτηριστικά:

```
>>> x.balance
100000.0
>>> x.balance = 'εκατό χιλιάδες'
>>> y = x.get_balance()
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    y = x.get_balance()
  File "/Users/dzoros/Desktop/test.py", line 16, in get_balance
    print('Υπόλοιπο: €'+format(self.balance, '.2f'))
ValueError: Unknown format code 'f' for object of type 'str'
```

Ακόμα υπάρχει η δυνατότητα να γίνει απευθείας αναφορά στα ιδιοχαρακτηριστικά...

Προστασία Ιδιοχαρακτηριστικών

Για να **απαγορεύσουμε** στον κώδικα **έξω** από την κλάση να έχει πρόσβαση στα ιδιοχαρακτηριστικά, θα πρέπει να τα κάνουμε **ιδιωτικά** (private), ξεκινώντας τα ονόματά τους με δύο κάτω παύλες (__).

```
class Client:
    def __init__(self, first = ' ', last = ' ', balance = 0):
        self.__name = first
        self.__surname = last
        self.__balance = balance

    # setter:
    def new_balance(self, balance):
        try:
            self.__balance = float(balance)
        except ValueError:
            print('Λάθος υπόλοιπο')

    #getter:
    def get_balance(self):
        print('Υπόλοιπο: €'+format(self.__balance, '.2f'))
        return self.__balance
```


Προστασία Ιδιοχαρακτηριστικών

```
>>> x = Client('Δημήτρης', 'Ζώρος', 1000)
>>> x.__balance
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    x.__balance
AttributeError: 'Client' object has no attribute '__balance'. Did
you mean: 'get_balance'?
>>> x.__balance = 100000 #Ορίζουμε μεταβλητή με όνομα x.__balance!
>>> type(x.__balance) #Η x.__balance δεν έχει καμία σχέση με το x!
<class 'int'>
>>> y = x.get_balance()
Υπόλοιπο: €1000.00
>>> x.new_balance(100000)
>>> x.new_balance('εκατό χιλιάδες')
Λάθος υπόλοιπο
>>> y = x.get_balance()
Υπόλοιπο: €100000.00
>>> y - 50000
50000.0
```

Με αυτόν τον τρόπο:

- Η αρχικοποίηση ιδιοχαρακτηριστικών γίνεται με τον **ορισμό** του αντικειμένου (με τη μέθοδο `__init__`).
- Οι αλλαγές πλέον γίνονται **μόνο** μέσω των τροποποιητικών μεθόδων!

Παράδειγμα

```
class Client:
    def __init__(self, first = ' ', last = ' ', balance = 0):
        self.__name = first
        self.__surname = last
        self.__balance = balance

    # setters:
    def deposit(self, amount):
        try:
            self.__balance += float(amount)
        except ValueError:
            print('Λάθος καταχώρηση')

    def withdrawal(self, amount):
        try:
            if self.__balance >= float(amount):
                self.__balance -= float(amount)
            else:
                print('Ανεπαρκές υπόλοιπο')
        except ValueError:
            print('Λάθος καταχώρηση')

    #getter:
    def get_balance(self):
        return self.__balance
```

```
>>> client = Client('Δημήτρης', 'Ζώρος', 300)
>>> client.deposit(400)
>>> client.withdrawal(1000)
Ανεπαρκές υπόλοιπο
>>> print(format(client.get_balance(), '.2f'))
700.00
>>> print(client)
<__main__.Client object at 0x10622ea70>
```

Ενδεχομένως όχι αυτό που θα θέλαμε...

Οργάνωση κλάσεων σε module

Μπορούμε να οργανώσουμε τις κλάσεις μας σε module (όπως κάναμε με τις συναρτήσεις), έτσι ώστε να μπορούμε να τις συμπεριλάβουμε σε οποιοδήποτε πρόγραμμα χρειάζεται αντικείμενα αυτής της μορφής.

(Αποθήκευση ως account.py)

```
class Client:
    def __init__(self, first = ' ', last = ' ', balance = 0):
        self.__name = first
        self.__surname = last
        self.__balance = balance

    def deposit(self, amount):
        try:
            self.__balance += float(amount)
        except ValueError:
            print('Λάθος καταχώρηση')

    def withdrawal(self, amount):
        try:
            if self.__balance >= float(amount):
                self.__balance -= float(amount)
            else:
                print('Ανεπαρκές υπόλοιπο')
        except ValueError:
            print('Λάθος καταχώρηση')

    def get_balance(self):
        return self.__balance
```

Οργάνωση κλάσεων σε module

```
import account # Φόρτιση του module με την κλάση Client

first, last = input('Δώσε Όνομα/Επώνυμο: ').split('/')

savings_account = account.Client(first, last) #όνομα_module.κλάση()

print('\nΕπιλογές:\n1) Κατάθεση\n2) Ανάληψη\n3) Ερώτηση Υπολοίπου\n4) Έξοδος\n')
while True:
    print('-'*60)
    answer = input('Επόμενη ενέργεια: ')
    if answer == '1':
        amount = input('Ποσό κατάθεσης: ')
        savings_account.deposit(amount)

    elif answer == '2':
        amount = input('Ποσό ανάληψης: ')
        savings_account.withdrawal(amount)

    elif answer == '3':
        print('Υπόλοιπο: €'+format(savings_account.get_balance(), '.2f'))

    elif answer == '4':
        break
    else:
        print('Λάθος επιλογή')
```

Παράδειγμα

(Τηλεφωνικός κατάλογος)

(Αποθήκευση ως Contact_class.py)

```
class Contact:
    def __init__(self, name = '', phone = '', email = ''):
        self.__name = name
        self.__phone = phone
        self.__email = email

    # Setters:

    def set_phone(self, phone):
        self.__phone = phone

    def set_email(self, email):
        self.__email = email

    # Getter:

    def get_contact(self):
        return (self.__name, self.__phone, self.__email)
```

Παράδειγμα

(Τηλεφωνικός κατάλογος)

(Αποθήκευση ως contact_functions.py)

```
import Contact_class
# Αποθηκεύουμε τις επαφές σε ένα λεξικό
def add(my_contacts):
    name = input('Όνομα: ')
    if name not in my_contacts:
        phone = input('Τηλέφωνο: ')
        email = input('email: ')
        new_contact = Contact_class.Contact(name, phone, email)
        my_contacts[name] = new_contact
    else:
        print('Υπάρχει ήδη!')

def delete(my_contacts):
    name = input('Όνομα επαφής για διαγραφή: ')
    if name in my_contacts:
        del my_contacts[name]

def find(my_contacts):
    name = input('Όνομα: ')
    print(my_contacts.get(name, 'Δεν βρέθηκε!')) # Δεν θα μας δώσει αυτό
                                                # που θέλουμε...

def change(my_contacts):
    name = input('Όνομα: ')
    if name in my_contacts:
        phone = input('Τηλέφωνο: ')
        email = input('email: ')
        my_contacts[name].set_phone(phone)
        my_contacts[name].set_email(email)
    else:
        print('Δεν βρέθηκε!')
```

Παράδειγμα

(Τηλεφωνικός κατάλογος)

```
import pickle
import contact_functions
# Αποθηκεύουμε το λεξικό σε ένα δυαδικό αρχείο
try:
    f = open('contacts.bin', 'rb')
    D = pickle.load(f)
    f.close()
except FileNotFoundError:
    f = open('contacts.bin', 'wb') # Δημιουργούμε το αρχείο
    f.close()
    D = {}
print('Επιλογές:\n1) Προσθήκη\n2) Διαγραφή\n3) Εύρεση\n4) Αλλαγή\n5) Έξοδος\n')
while True:
    print('-'*60)
    answer = input('Επόμενη ενέργεια: ')
    if answer == '1':
        print('-'*60)
        contact_functions.add(D)
    elif answer == '2':
        print('-'*60)
        contact_functions.delete(D)
    elif answer == '3':
        print('-'*60)
        contact_functions.find(D)
    elif answer == '4':
        print('-'*60)
        contact_functions.change(D)
    elif answer == '5':
        f = open('contacts.bin', 'wb')
        pickle.dump(D, f)
        f.close()
        break
    else:
        print('Λάθος επιλογή')
```

Η Μέθοδος `__str__`

Παράδειγμα:

```
>>> import Contact_class
>>> contact = Contact_class.Contact('Dimitris', '2107276400',
'dzoros@math.uoa.gr')
>>> print(contact)
<Contact_class.Contact object at 0x10bfb94e0>
```

Θα ήταν πιο χρήσιμο να εμφάνιζε π.χ. την τιμή που έχουν τα ιδιοχαρακτηριστικά του αντικειμένου (**κατάσταση αντικειμένου**).

- Μέθοδος ορισμού κατάστασης αντικειμένου:

```
def __str__(self):
    ...
    return συμβολοσειρά
```

όπου η συμβολοσειρά «περιγράφει» την κατάσταση του αντικειμένου.

