Panellinies Online

ΠΑΝΕΛΛΑΔΙΚΕΣ ΕΠΑ.Λ.

ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ · ΜΑΘΗΜΑ 1

Προγραμματισμός Υπολογιστών

Γ΄ ΕΠΑ.Λ. — Σημειώσεις μαθήματος

Θεωρία, ορισμοί και λυμένα παραδείγματα σε Python

ΠΕΡΙΕΧΕΙ

7	28	Γ΄
κεφάλαια	ενότητες	τάξη ΕΠΑ.Λ.

Περιεχόμενα

Ενότητα 1: Βασικές Έννοιες

1.1 Μεταβλητές και τύποι δεδομένων

Τι είναι οι τύποι δεδομένων;

Κάθε πρόγραμμα επεξεργάζεται δεδομένα — αριθμούς, κείμενο, λογικές τιμές. Ο **τύπος δεδομένων** καθορίζει πως αποθηκεύεται και επεξεργάζεται μια τιμή στη μνήμη. Ιδιαίτερο χαρακτηριστικό της Python: **δεν χρειάζεται να δηλώσουμε τον τύπο** — η γλώσσα τον αναγνωρίζει αυτόματα από την τιμή που δίνουμε.

Βασικοί τύποι δεδομένων

Ακέραιοι (int)
Αριθμοί χωρίς δεκαδικό μέρος, π.χ. `19`, `-5`, `1024`.

Κινητής υποδιαστολής (float)
Αριθμοί με δεκαδικό μέρος, π.χ. `3.14`, `28.2E-5`. Το δεκαδικό γράφεται με **τελεία** (.), όχι κόμμα.

Λογικός (bool)
Δύο τιμές: `True` (Αληθής) και `False` (Ψευδής). Χρησιμοποιείται σε ελέγχους συνθηκών.

Συμβολοσειρές (str)
Ακολουθία χαρακτήρων. Γράφονται με μονά (`'...'`) ή διπλά (`"..."`) εισαγωγικά. Υποστηρίζουν Unicode (ελληνικά, αγγλικά).

Παραδείγματα στην Python

python 2.7

```
# Ακέραιος (int)
x = 19
print type(x)  # <type 'int'>
# Κινητής υποδιαστολής (float)
y = 3.14
z = 28.2E-5    # ίσο με 0.000282
print type(y)  # <type 'float'>
# Λογικός τύπος (bool)
flag = True
print type(flag) # <type 'bool'>
# Συμβολοσειρά (str)
name = "Καλημέρα"
code = '221051445'
print type(name) # <type 'str'>
```

Σύνοψη

Η Python αναγνωρίζει αυτόματα τον τύπο δεδομένων — **δεν δηλώνουμε τύπο**.
Βασικοί τύποι: `int`, `float`, `bool`, `str`.
Το δεκαδικό μέρος γράφεται με **τελεία** (.), όχι κόμμα.
Η συνάρτηση `type()` επιστρέφει τον τύπο μιας τιμής ή μεταβλητής.

1.2 Αριθμητικές και λογικές πράξεις και εκφράσεις

Αριθμητικοί τελεστές

Οι **τελεστές** είναι σύμβολα που χρησιμοποιούμε για να δημιουργούμε εκφράσεις. Οι βασικοί αριθμητικοί τελεστές στην Python είναι:

+

Πρόσθεση

-

Αφαίρεση

*

Πολλαπλασιασμός

/

Διαίρεση

**

Ύψωση σε δύναμη

%

Υπόλοιπο διαίρεσης (mod)

Προτεραιότητα πράξεων

Σε κάθε αριθμητική έκφραση ακολουθείται συγκεκριμένη σειρά εκτέλεσης:

1. **Παρενθέσεις** `( )` — εκτελούνται πρώτες
2. **Ύψωση σε δύναμη** `**`
3. **Πολλαπλασιασμός, Διαίρεση, Υπόλοιπο** `* / %`
4. **Πρόσθεση, Αφαίρεση** `+ -`

Παράδειγμα: `(2 + 3) * 5` → πρώτα η παρένθεση (5), μετά `5 * 5 = 25`. Χωρίς παρένθεση, `2 + 3 * 5` → πρώτα ο πολλαπλασιασμός: `2 + 15 = 17`.

Σχεσιακοί (συγκριτικοί) τελεστές

Χρησιμοποιούνται για σύγκριση δύο τιμών. Το αποτέλεσμα είναι πάντα True ή False.