Η Μέθοδος `__str__`

Παράδειγμα:

```
class Contact:
    def __init__(self, name = '', phone = '', email = ''):
        self.__name = name
        self.__phone = phone
        self.__email = email

    ...

    def __str__(self):
        S_1 = 'Όνομα: '+self.__name+'\n'
        S_2 = 'Τηλ.: '+self.__phone+'\n'
        S_3 = 'Email: '+self.__email
        return S_1+S_2+S_3
```

```
>>> import Contact_class
>>> contact = Contact_class.Contact('Dimitris', '2107276400',
    'dzoros@math.uoa.gr')
>>> print(contact)
Όνομα: Dimitris
Τηλ.: 2107276400
Email: dzoros@math.uoa.gr
>>> str(contact)
'Όνομα: Dimitris\nΤηλ.: 2107276400\nEmail: dzoros@math.uoa.gr'
```


Παράδειγμα

(Σημεία στο επίπεδο)

```
import math

class Point:
    def __init__(self, x=0 , y=0):
        self.__x = x
        self.__y = y

    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y

    def __str__(self):
        return '('+format(self.__x, '.2f')+', '+\
            format(self.__y, '.2f')+')'

    #Δέχονται σαν όρισμα άλλο αντικείμενο της κλάσης Point
    def dist(self, other_point):
        return math.sqrt((self.__x - other_point.get_x())**2+\
            (self.__y - other_point.get_y())**2)

    def is_equal(self, other_point):
        return self.__x == other_point.get_x() and \
            self.__y == other_point.get_y()
```

- Μεταβιβάζουμε «αναφορά» στο αντικείμενο
- Έχουμε πρόσβαση στις μεθόδους και στα ιδιοχαρακτηριστικά του

```
>>> p_1, p_2 = Point(), Point(1,2)
>>> print(p_1)
(0.00,0.00)
>>> p_1.is_equal(p_2)
False
>>> p_1.dist(p_2)
2.23606797749979
```

Παράδειγμα

(Σημεία στο επίπεδο)

```
import Point_class # Το module με την κλάση Point
import random

def main(n):
    points = []
    for i in range(n):
        x = random.random()
        y = random.random()
        points.append(Point_class.Point(x,y)) #Λίστα με αντικείμενα σημεία

    distance, p_1, p_2 = diameter(points)

    print('Μέγιστη απόσταση:', format(distance, '.2f'))
    print('μεταξύ των σημείων', str(p_1), 'και', str(p_2))

def diameter(points):
    n = len(points)
    distance = 0
    for i in range(n-1): # Το τελευταίο το έχουμε ελέγξει ήδη
        for j in range(i+1, n):
            ij_distance = points[i].dist(points[j])
            if ij_distance > distance:
                distance = ij_distance
                p_1 = points[i]
                p_2 = points[j]
    return distance, p_1, p_2

main(100)
```

Υπερφόρτωση Τελεστών

- Έλεγχος ισότητας σημείων στο επίπεδο:

```
>>> p_1, p_2 = Point(), Point()
>>> p_1.is_equal(p_2)
True
>>> p_1 == p_2 # Διαφορετικά αντικείμενα (δεν ελέγχεται η ισότητα των πεδίων τους)
False
```

Αν θέλουμε να ελέγχουμε την ισότητα με τον τελεστή `==`, θα πρέπει να τον **υπερφορτώσουμε**.

Αυτό γίνεται χρησιμοποιώντας στη κλάση `Point` το όνομα `__eq__` αντί για `is_equal`:

```
import math

class Point:

    ...

    def __eq__(self, other_point):
        return self.__x == other_point.get_x() and \
               self.__y == other_point.get_y()
```

```
>>> p_1, p_2 = Point(), Point()
>>> p_1 == p_2 # Καλείται η __eq__ (πρώην is_equal)
True
>>> p_1.__eq__(p_2)
True
```

Υπερφόρτωση Τελεστών

- Πρόσθεση διανυσμάτων (με αρχή (0,0)):

```
>>> p_1, p_2 = Point(1,2), Point(3,4)
>>> p_1 + p_2
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    p_1 + p_2
TypeError: unsupported operand type(s) for +: 'Point' and 'Point'
```

Θα **υπερφορτώσουμε** τον τελεστή +, προσθέτοντας μέθοδο με όνομα `__add__`:

```
import math

class Point:

    ...

    def __add__(self, other_point): # Επιστρέφει σημείο
        return Point(self.__x + other_point.get_x(), \
                      self.__y + other_point.get_y())
```

```
>>> p_1, p_2 = Point(1,2), Point(3,4)
>>> print(p_1 + p_2)
(4.00,6.00)
```

Υπερφόρτωση τελεστή:

Διαφορετική λειτουργία του **ίδιου** τελεστή για **διαφορετικό** τύπο αντικειμένων.

Υπερφόρτωση Τελεστών

Παράδειγμα: (υπερφόρτωσης του +)

```
>>> 1+1
2
>>> [1]+[1]
[1, 1]
>>> '1'+ '1'
'11'
```

Εσωτερικά ονόματα μεθόδων που αντιστοιχούν σε τελεστές:

Αριθμητικοί τελεστές:

Τελεστής	Όνομα μεθόδου
+	__add__
-	__sub__
*	__mul__
/	__truediv__
//	__floordiv__
%	__mod__
**	__pow__

Σχεσιακοί τελεστές:

Τελεστής	Όνομα μεθόδου
==	__eq__
<	__lt__
>	__gt__
<=	__le__
>=	__ge__
!=	__ne__

Λογικοί τελεστές:

Τελεστής	Όνομα μεθόδου
	__or__
&	__and__
^	__xor__

Παράδειγμα

(Ρητοί αριθμοί)

```
import math

class Rational:
    def __init__(self, num = 0, den = 1):
        self.__num = num
        if den == 0:
            print('* Ο παρανομαστής δεν μπορεί να είναι 0,')
            print('τον μετέτρεψα σε 1 *')
            self.__den = 1
        elif den < 0:
            self.__num = -num # Βάζουμε το πρόσημο στον αριθμητή
            self.__den = -den
        else:
            self.__den = den

    def get_num(self):
        return self.__num

    def get_den(self):
        return self.__den

    def __str__(self):
        if self.__den == 1:
            return str(self.__num)
        else:
            return str(self.__num) + '/' + str(self.__den)
```

Παράδειγμα

(Ρητοί αριθμοί)

• • •

```
# Μέθοδος απλοποίησης ρητού
def simplify(self):
    gcd = math.gcd(self.__num, self.__den) #ΜΚΔ των αριθμών
    self.__num //= gcd
    self.__den //= gcd

# Πράξεις ρητών αριθμών
def __add__(self, other):
    num = self.__num * other.get_den() + \
          self.__den * other.get_num()
    den = self.__den * other.get_den()
    result = Rational(num, den)
    result.simplify()
    return result

def __mul__(self, other):
    num = self.__num * other.get_num()
    den = self.__den * other.get_den()
    result = Rational(num, den)
    result.simplify()
    return result

# Έλεγχος ισότητας
def __eq__(self, other):
    self.simplify()
    other.simplify()
    return self.__num == other.get_num() and \
           self.__den == other.get_den()
```

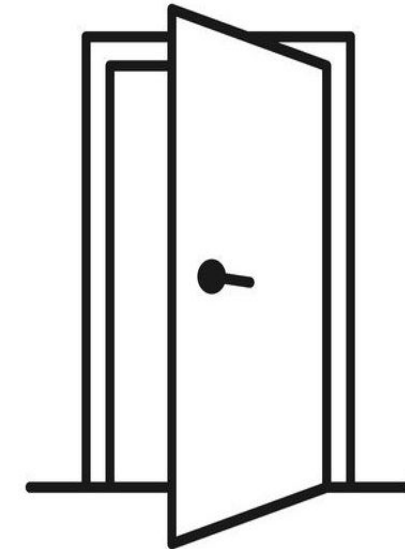
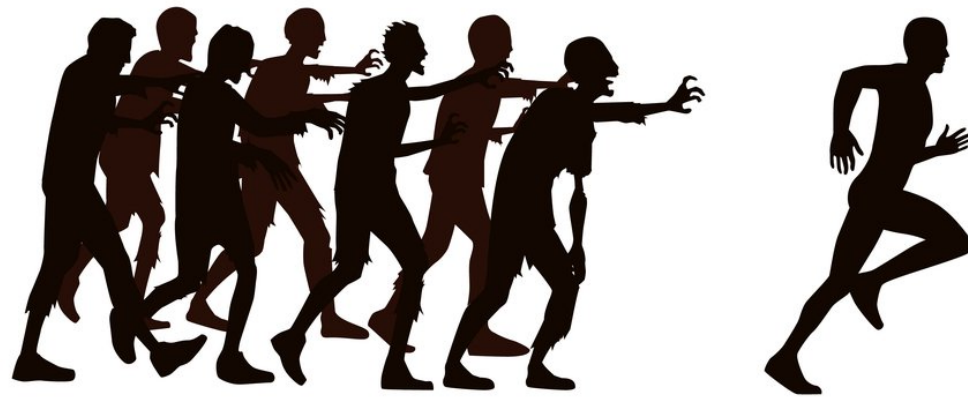