<

Μικρότερο από

<=

Μικρότερο ή ίσο

>

Μεγαλύτερο από

>=

Μεγαλύτερο ή ίσο

==

Ίσο με

!=

Διάφορο από

Λογικοί τελεστές

Οι λογικοί τελεστές συνδυάζουν λογικές τιμές (True/False). Σειρά προτεραιότητας: **not** → **and** → **or**.

not

Άρνηση (ΟΧΙ)

and

Σύζευξη (ΚΑΙ)

or

Διάζευξη (Ή)

Πίνακας αλήθειας λογικών τελεστών

P	Q	P and Q	P or Q	not P
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Παραδείγματα εκφράσεων

python 2.7

```
# Αριθμητικές πράξεις
print 2 + 8 * 2      # 18  (πρώτα ο πολ/σμός)
print 45 / 10       # 4 (ακέραια διαίρεση)
print 45.0 / 10     # 4.5 (αν float, βγάζει float)
print 45 % 10       # 5 (υπόλοιπο διαίρεσης)
print 2 ** 3        # 8 (2 στην 3η)

# Σχεσιακοί τελεστές
print 23 == 23    # True
print 34 != 45    # True
print 56 <= 12   # False

# Λογικοί τελεστές
print True and True   # True
print True and False  # False
print True or False   # True
print not(56 <= 12) # True
```

Μεταβλητές — Εκχώρηση & κανόνες

Στην Python **δεν δηλώνουμε** μεταβλητές — απλά τους εκχωρούμε τιμή με =. Το = **δεν είναι** μαθηματική ισότητα, αλλά τελεστής εκχώρησης: βάζει τη τιμή του δεξιού μέρους στη μεταβλητή του αριστερού.

Σωστά ονόματα

onoma1, onoma_2, timi, mesi_timi, embado

Λάθος ονόματα

1o_onoma (ξεκινά με αριθμό), print (δεσμευμένη λέξη)

Μια μεταβλητή μπορεί να αλλάζει τύπο κατά τη διάρκεια του προγράμματος:

```
python 2.7

a = 125             # int
print type(a)    # <type 'int'>

a = 250.7           # float (άλλαξε τύπος!)
print type(a)    # <type 'float'>

a = True             # bool
print type(a)    # <type 'bool'>

a = "Καλημέρα"      # str
print type(a)    # <type 'str'>
```

Εισαγωγή & εξαγωγή δεδομένων

`print`
Εμφανίζει τιμές στην οθόνη. Μπορεί να δεχτεί μεταβλητή, αριθμό, συμβολοσειρά ή έκφραση.

print

name

print

1045.34

print

message

*

3

`input()` **&** `raw_input()`
Διαβάζουν τιμή από το πληκτρολόγιο. Η `input()` επιστρέφει αριθμό, η `raw_input()` επιστρέφει πάντα συμβολοσειρά.

num = input("Δώσε αριθμό: ")
name = raw_input("Όνομα: ")

Σχόλια

Τα σχόλια ξεκινούν με # και αγνοούνται από τον διερμηνευτή. Βοηθούν στην κατανόηση του κώδικα.

```
python 2.7

# Πρόγραμμα πολλαπλασιασμού δύο αριθμών
x = input("Δώσε τον πρώτο αριθμό: ")
y = input("Δώσε τον δεύτερο αριθμό: ")
ginomeno = x * y  # πολλαπλασιάζει και αποθηκεύει
print ginomeno      # εμφανίζει το αποτέλεσμα
```

Σύνοψη

Αριθμητικοί τελεστές: `+ - * / ** %` — προτεραιότητα: **( ) → ** → * / % → + -**
Σχεσιακοί τελεστές: `< <= > >= == !=` — αποτέλεσμα πάντα True ή False
Λογικοί τελεστές: `not and or` — σειρά: **not → and → or**
Το = είναι **εκχώρηση**, όχι ισότητα — βάζει τιμή στη μεταβλητή
Εισαγωγή: `input()` (αριθμός) / `raw_input()` (συμβολοσειρά) — Εξαγωγή: `print`

1.3 Βασικές (ενσωματωμένες) συναρτήσεις

Συναρτήσεις μετατροπής τύπων

Η Python προσφέρει έτοιμες συναρτήσεις για μετατροπή δεδομένων από έναν τύπο σε άλλον:

int()

Μετατρέπει μια τιμή σε ακέραιο. Κόβει τα δεκαδικά ψηφία.

float()

Μετατρέπει ακεραίους ή συμβολοσειρές σε δεκαδικό.

str()

Μετατρέπει οποιαδήποτε τιμή σε συμβολοσειρά.

Χρήσιμες μαθηματικές συναρτήσεις

abs()

Επιστρέφει την απόλυτη τιμή ενός αριθμού.

pow(a, b)

Επιστρέφει το a υψωμένο στη δύναμη b.

divmod(x, y)

Επιστρέφει (πηλίκο, υπόλοιπο) της x / y.

Παραδείγματα

python 2.7

```
# Μετατροπές τύπων
print float(10)       # 10.0
print int(5.678)     # 5 (κόβει τα δεκαδικά)
print str(5.678)     # "5.678" (string)

# Μαθηματικές
print abs(-45)       # 45
print divmod(10, 3)  # (3, 1) — πηλίκο, υπόλοιπο
print pow(2, 3)       # 8

# Ανάγνωση ακέραιου με input()
a = int(input("Δώσε ακέραιο: "))
```

Εξωτερικές βιβλιοθήκες

Εκτός από τις ενσωματωμένες, υπάρχουν βιβλιοθήκες (modules) με επιπλέον συναρτήσεις. Για να τις χρησιμοποιήσουμε, τις εισάγουμε με την εντολή `import`. Η πρόσβαση γίνεται με **dot notation**: `module.όνομα_συνάρτησης()`.

math — μαθηματικές συναρτήσεις
riza = math.sqrt(2) # 1.4142
x = math.pi # 3.14159...

Εισαγωγή με import
import math

import random

import turtle

Σύνοψη

Μετατροπή τύπων: `int()`, `float()`, `str()`
Μαθηματικές: `abs()`, `pow()`, `divmod()`
Για ακέραια είσοδο: `int(input("..."))`
Εξωτερικές βιβλιοθήκες: `import math` και dot notation `math.sqrt()`

1.4 Δομή προγράμματος και καλές πρακτικές

Καλές πρακτικές προγραμματισμού

Για να γράφουμε καθαρό, ευανάγνωστο και συντηρήσιμο κώδικα, ακολουθούμε ορισμένες βασικές συμβάσεις:

Τίτλος & σχόλια
Δώστε τίτλο στην αρχή με # και προσθέστε επεξηγηματικά σχόλια — βοηθούν εσάς και άλλους να καταλάβετε τον κώδικα.

Συνεπή εισαγωγικά
Χρησιμοποιείτε είτε μονά (`'...'`) είτε διπλά (`"..."`) εισαγωγικά, αλλά ποτέ ανάμικτα.

Ονόματα με νόημα
Τα ονόματα μεταβλητών να σχετίζονται με τη χρήση τους: `mesi_timi` αντί για `x`.

Σωστή στοίχιση
Η Python βασίζεται στα κενά για να ορίσει ομάδες εντολών — προσοχή στις εσοχές!

```
python 2.7

# Πρόγραμμα: Υπολογισμός μέσης τιμής
# Δημιουργός: ...

grade1 = 8
grade2 = 9
mesi_timi = (grade1 + grade2) / 2
print "Μέση τιμή:", mesi_timi
```

Σύνοψη

Χρησιμοποιούμε σχόλια (#) για τίτλο και επεξηγήσεις
Συνεπή εισαγωγικά — όχι ανάμικτα μονά/διπλά
Ονόματα μεταβλητών περιγραφικά· προσοχή στις εσοχές
Τα σχόλια αγνοούνται από τον διερμηνευτή

1.5 Τύποι και δομές δεδομένων στις γλώσσες προγραμματισμού

Τύποι δεδομένων — γενική θεώρηση

Κάθε γλώσσα προγραμματισμού ορίζει **τύπους δεδομένων** ως σύνολα τιμών και πράξεων πάνω σε αυτές. Διακρίνονται σε **πρωτογενείς** (προκαθορισμένους) και **μη πρωτογενείς** (ορίζονται από τον προγραμματιστή).

Απλοί (πρωτογενείς) τύποι

Μία μεταβλητή — μία τιμή.

ΑΚΕΡΑΙΟΣ

int

=

42

ΠΡΑΓΜΑΤΙΚΟΣ

float

=

3.14

ΛΟΓΙΚΟΣ

True

/

False

ΑΛΦΑΡΙΘΜΗΤΙΚΟΣ

string

=

"κείμενο"

Σύνθετοι (μη πρωτογενείς)

Πολλές τιμές — ονομάζονται **δομές δεδομένων**.