Παράδειγμα

(Ρητοί αριθμοί)

```
>>> x = Rational()
>>> print(x)
0
>>> x = Rational(8)
>>> print(x)
8
>>> x = Rational(8,0)
** Ο παρονομαστής δεν μπορεί να είναι 0,
   τον μετέτρεψα σε 1 **
>>> print(x)
8
>>> x = Rational(8,-4)
>>> print(x)
-8/4
>>> x.simplify() # Δεν επιστρέφει κάτι!
>>> print(x)
-2
>>> y = Rational(4,3)
>>> z = x + y
>>> print(z)
-2/3
>>> z = x * Rational(-5,4)
>>> print(z)
5/2
>>> z == Rational(10,4)
True
```

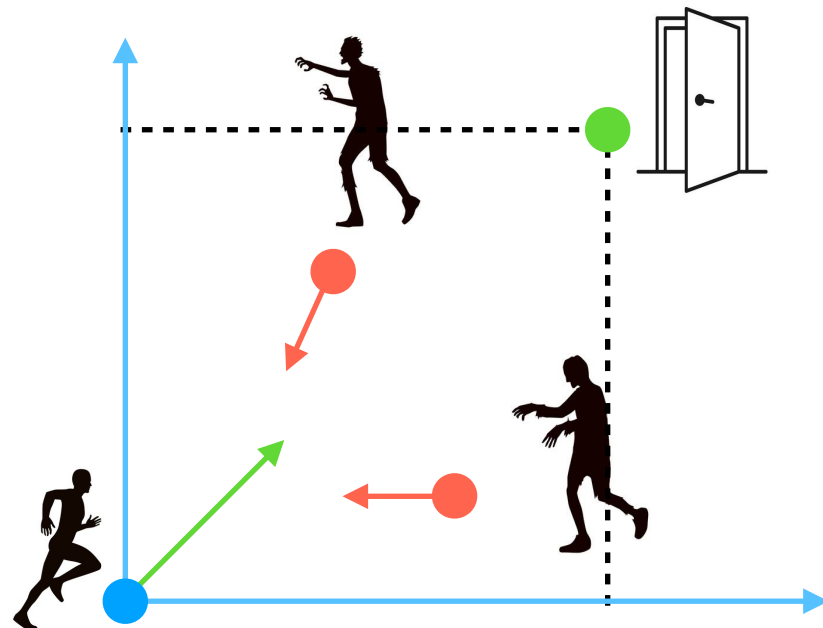

Παράδειγμα

(Κυνηγητό από Ζόμπι)



Κανόνες:

- Ξεκινάμε από το σημείο (0,0).
- Η πόρτα βρίσκεται στο σημείο (10,10).
- Το «δωμάτιο» έχει διαστάσεις 10 m x 10 m
- Τα ζόμπι ξεκινάνε από ένα τυχαίο σημείο (αλλά απέχουν τουλάχιστον 1 μέτρο από εμάς).
- Τρέχουμε με ταχύτητα 3.5 m/sec (προς την πόρτα).
- Κάθε ζόμπι τρέχει κατά πάνω μας με ταχύτητα 0, 1 ή 2 m/sec (τυχαία επιλογή).
- Αν απέχουμε απόσταση λιγότερο από 0,5 m από κάποιο ζόμπι μας έπιασε...



Παράδειγμα

(Κυνηγητό από Ζόμπι)

- Κλάση για τον άνθρωπο:

```
import Point_class
import random

class Human:
    def __init__(self, position = Point_class.Point(0,0), speed = 3.5):
        self.__position = position
        self.__speed = speed

    def get_position(self):
        return self.__position

    def get_speed(self):
        return self.__speed

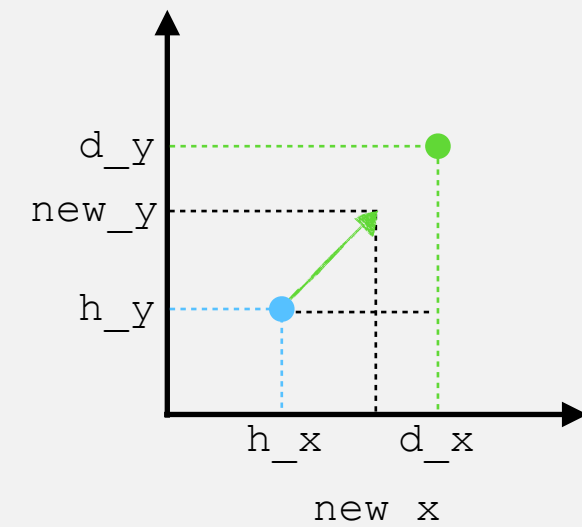
    def dist(self, other):
        return self.__position.dist(other.get_position())

    def move(self, door):
        d_x = door.get_position().get_x()
        d_y = door.get_position().get_y()
        h_x = self.__position.get_x()
        h_y = self.__position.get_y()

        new_x = h_x + (d_x - h_x) * self.__speed/self.dist(door)
        new_y = h_y + (d_y - h_y) * self.__speed/self.dist(door)

        self.__position = Point_class.Point(my_round(new_x), my_round(new_y))

    def saved(self, door):
        return self.__position.get_x() >= door.get_position().get_x() or \
               self.__position.get_y() >= door.get_position().get_y()
```



Παράδειγμα

(Κυνηγητό από Ζόμπι)

- Κλάση για τον άνθρωπο:

```
import Point_class
import random

class Human:
    def __init__(self, position = Point_class.Point(0,0), speed = 3.5):
        self.__position = position
        self.__speed = speed

    def get_position(self):
        return self.__position

    def get_speed(self):
        return self.__speed

    def dist(self, other):
        return self.__position.dist(other.get_position())

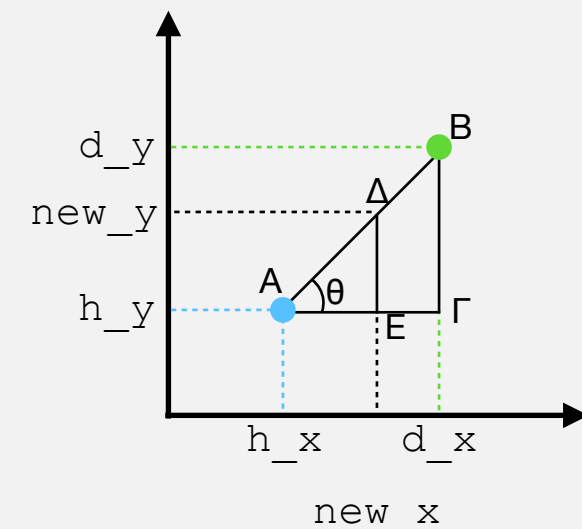
    def move(self, door):
        d_x = door.get_position().get_x()
        d_y = door.get_position().get_y()
        h_x = self.__position.get_x()
        h_y = self.__position.get_y()

        new_x = h_x + (d_x - h_x) * self.__speed/self.dist(door)
        new_y = h_y + (d_y - h_y) * self.__speed/self.dist(door)

        self.__position = Point_class.Point(my_round(new_x), my_round(new_y))

    def saved(self, door):
        return self.__position.get_x() >= door.get_position().get_x() or \
               self.__position.get_y() >= door.get_position().get_y()
```

$$\begin{aligned} \text{συν}_{(AB\Gamma)}\theta &= \text{συν}_{(A\Delta E)}\theta & AB &= \text{self.dist(door)} \\ \eta\mu_{(AB\Gamma)}\theta &= \eta\mu_{(A\Delta E)}\theta & A\Delta &= \text{self.__speed} \end{aligned}$$



Παράδειγμα

(Κυνηγητό από Ζόμπι)

- Κλάση για τα ζόμπι:

```
• • • class Zombie:
    def __init__(self, random_choice = True):
        if random_choice:
            self.__position = Point_class.Point(random.uniform(1,10), \
                                                random.uniform(1,10))
            self.__speed = random.choice([0,1,2])
        else:
            print('Ποια είναι η θέση του ζόμπι;')
            z_x = float(input('x= '))
            z_y = float(input('y= '))
            self.__position = Point_class.Point(z_x,z_y)
            self.__speed = float(input('Ποια είναι η ταχύτητά του; '))

    def get_position(self):
        return self.__position

    def get_speed(self):
        return self.__speed

    def dist(self, other):
        return self.__position.dist(other.get_position())

    def move(self, human):
        h_x = human.get_position().get_x()
        h_y = human.get_position().get_y()
        z_x = self.__position.get_x()
        z_y = self.__position.get_y()

        new_x = z_x + (h_x - z_x) * self.__speed/self.dist(human)
        new_y = z_y + (h_y - z_y) * self.__speed/self.dist(human)

        self.__position = Point_class.Point(my_round(new_x), my_round(new_y))

    def caught(self, human):
        return self.dist(human) <= 0.5
```

Παράδειγμα

(Κυνηγητό από Ζόμπι)

- Κλάση για την πόρτα:

```
• • • class Door:
    def __init__(self, position = Point_class.Point(10,10)):
        self.__position = position

    def get_position(self):
        return self.__position

def my_round(x): # Το δωμάτιο έχει διαστάσεις 10m X 10m
    if x > 10:
        return 10
    elif x < 0:
        return 0
    else:
        return x
```

(Αποθήκευση όλων των κλάσεων ως zombie_chase.py)

A. Ενθυλάκωση: Οι κλάσεις που ορίσαμε περιέχουν τόσο τα χαρακτηριστικά των αντικειμένων όσο και τις βασικές λειτουργίες τους.

B. Αφαίρεση: Έχουμε απλοποιημένες εκδόσεις των αντικειμένων στον φυσικό κόσμο (διατηρούμε μόνο τα χαρακτηριστικά και τις λειτουργίες που χρειαζόμαστε για το πρόβλημά μας).

Παράδειγμα

(Κυνηγητό από Ζόμπι)

```
import zombie_chase

Daryl = zombie_chase.Human()
no = int(input('Πόσα ζόμπι θα κυνηγήσουν τον Daryl; '))
print()
zombies = []
for i in range(no):
    zombies.append(zombie_chase.Zombie())
    print('Θέση '+str(i+1)+'ου ζόμπι:', zombies[i].get_position(), \
          '(ταχύτητα:', str(zombies[i].get_speed()))+')')
door = zombie_chase.Door()
i = 1
flag = False

input('\nΠάτα enter για να αρχίσουμε.\n')

while True:
    print('\n'+str(i)+'sec...')
    i += 1
    Daryl.move(door)
    print('Ο Daryl είναι:', Daryl.get_position())
    if Daryl.saved(door):
        print('\nΟ Daryl σώθηκε!')
        break

    for zombie in zombies:
        zombie.move(Daryl)
        print('Το '+str(zombies.index(zombie)+1)+'ο ζόμπι είναι:', zombie.get_position())
        if zombie.caught(Daryl):
            flag = True
            break

    if flag == True:
        print('\nΤο '+str(zombies.index(zombie)+1)+'ο ζόμπι έφαγε τον Daryl...')
        break
```

Παρατηρήσεις

- Πρόσβαση μεθόδων σε ιδιωτικά χαρακτηριστικά:

```
import math

class Point:
    def __init__(self, x=0 , y=0):
        self.__x = x
        self.__y = y

    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y

    def __str__(self):
        return '('+format(self.__x, '.2f')+','+\
            format(self.__y, '.2f')+')'

    def dist(self, other):
        return math.sqrt((self.__x - other.__x)**2 + (self.__y - other.__y)**2)

    def is_equal(self, other):
        return self.__x == other.__x and self.__y == other.__y
```

Αν other αντικείμενο τύπου Point έχουμε πρόσβαση (δεν χρειαζόμαστε τις get_x , get_y)



```
>>> A = Point(1,2)
>>> B = Point()
>>> A.dist(B)
2.23606797749979
```


Παρατηρήσεις

- Τα ιδιοχαρακτηριστικά δεν χρειάζεται να είναι πάντα ιδιωτικά!

```
import math

class Point:
    def __init__(self, x=0 , y=0):
        self.x = x
        self.y = y

    def __str__(self):
        return '('+format(self.x, '.2f')+','+\
            format(self.y, '.2f')+')'

    def dist(self, other):
        return math.sqrt((self.x - other.x)**2+(self.y - other.y)**2)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __add__(self, other):
        return Point(self.x + other.x,self.y + other.y)
```

```
>>> A = Point(1,2)
>>> B = Point()
>>> A.x, A.y # Απευθείας πρόσβαση στα πεδία
(1, 2)
>>> A = B
>>> A.x, A.y
(0, 0)
>>> A is B # Δύο ονόματα για το αντικείμενο B
True
>>> C = Point()
>>> B is C
False
```


Παρατηρήσεις

- Η μέθοδος `__init__` δεν είναι απαραίτητη!

```
import math

class Point:
    x = 0
    y = 0

    # Θα χρειαστούμε μία τροποποιητική μέθοδο:

    def set_xy(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return '('+format(self.x, '.2f')+','+\
            format(self.y, '.2f')+')'

    def dist(self, other):
        return math.sqrt((self.x - other.x)**2+(self.y - other.y)**2)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)
```

```
>>> A = Point()
>>> print(A)
(0.00,0.00)
>>> A.set_xy(1,2)
>>> A.x, A.y
(1, 2)
```

Παρατηρήσεις

- Ιδιοχαρακτηριστικά VS τοπικές μεταβλητές:

```
class Test:
    a = 'Ιδιοχαρακτηριστικό'
    def method(self):
        b = 'Τοπική μεταβλητή'
        self.c = 'Ιδιοχαρακτηριστικό όταν τρέξει η method!'
    def __init__(self):
        d = 'Τοπική μεταβλητή'
        self.e = 'Ιδιοχαρακτηριστικό'
```

```
>>> obj = Test()
>>> obj.a
'Ιδιοχαρακτηριστικό'
>>> obj.b
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    obj.b
AttributeError: 'Test' object has no attribute 'b'
>>> obj.c
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    obj.c
AttributeError: 'Test' object has no attribute 'c'
>>> obj.d
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    obj.d
AttributeError: 'Test' object has no attribute 'd'
>>> obj.e
'Ιδιοχαρακτηριστικό'
>>> obj.method()
>>> obj.c
'Ιδιοχαρακτηριστικό όταν τρέξει η method!'
```

Κληρονομικότητα και Πολυμορφισμός

- Εξειδίκευση κλάσεων
- Υπερκλάσεις και υποκλάσεις
- Πολλαπλή κληρονομικότητα
- Πολυμορφισμός

Εξειδίκευση κλάσεων

Παράδειγμα:

```
class Car:
    def __init__(self, model, year, mileage, doors, price):
        self.__model = model
        self.__year = year
        self.__mileage = mileage
        self.__doors = doors
        self.__price = price

    def show_info(self):
        print('Μοντέλο:', self.__model)
        print('Έτος:', self.__year)
        print('Χιλιόμετρα:', self.__mileage)
        print('Αρ. Θυρών:', self.__doors)
        print('Τιμή: €'+format(self.__price, '.2f'))
```

```
>>> a = Car('Ford mustang GT', 1967, 100000, 2, 15000)
>>> a.show_info()
Μοντέλο: Ford mustang GT
Έτος: 1967
Χιλιόμετρα: 100000
Αρ. Θυρών: 2
Τιμή: €15000
```

Εξειδίκευση κλάσεων

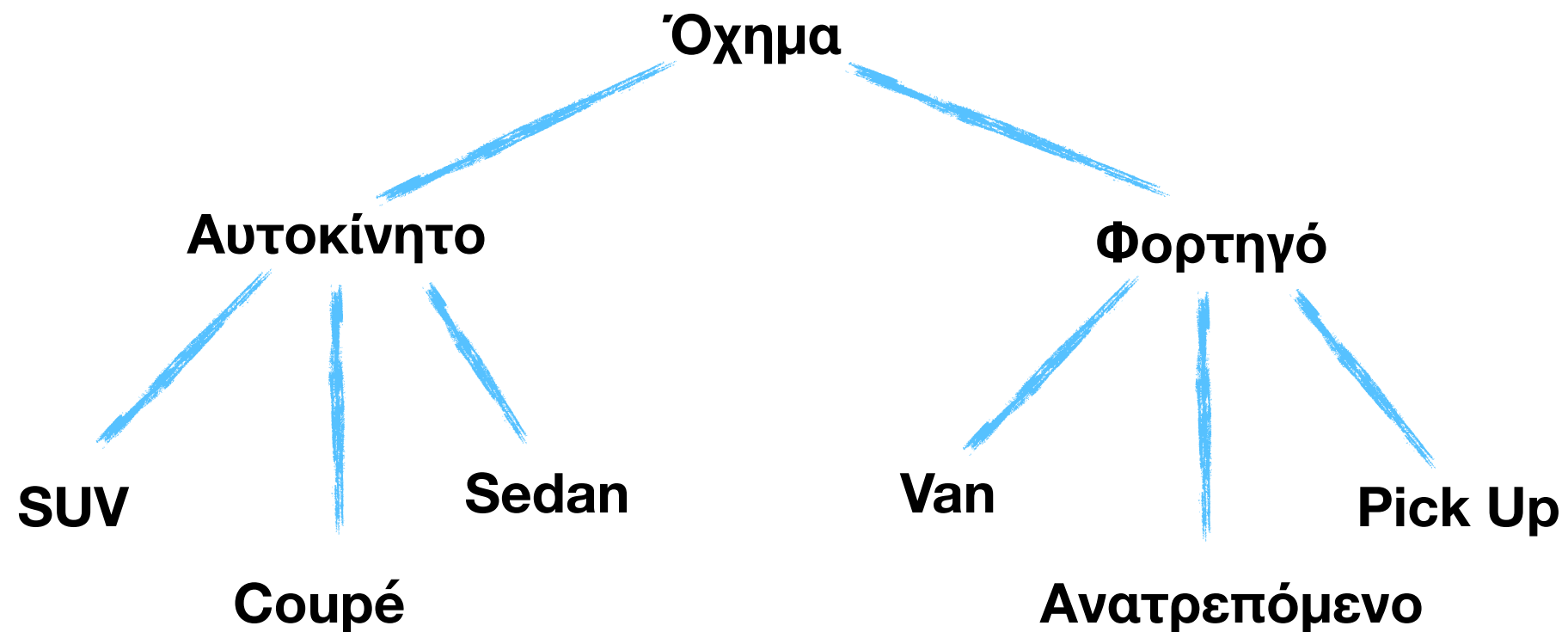
Παράδειγμα:

```
class Truck:
    def __init__(self, model, year, mileage, weight, price):
        self.__model = model
        self.__year = year
        self.__mileage = mileage
        self.__weight = weight
        self.__price = price

    def show_info(self):
        print('Μοντέλο:', self.__model)
        print('Έτος:', self.__year)
        print('Χιλιόμετρα:', self.__mileage)
        print('Βάρος:', self.__weight)
        print('Τιμή: €'+format(self.__price, '.2f'))
```