ΛΙΣΤΑ

List — διατεταγμένη συλλογή

ΣΤΟΙΒΑ

Stack — LIFO

ΟΥΡΑ

Queue — FIFO

Παράδειγμα κώδικα

```python
# Απλός τύπος: μία τιμή
age = 18          # int
name = "Άννα"      # str
is_student = True  # bool

# Σύνθετος τύπος: πολλές τιμές
grades = [8, 9, 7]   # λίστα (list)
```

Σύνοψη

Απλοί τύποι: ακέραιος, πραγματικός, λογικός, αλφαριθμητικός — μία τιμή
Σύνθετοι τύποι (δομές δεδομένων): λίστα, στοίβα, ουρά — πολλές τιμές
Οι σύνθετοι αποτελούνται από πρωτογενείς ή/και άλλους σύνθετους τύπους

Ενότητα 2: Αλγοριθμικές δομές

2.1.1 Δομή ακολουθίας

Τι είναι η δομή ακολουθίας;

Πρόκειται για μια **σειρά από εντολές** που εκτελούνται η μία μετά την άλλη, με τη σειρά που έχουν γραφεί. Είναι η απλούστερη αλγοριθμική δομή και χρησιμοποιείται όταν:

Η σειρά των βημάτων είναι **καθορισμένη**

Όλα τα βήματα εκτελούνται **πάντοτε**

Δεν υπάρχουν **εξαιρέσεις**

Παράδειγμα: Εμβαδόν ορθογωνίου

Να γραφεί πρόγραμμα που να υπολογίζει το εμβαδόν ενός ορθογωνίου παραλληλογράμμου, διαβάζοντας τη βάση και το ύψος από τον χρήστη.

```
python 2.7

# Πρόγραμμα υπολογισμού εμβαδού ορθογωνίου
# Διάβασμα δεδομένων
base = float(input("Δώσε τη βάση: "))
height = float(input("Δώσε το ύψος: "))

# Υπολογισμός
embado = base * height

# Εμφάνιση αποτελέσματος
print "Το εμβαδόν του ορθογωνίου είναι:", embado
```

Οι εντολές εκτελούνται αυστηρά με τη σειρά: είσοδος → επεξεργασία → έξοδος. Κάθε βήμα είναι υποχρεωτικό και η σειρά δεν αλλάζει.

Σύνοψη

Εντολές εκτελούνται **σειριακά**, μία μετά την άλλη
Η σειρά είναι δεδομένη, όλες οι εντολές εκτελούνται πάντα
Κατάλληλη για απλά προβλήματα χωρίς εξαιρέσεις ή συνθήκες

2.1.2 Δομή επιλογής if (ΑΝ)

Η δομή επιλογής

Η δομή επιλογής `if` χρησιμοποιείται όταν θέλουμε να εκτελεστεί μια ομάδα εντολών **μόνο εφόσον** ισχύει μια συγκεκριμένη συνθήκη. Έτσι δημιουργούμε λογικά μονοπάτια στον αλγόριθμο.

Απλή if (Αν...τότε)

Αν η συνθήκη είναι True, εκτελούνται οι εντολές που περιέχονται στη δομή. Αλλιώς, η ροή προσπερνά τη δομή.

```
python 2.7

# Έλεγχος αρνητικού ποσού
a = float(input("Δώσε ένα ποσό: "))
if a < 0:
    print "ΠΡΟΣΟΧΗ: το ποσό είναι αρνητικό"
```

If-else (Αν...αλλιώς)

Η δομή `if-else` επιτρέπει δύο διακλαδώσεις: μία για όταν η συνθήκη είναι αληθής και μία για όταν είναι ψευδής.

```
python 2.7

if συνθήκη:
    # εντολές αν αληθής
else:
    # εντολές αν ψευδής
```

Σημείωση: Οι ομάδες εντολών ορίζονται ως **μπλοκ** με τη χρήση εσοχής (κενά). Οι διαδοχικές εντολές με την ίδια εσοχή ανήκουν στο ίδιο μπλοκ. Οι εσοχές μπαίνουν αυτόματα με το πάτημα `Enter` μετά το σύμβολο `:`.

```
python 2.7

# Σύγκριση ηλικιών
a = int(input("Δώσε την ηλικία A: "))
b = int(input("Δώσε την ηλικία B: "))

if a > b:
    print "Η ηλικία A είναι μεγαλύτερη από τη B"
else:
    print "Η ηλικία A είναι μικρότερη ή ίση (<=) της ηλικίας B"
```

Πολλαπλή επιλογή με elif

Για περισσότερες από δύο περιπτώσεις, χρησιμοποιούμε την εντολή `elif`:

```
python 2.7

if συνθήκη1:
    # εντολές
elif συνθήκη2:
    # εντολές_2
else:
    # εντολές_3
```

```python
# Πρόγραμμα εντοπισμού ζώνης συναγερμού
alarm = int(input("Δώσε κωδικό συναγερμού: "))

if alarm > 20:
    z = 3
elif alarm > 10:
    z = 2
else:
    z = 1

print "Πρόβλημα από τη ζώνη:", z
print "τέλος"
```