- Οι δύο κλάσεις έχουν πολλά κοινά χαρακτηριστικά (ιδιοχαρακτηριστικά) και λειτουργίες (μεθόδους).
- Θα ήταν προτιμότερο να δομήσουμε τις κλάσεις που χρειαζόμαστε στο πρόγραμμά μας ως προς κάποια ιεραρχία (αντίστοιχα με τον τρόπο που ιεραρχούνται στην πραγματική ζωή).

Εξειδίκευση κλάσεων



Κάθε κατηγορία έχει όλα τα χαρακτηριστικά και τις λειτουργίες τις παραπάνω κατηγορίας και ίσως μερικά ακόμα.

Γ. Κληρονομικότητα: Μία κλάση **επεκτείνει** (εξειδικεύει) μία άλλη κλάση και **κληρονομεί** τα στοιχεία της (ιδιοχαρακτηριστικά και μεθόδους).

Υπερκλάσεις και υποκλάσεις

- Δημιουργία υποκλάσης μίας κλάσης:

```
class 'Όνομα_υπερκλάσης':  
    ...  
  
class 'Όνομα_υποκλάσης'('Όνομα_υπερκλάσης') :  
    ...
```

Παράδειγμα:

```
class A: # Η υπερκλάση  
    x = 3  
  
    def method(self):  
        print('x =', self.x)  
  
class B(A): # Η υποκλάση  
    y = 3
```

```
>>> b = B()  
>>> isinstance(b,B)  
True  
>>> isinstance(b,A) # Αντικείμενο και της A  
True  
>>> b.x # Το κληρονομεί από την A  
3  
>>> b.method() # Την κληρονομεί από την A  
x = 3  
>>> b.y # Επιπλέον ιδιοχαρακτηριστικό της B  
3
```


Υπερκλάσεις και υποκλάσεις

Παράδειγμα:

```
class Vehicle: # Η υπερκλάση
    def __init__(self, model, year, mileage, price):
        self.__model = model
        self.__year = year
        self.__mileage = mileage
        self.__price = price

    def show_info(self):
        print('Μοντέλο:', self.__model)
        print('Έτος:', self.__year)
        print('Χιλιόμετρα:', self.__mileage)
        print('Τιμή: €'+format(self.__price, '.2f'))

class Car(Vehicle): # Η υποκλάση
    __doors = 4

    def set_doors(self, doors):
        self.__doors = doors

    def show_doors(self):
        print('Αρ. Θυρών:', self.__doors)
```


Υπερκλάσεις και υποκλάσεις

Η κλάση Car ως υποκλάση της Vehicle «κληρονομεί» τα ιδιοχαρακτηριστικά:

`__model`
`__year`
`__mileage`
`__price`

και τις μεθόδους: `__init__`
`show_info`

Έχει ένα επιπλέον ιδιοχαρακτηριστικό το: `__doors`

και δύο επιπλέον μεθόδους: `set_doors`
`show_doors`

```
>>> a = Car('Ford mustang GT', 1967, 100000, 15000)
>>> a.set_doors(2)
>>> a.show_info()
Μοντέλο: Ford mustang GT
Έτος: 1967
Χιλιόμετρα: 100000
Τιμή: €15000.00
>>> a.show_doors()
Αρ. Θυρών: 2
```

Υπερκλάσεις και υποκλάσεις

```
• • • class Car(Vehicle): # Η υποκλάση
    def __init__(self, model, year, mileage, price, doors):
        Vehicle.__init__(self, model, year, mileage, price)
        self.__doors = doors

    def show_info(self):
        Vehicle.show_info(self)
        print('Αρ. Θυρών:', self.__doors)
```

Καλούμε τη μέθοδο της υπερκλάσης με όρισμα το αντικείμενο της υποκλάσης

Η κλάση Car ως υποκλάση της Vehicle «κληρονομεί» τα ιδιοχαρακτηριστικά:

__model
__year
__mileage
__price

και τις μεθόδους: __init__
show_info τις οποίες όμως **υπερκαλύπτει** (τις «επεκτείνει»).

```
>>> a = Car('Ford mustang GT', 1967, 100000, 15000, doors = 2)
>>> a.show_info()
Μοντέλο: Ford mustang GT
Έτος: 1967
Χιλιόμετρα: 100000
Τιμή: €15000.00
Αρ. Θυρών: 2
```

Υπερκάλυψη μεθόδων

- Υπερκλάση και υποκλάση έχουν μεθόδους με το ίδιο όνομα...
- Ένα αντικείμενο της υποκλάσης είναι και αντικείμενο της υπερκλάσης...
- Ποια από τις δύο μεθόδους θα εκτελέσει ο διερμηνέας?!
 - Εκτελείται η μέθοδος της υποκλάσης.

```
>>> a = Car('Ford mustang GT', 1967, 100000, 15000, doors = 2)
>>> b = Vehicle('Citroën 2 CV', 1948, 500000, 2000)
>>> a.show_info() # Της Car
Μοντέλο: Ford mustang GT
Έτος: 1967
Χιλιόμετρα: 100000
Τιμή: €15000
Αρ. Θυρών: 2
>>> b.show_info() # Της Vehicle
Μοντέλο: Citroën 2 CV
Έτος: 1948
Χιλιόμετρα: 500000
Τιμή: €2000
>>> isinstance(a, Vehicle)
True
>>> isinstance(b, Car) # Δεν είναι αντικείμενο της Car...
False
```

Παραδείγματα

(Απογραφή μεταχειρισμένων οχημάτων)

Εκτός από τα χαρακτηριστικά που κληρονομεί, μία υποκλάση μπορεί να έχει **και άλλα** χαρακτηριστικά και λειτουργίες (**περισσότερα $\Rightarrow$ εξειδίκευση**).

```
• • • class Truck(Vehicle):
    def __init__(self, model, year, mileage, price, weight):
        Vehicle.__init__(self, model, year, mileage, price)
        self.__weight = weight

    def show_info(self):
        Vehicle.show_info(self)
        print('Βάρος: kg'+str(self.__weight))

    def print_invoice(self):
        print('Τιμή πώλησης χωρίς ΦΠΑ: €'+format\
              (self.__price / 1.24, '.2f'))
        print('ΦΠΑ: €'+format(self.__price * 0.24, '.2f'))
        print('Τελική τιμή πώλησης: €'+format(self.__price, '.2f'))
```

```
>>> a = Truck('Nissan Navara', 2010, 360000, 9000, weight = 3200)
>>> a.show_info()
Μοντέλο: Nissan Navara
Έτος: 2010
Χιλιόμετρα: 360000
Τιμή: €9000.00
Βάρος: kg3200
```

Παραδείγματα

(Απογραφή μεταχειρισμένων οχημάτων)

Εκτός από τα χαρακτηριστικά που κληρονομεί, μία υποκλάση μπορεί να έχει **και άλλα** χαρακτηριστικά και λειτουργίες (**περισσότερα $\Rightarrow$ εξειδίκευση**).

```
• • • class Truck(Vehicle):
    def __init__(self, model, year, mileage, price, weight):
        Vehicle.__init__(self, model, year, mileage, price)
        self.__weight = weight

    def show_info(self):
        Vehicle.show_info(self)
        print('Βάρος: kg'+str(self.__weight))

    def print_invoice(self):
        print('Τιμή πώλησης χωρίς ΦΠΑ: €'+format\
              (self.__price / 1.24, '.2f'))
        print('ΦΠΑ: €'+format(self.__price * 0.24, '.2f'))
        print('Τελική τιμή πώλησης: €'+format(self.__price, '.2f'))
```

```
>>> a = Truck('Nissan Navara', 2010, 360000, 9000, weight = 3200)
>>> a.print_invoice()
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    a.print_invoice()
  File "/Users/dzoros/Desktop/test.py", line 25, in print_invoice
    print('Τιμή πώλησης χωρίς ΦΠΑ: €'+format(self.__price / 1.24, '.2f'))
AttributeError: 'Truck' object has no attribute '__Truck__price'. Did you
mean: '__Truck__weight'?
```

Γιατί?!

Παραδείγματα

(Απογραφή μεταχειρισμένων οχημάτων)

```
class Vehicle: # Η υπερκλάση
    def __init__(self, model, year, mileage, price):
        self.__model = model
        self.__year = year
        self.__mileage = mileage
        self.__price = price
```



Ιδιωτικά: Ο κώδικας έξω από την κλάση **δεν** έχει πρόσβαση σε αυτά

```
def show_info(self):
    print('Μοντέλο:', self.__model)
    print('Έτος:', self.__year)
    print('Χιλιόμετρα:', self.__mileage)
    print('Τιμή: €'+format(self.__price, '.2f'))
```

```
class Truck(Vehicle):
```

...

Εκτελείται η show_info της Vehicle
(έχει πρόσβαση στα __model, __year κλπ.)

```
def show_info(self):
    Vehicle.show_info(self)
```

print('Βάρος: kg'+str(self.__weight))

Κώδικας έξω από την Vehicle
(δεν έχει πρόσβαση στο __price)

```
def print_invoice(self):
```

```
    print('Τιμή πώλησης χωρίς ΦΠΑ: €'+format\
          (self.__price / 1.24, '.2f'))
```

```
    print('ΦΠΑ: €'+format(self.__price * 0.24, '.2f'))
```

```
    print('Τελική τιμή πώλησης: €'+format(self.__price, '.2f'))
```

Παραδείγματα

(Απογραφή μεταχειρισμένων οχημάτων)

Η κλάση `Truck` ως υποκλάση της `Vehicle` «κληρονομεί» τα ιδιοχαρακτηριστικά:

`__model`
`__year`
`__mileage`
`__price`

καθώς όμως είναι ιδιωτικά **δεν** κληρονομεί πρόσβαση σε αυτά!

Αυτό είναι **καλό** γιατί σε αντίθετη περίπτωση θα μπορούσαμε να τα κάνουμε «δημόσια» απλά ορίζοντας μία υποκλάση:

```
class A:
    __x = 'Ιδιωτικό ιδιοχαρακτηριστικό' ← 'Όχι getter άρα «κρυφό»

class B(A): # Υποκλάση της A
    def make_public(self):
        return self.__x ← Το κάνει «φανερό»!
```

```
>>> b = B()
>>> x = b.make_public()
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    b.make_public()
  File "/Users/dzoros/Desktop/test.py", line 6, in make_public
    return self.__x
AttributeError: 'B' object has no attribute '_B__x'. Did you mean:
'_A__x'?
```


*Διόρθωση

- Πρώτος τρόπος: (Τα ιδιοχαρακτηριστικά **δεν** χρειάζονται «προστασία»)

```
class Vehicle: # Η υπερκλάση
    def __init__(self, model, year, mileage, price):
        self.model = model
        self.year = year
        self.mileage = mileage
        self.price = price

    def show_info(self):
        print('Μοντέλο:', self.model)
        print('Έτος:', self.year)
        print('Χιλιόμετρα:', self.mileage)
        print('Τιμή: €'+format(self.price, '.2f'))

class Truck(Vehicle):
    def __init__(self, model, year, mileage, price, weight):
        Vehicle.__init__(self, model, year, mileage, price)
        self.weight = weight

    def show_info(self):
        Vehicle.show_info(self)
        print('Βάρος: kg'+str(self.weight))

    def print_invoice(self):
        print('Τιμή πώλησης χωρίς ΦΠΑ: €'+format\
              (self.price / 1.24, '.2f'))
        print('ΦΠΑ: €'+format(self.price * 0.24, '.2f'))
        print('Τελική τιμή πώλησης: €'+format(self.price, '.2f'))
```


*Διόρθωση

```
>>> a = Truck('Nissan Navara', 2010, 360000, 9000, weight = 3200)
>>> a.print_invoice()
Τιμή πώλησης χωρίς ΦΠΑ: €7258.06
ΦΠΑ: €2160.00
Τελική τιμή πώλησης: €9000
```

- Δεύτερος τρόπος: (Χρειάζονται «προστασία» αλλά έχουμε **getters**)

```
class Vehicle: # Η υπερκλάση
    def __init__(self, model, year, mileage, price):
        self.__model = model
        self.__year = year
        self.__mileage = mileage
        self.__price = price

    def show_info(self):
        print('Μοντέλο:', self.__model)
        print('Έτος:', self.__year)
        print('Χιλιόμετρα:', self.__mileage)
        print('Τιμή: €'+format(self.__price, '.2f'))

    def get_price(self):
        return self.__price
```

• • •

*Διόρθωση

```
• • • class Truck(Vehicle):
    def __init__(self, model, year, mileage, price, weight):
        Vehicle.__init__(self, model, year, mileage, price)
        self.__weight = weight

    def show_info(self):
        Vehicle.show_info(self)
        print('Βάρος: kg'+str(self.__weight))

    def print_invoice(self):
        print('Τιμή πώλησης χωρίς ΦΠΑ: €'+format\
              (Vehicle.get_price(self) / 1.24, '.2f'))
        print('ΦΠΑ: €'+format(Vehicle.get_price(self) * 0.24, '.2f'))
        print('Τελική τιμή πώλησης: €'+format(Vehicle.get_price(self), \
              '.2f'))
```

```
>>> a = Truck('Nissan Navara', 2010, 360000, 9000, weight = 3200)
>>> a.print_invoice()
Τιμή πώλησης χωρίς ΦΠΑ: €7258.06
ΦΠΑ: €2160.00
Τελική τιμή πώλησης: €9000
```

Καλούμε τη `get_price` από την `Vehicle` με είσοδο το αντικείμενο τύπου `Truck`
(που είναι **και αντικείμενο τύπου** `Vehicle`)

*Διόρθωση

```
• • • class Truck(Vehicle):  
    def __init__(self, model, year, mileage, price, weight):  
        Vehicle.__init__(self, model, year, mileage, price)  
        self.__weight = weight  
  
    def show_info(self):  
        Vehicle.show_info(self)  
        print('Βάρος: kg'+str(self.__weight))  
  
    def print_invoice(self):  
        print('Τιμή πώλησης χωρίς ΦΠΑ: €'+format\  
              (self.get_price() / 1.24, '.2f'))  
        print('ΦΠΑ: €'+format(self.get_price() * 0.24, '.2f'))  
        print('Τελική τιμή πώλησης: €'+format(self.get_price(), '.2f'))
```

```
>>> a = Truck('Nissan Navara', 2010, 360000, 9000, weight = 3200)  
>>> a.print_invoice()  
Τιμή πώλησης χωρίς ΦΠΑ: €7258.06  
ΦΠΑ: €2160.00  
Τελική τιμή πώλησης: €9000
```

Η κλάση Truck **κληρονομεί** τη μέθοδο `get_price` («απευθείας») από την `Vehicle`
(δεν έχουμε υπερκάλυψη)

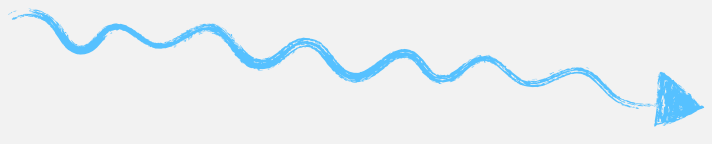
*Διόρθωση

- Τρίτος τρόπος: (Τα ιδιοχαρακτηριστικά **δεν είναι πραγματικά ιδιωτικά...**)

```
• • • class Truck(Vehicle):
    def __init__(self, model, year, mileage, price, weight):
        Vehicle.__init__(self, model, year, mileage, price)
        self.__weight = weight

    def show_info(self):
        Vehicle.show_info(self)
        print('Βάρος: kg'+str(self.__weight))

    def print_invoice(self):
        print('Τιμή πώλησης χωρίς ΦΠΑ: €'+format\
              (self._Vehicle__price / 1.24, '.2f'))
        print('ΦΠΑ: €'+format(self._Vehicle__price * 0.24, '.2f'))
        print('Τελική τιμή πώλησης: €'+format(self._Vehicle__price, '.2f'))
```



```
>>> a = Truck('Nissan Navara', 2010, 360000, 9000, weight = 3200)
>>> a.print_invoice()
Τιμή πώλησης χωρίς ΦΠΑ: €7258.06
ΦΠΑ: €2160.00
Τελική τιμή πώλησης: €9000
>>> a._Truck__weight
3200
>>> a._Vehicle__model # Χρησιμοποιώντας το σωστό όνομα έχουμε πρόσβαση!
'Nissan Navara'
```

Παραδείγματα

(Απογραφή μεταχειρισμένων οχημάτων)

Στα αυτοκίνητα υπάρχει και η δυνατότητα απόσυρσης παλιού οχήματος:

```
class Car(Vehicle):
    def __init__(self, model, year, mileage, price, doors):
        Vehicle.__init__(self, model, year, mileage, price)
        self.__doors = doors

    def show_info(self):
        Vehicle.show_info(self)
        print('Αρ. Θυρών:', self.__doors)

    def print_invoice(self, withdrawal = 0):
        print('Τελική τιμή πώλησης: €'+format(self.get_price()\
                                                -withdrawal, '.2f'))
```

```
>>> a = Car('Ford mustang GT', 1967, 100000, 15000, doors = 2)
>>> a.show_info()
Μοντέλο: Ford mustang GT
Έτος: 1967
Χιλιόμετρα: 100000
Τιμή: €15000
Αρ. Θυρών: 2
>>> a.print_invoice(1300)
Τελική τιμή πώλησης: €13700.00
```

• • •

(Δεύτερη
διόρθωση)

Παραδείγματα

(Απογραφή μεταχειρισμένων οχημάτων)

```
import Vehicles
import pickle

# Τα οχήματα είναι αποθηκευμένα π.χ. σε ένα σύνολο
# Το σύνολο είναι αποθηκευμένη στο αρχείο vehicles.bin

def menu():
    print('\n-- Επιλογές--')
    print('1. Δημιουργία καταλόγου')
    print('2. Καταχώρηση οχήματος')
    print('3. Εμφάνιση οχημάτων')
    print('4. Έξοδος')
    return input('Επόμενη ενέργεια: ')

def create_catalog():
    S = set()
    f = open('vehicles.bin', 'wb')
    pickle.dump(S, f)
    f.close()
```



Παραδείγματα

(Απογραφή μεταχειρισμένων οχημάτων)

```
• • • def new_vehicle():
    f = open('vehicles.bin', 'rb')
    S = pickle.load(f)
    f.close()

    answer = input('Τύπος οχήματος (1. Επιβατικό / 2. Φορτηγό): ')
    model = input('Μοντέλο: ')
    year = int(input('Έτος: '))
    mileage = int(input('Χιλιόμετρα: '))
    price = int(input('Τιμή: '))

    if answer == '1':
        doors = int(input('Αρ. Θυρών: '))
        S.add(Vehicles.Car(model, year, mileage, price, doors))
    else:
        weight = int(input('Βάρος: '))
        S.add(Vehicles.Truck(model, year, mileage, price, weight))

    f = open('vehicles.bin', 'wb')
    pickle.dump(S, f)
    f.close()
```

• • •

Παραδείγματα

(Απογραφή μεταχειρισμένων οχημάτων)

```
• • • def show_vehicles():
    f = open('vehicles.bin', 'rb')
    S = pickle.load(f)
    f.close()
    print('\n'+ '-'*30)
    print('Κατάλογος οχημάτων:')
    print('-'*30)
    for vehicle in S:
        if isinstance(vehicle, Vehicles.Car):
            print('-Επιβατικό όχημα-')
        else:
            print('-Φορτηγό όχημα-')
        vehicle.show_info()
        print('-'*30)

def main():
    while True:
        answer = menu()
        if answer == '1':
            create_catalog()
        elif answer == '2':
            new_vehicle()
        elif answer == '3':
            show_vehicles()
        else:
            break

main()
```


Παραδείγματα

(Κυνηγητό από Ζόμπι)

```
import Point_class # Κλάση για δημιουργία σημείων στο επίπεδο
import random

class Object_in_plane:
    def __init__(self, position, speed):
        self.__position = position
        self.__speed = speed

    def get_position(self):
        return self.__position

    def get_speed(self):
        return self.__speed

    def dist(self, other):
        return self.__position.dist(other.get_position())

    def move(self, other):
        o_x = other.get_position().get_x() # ή other.__position.get_x()
        o_y = other.get_position().get_y() # ή other.__position.get_y()
        s_x = self.__position.get_x()
        s_y = self.__position.get_y()

        new_x = s_x + (o_x - s_x) * self.__speed/self.dist(other)
        new_y = s_y + (o_y - s_y) * self.__speed/self.dist(other)

        self.__position = Point_class.Point(my_round(new_x), \
                                              my_round(new_y))
```



Παραδείγματα

(Κυνηγητό από Ζόμπι)

```
• • • class Human(Object_in_plane):
    def __init__(self, position = Point_class.Point(0,0), speed = 3.5):
        Object_in_plane.__init__(self, position, speed)

    def saved(self, door):
        return self.get_position().get_x() >= door.get_position().get_x() or\
            self.get_position().get_y() >= door.get_position().get_y()

class Zombie(Object_in_plane):
    def __init__(self, random_choice = True):
        if random_choice:
            position = Point_class.Point(random.uniform(1,10),\
                                          random.uniform(1,10))
            speed = random.choice([0,1,2])
            Object_in_plane.__init__(self, position, speed)
        else:
            print('Ποιά είναι η θέση του ζόμπι;')
            z_x = float(input('x= '))
            z_y = float(input('y= '))
            position = Point_class.Point(z_x,z_y)
            speed = float(input('Ποιά είναι η ταχύτητά του; '))
            Object_in_plane.__init__(self, position, speed)

    def caught(self, human):
        return self.dist(human) <= 0.5
```

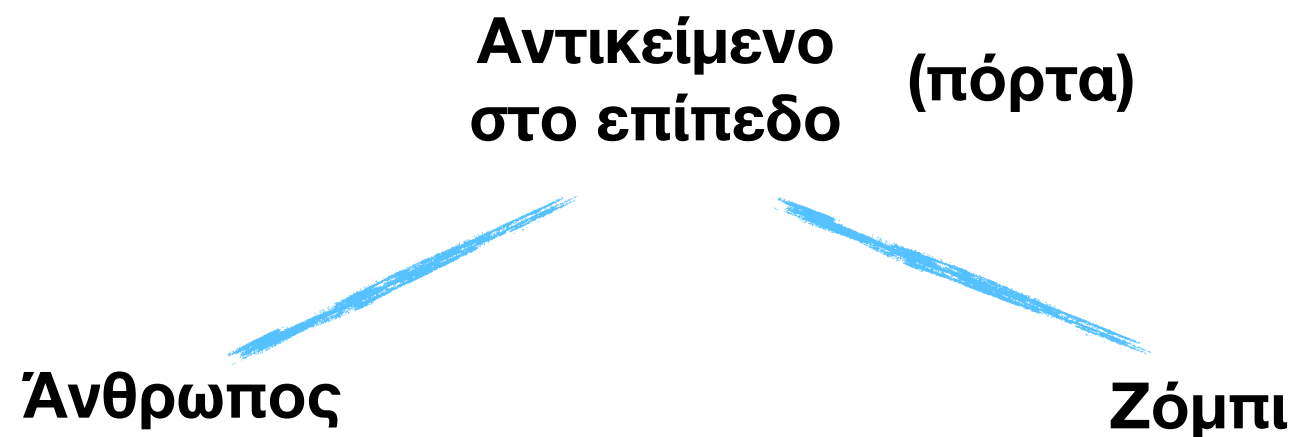
Παραδείγματα

(Κυνηγητό από Ζόμπι)

• • •

```
# Στην πραγματικότητα δεν χρειαζόμαστε ξεχωριστή κλάση για την πόρτα  
  
# Το δωμάτιο έχει διαστάσεις 10m X 10m:  
def my_round(x):  
    if x > 10:  
        return 10  
    elif x < 0:  
        return 0  
    else:  
        return x
```

(Αποθήκευση των κλάσεων ως zombie_chase.py)



Παραδείγματα

(Κυνηγητό από Ζόμπι)

```
import zombie_chase
import Point_class

Daryl = zombie_chase.Human()
no = int(input('Πόσα ζόμπι θα κυνηγήσουν τον Daryl; '))
print()
zombies = []
for i in range(no):
    zombies.append(zombie_chase.Zombie())
    print('Θέση '+str(i+1)+'ου ζόμπι:', zombies[i].get_position(), \
          '(ταχύτητα:', str(zombies[i].get_speed()))+')')

door_position = Point_class.Point(10,10)
door = zombie_chase.Object_in_plane(door_position, 0) # Δεν έχουμε ξεχωριστή κλάση για
# την πόρτα

i = 1
flag = False
input('\nΠάτα enter για να αρχίσουμε.')
while True:
    print('\n'+str(i)+'sec...')
    i += 1
    Daryl.move(door)
    print('O Daryl είναι:', Daryl.get_position())
    if Daryl.saved(door):
        print('\nO Daryl σώθηκε!')
        break
    for zombie in zombies:
        zombie.move(Daryl)
        print('To '+str(zombies.index(zombie)+1)+'ο ζόμπι είναι:', zombie.get_position())
        if zombie.caught(Daryl):
            flag = True
            break
    if flag == True:
        print('\nTo '+str(zombies.index(zombie)+1)+'ο ζόμπι έφαγε τον Daryl...')
        break
```

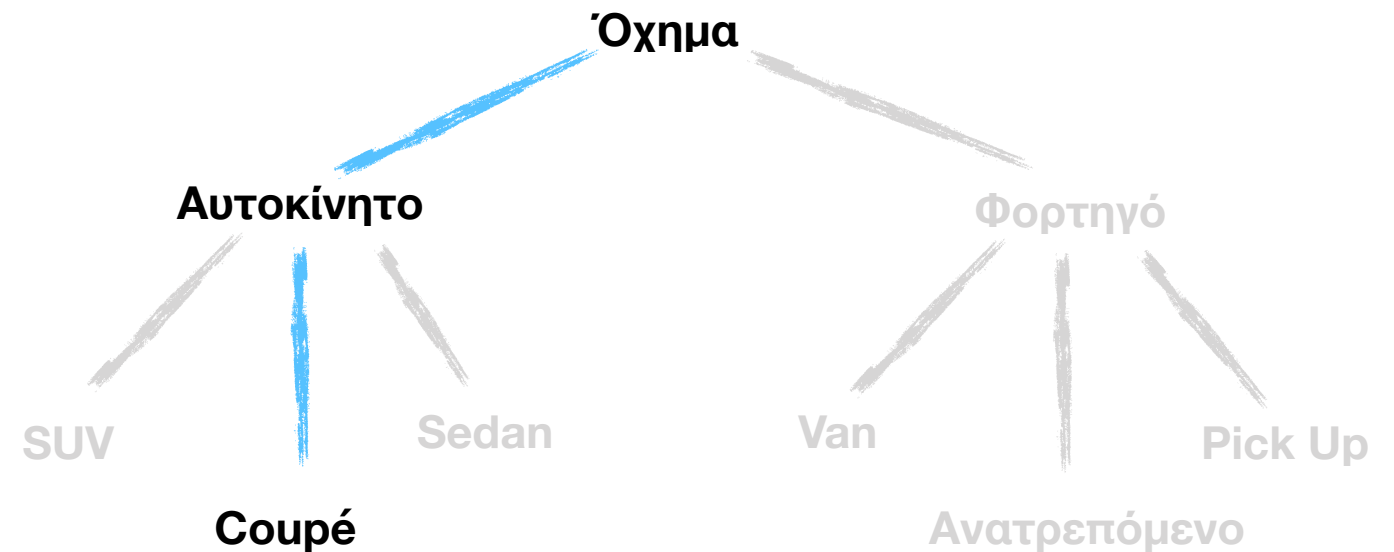
Πολλαπλή κληρονομικότητα

«Πολυεπίπεδη» κληρονομικότητα:

Κλάση A

Κλάση B

Κλάση Γ



```
• • • class Coupe(Car):  
    # Έχουν δύο πόρτες...  
    def __init__(self, model, year, mileage, price):  
        Car.__init__(self, model, year, mileage, price, 2)
```

```
>>> a = Coupe('Ford mustang GT', 1967, 100000, 15000)  
>>> isinstance(a, Car)  
True  
>>> isinstance(a, Vehicle)  
True
```

Πολλαπλή κληρονομικότητα

«Πολλαπλή» κληρονομικότητα:



```
class Animal():
    def __init__(self, number_of_legs):
        self.__legs = number_of_legs

    def info(self):
        print('Έχει', self.__legs, 'πόδια')

class Flying_object():
    def __init__(self, altitude):
        self.__altitude = altitude

    def info(self):
        print('Πετάει μέχρι τα', self.__altitude, 'πόδια')
```

Πολλαπλή κληρονομικότητα

```
• • • class Bird(Animal, Flying_object):  
    def __init__(self, number_of_legs, altitude):  
        Animal.__init__(self, number_of_legs)  
        Flying_object.__init__(self, altitude)  
  
    def info(self):  
        Animal.info(self)  
        Flying_object.info(self)
```

```
>>> eagle = Bird(2, 10000)  
>>> isinstance(eagle, Animal)  
True  
>>> isinstance(eagle, Flying_object)  
True  
>>> eagle.info()  
Έχει 2 πόδια  
Πετάει μέχρι τα 10000 πόδια  
>>> # «Όλα είναι αντικείμενα» και ανήκουν στην κλάση object:  
>>> isinstance(eagle, object)  
True  
>>> isinstance(11, object)  
True  
>>> isinstance('11', object)  
True
```


Πολυμορφισμός

Δ. Πολυμορφισμός: Αλλαγή της «συμπεριφοράς» ενός στοιχείου **ανάλογα** με το αντικείμενο που χρησιμοποιούμε.

- Σε «δικές μας» συναρτήσεις :

```
def mult(L):  
    result = 1  
    for element in L:  
        result *= element  
    return result
```

```
>>> mult([1,2,3]) # Λίστα  
6  
>>> mult((1,2,3)) # Πλειάδα  
6  
>>> mult({1,2,3}) # Σύνολο  
6  
>>> mult({1:'', 2:'', 3:''}) # Λεξικό  
6  
>>> mult(range(1,4)) # Αλληλουχία  
6
```


Παραδείγματα πολυμορφισμού

- Στις συναρτήσεις `len`, `min`/`max`, `sum`, `sorted` κ.λπ.:

```
>>> len('Python')
6
>>> len([1, 2, 3])
3
>>> min('Python')
'P'
>>> min([1, 2, 3])
1
>>> sum([1, 2, 3])
6
>>> sum({1: 'a', 2: 'b', 3: 'c'})
6
>>> sorted('Python')
['P', 'h', 'n', 'o', 't', 'y']
>>> sorted({1: 'a', 2: 'b', 3: 'c'})
[1, 2, 3]
```

Παραδείγματα πολυμορφισμού

- Στη δομή επανάληψης `for`:

```
for i in [1,2,3]:  
    for j in '123':  
        for k in {1:'a', 2:'b', 3:'c'}:  
            print(i, j, k, sep = ' ', end = ' ')  
        print()  
    print()
```

```
111 112 113  
121 122 123  
131 132 133
```

```
211 212 213  
221 222 223  
231 232 233
```

```
311 312 313  
321 322 323  
331 332 333
```

Παραδείγματα πολυμορφισμού

- Σε υπερκαλυμμένες μεθόδους:

```
class A:
    def method(self, other):
        print('Καλημέρα')

class B(A): # Υποκλάση της A
    def method(self, other):
        print('Καληνύχτα')

class C(B): # Υποκλάση της B (και της A)
    def method(self, other):
        other.method(self)
```

```
>>> a = A()
>>> b = B()
>>> c = C()
>>> a.method(b)
Καλημέρα
>>> a.method(a)
Καλημέρα
>>> b.method(a)
Καληνύχτα
>>> c.method(a)
Καλημέρα
>>> c.method(b)
Καληνύχτα
```

Παραδείγματα πολυμορφισμού

```
class Athlete:
    def __init__(self, name, surname, birthday):
        self.name = name
        self.surname = surname
        self.birthday = birthday
    def info(self):
        print(self.name, self.surname, f'({self.birthday})')
    def record(self):
        print('-No record-')

class Sprinter(Athlete):
    def __init__(self, name, surname, birthday, time):
        Athlete.__init__(self, name, surname, birthday)
        self.time = time
    def record(self):
        print('Χρόνος 100 m:', self.time, 'sec')

class Boxer(Athlete):
    def __init__(self, name, surname, birthday, wins, losses, draws):
        Athlete.__init__(self, name, surname, birthday)
        self.wins = wins
        self.losses = losses
        self.draws = draws
    def record(self):
        print('Νίκες:', self.wins)
        print('Ήττες:', self.losses)
        print('Ισοπαλίες:', self.draws)

class Footballer(Athlete):
    def __init__(self, name, surname, birthday, goals):
        Athlete.__init__(self, name, surname, birthday)
        self.goals = goals
    def record(self):
        print('Αριθμός γκολ:', self.goals)
```

Παραδείγματα πολυμορφισμού

```
import Athlete_classes

a = Athlete_classes.Athlete('Γιάννης', 'Αγιάνης', 1770)
b = Athlete_classes.Sprinter('Usain', 'Bolt', 1986, 9.58)
c = Athlete_classes.Boxer('Muhammad', 'Ali', 1942, 56, 37, 5)
d = Athlete_classes.Footballer('Diego Armando', 'Maradona', 1960, 354)

for i in a,b,c,d:
    print('-'*30)
    i.info()
    i.record()
print('-'*30)
```

```
-----
Γιάννης Αγιάνης (1770)
-No record-
-----
Usain Bolt (1986)
Χρόνος 100 m: 9.58 sec
-----
Muhammad Ali (1942)
Νίκες: 56
Ήττες: 37
Ισοπαλίες: 5
-----
Diego Armando Maradona (1960)
Αριθμός γκολ: 354
-----
```