Εμφωλευμένες δομές επιλογής (nested if)

Οι πολλαπλές επιλογές μπορούν να υλοποιηθούν και με if μέσα σε if:

```python
# Έλεγχος δείκτη υπεριώδους ακτινοβολίας (UV)
uv = float(input("Δώσε δείκτη UV: "))

if uv <= 15:
    if uv >= 11:
        print "Ακραία κατάσταση κινδύνου"
    else:
        if uv >= 6:
            print "Μεγάλος ή Πολύ μεγάλος κίνδυνος"
        else:
            if uv >= 0:
                print "Ελάχιστος ή μικρός κίνδυνος"
else:
    print "Ο αριθμός υπερβαίνει το 15. Μη αποδεκτή τιμή!"
```

Σύνοψη

`if`: εκτέλεση αν η συνθήκη είναι True
`if-else`: δύο διακλαδώσεις — True ή False
`elif`: πολλαπλές συνθήκες σε αλυσίδα
Εσοχές (κενά) ορίζουν τα μπλοκ εντολών
Οι **εμφωλευμένες if** επιτρέπουν ιεραρχικούς ελέγχους

2.1.3 Δομή επανάληψης (for και while)

Εισαγωγή στις επαναλήψεις

Συχνά σε ένα πρόγραμμα, μια ομάδα εντολών πρέπει να εκτελείται **περισσότερες από μία φορές**. Υπάρχουν δύο τύποι επαναλήψεων:

Προκαθορισμένες

Το πλήθος των επαναλήψεων είναι **γνωστό** από πριν. Π.χ. υπολογισμός μέσου όρου 22 μαθητών.

Μη προκαθορισμένες

Το πλήθος καθορίζεται **κατά την εκτέλεση**. Π.χ. υποβολή αιτήσεων έως ότου συμπληρωθεί θέση.

Στην Python χρησιμοποιούμε την εντολή `for` για προκαθορισμένες επαναλήψεις και την `while` για μη προκαθορισμένες.

Η εντολή for

Η `for` εκτελεί ένα τμήμα κώδικα για καθορισμένο αριθμό επαναλήψεων, χρησιμοποιώντας τη συνάρτηση `range()`.

```
python 2.7

for μεταβλητή in range(αρχή, μέχρι, βήμα):
    # εντολές
```

Η συνάρτηση range()

Ενσωματωμένη συνάρτηση που παράγει ακολουθίες ακεραίων. Σύνταξη: `range(αρχή, μέχρι, βήμα)`. Οι παράμετροι **αρχή** και **βήμα** είναι προαιρετικές — προεπιλογή: `αρχή=0`, `βήμα=1`. Η παράμετρος **μέχρι** είναι υποχρεωτική.

range(10)

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

range(1, 8)

[1,2,3,4,5,6,7]

range(8, -1, -1)

[8,7,6,5,4,3,2,1,0]

```
python 2.7

# Άθροισμα περιττών αριθμών από 1 έως 100
athroisma = 0

for i in range(1, 101, 2):
    athroisma = athroisma + i

print "Το άθροισμα των περιττών είναι:", athroisma
```

Η εντολή while

Η `while` (όσο...επανάλαβε) χρησιμοποιείται για μη προκαθορισμένο αριθμό επαναλήψεων. Σε κάθε επανάληψη, ελέγχεται η συνθήκη **πριν** εκτελεστούν οι εντολές.

```
python 2.7

μεταβλητή = αρχική_τιμή
while συνθήκη:
    # εντολές
    # ενημέρωση μεταβλητής
```

Προσοχή: Πρέπει να υπάρχει μέσα στο μπλοκ εντολή που να εξασφαλίζει ότι κάποτε η συνθήκη θα γίνει False — αλλιώς ο βρόχος δεν τερματίζει ποτέ!

Πριν από το βρόχο while, πρέπει να δώσουμε αρχική τιμή στη μεταβλητή ελέγχου.

```
python 2.7

# Απαρίθμηση από 40 έως 50
x = 40        # αρχική τιμή
while x < 50:    # έλεγχος συνθήκης
    x = x + 1    # αύξηση μετρητή
    print x

print "Τελική τιμή x:", x
```

Έλεγχος εγκυρότητας δεδομένων με while

Μία συνήθης εφαρμογή του while είναι ο έλεγχος των δεδομένων εισόδου — ζητάμε από τον χρήστη να δώσει τιμή μέχρι να λάβουμε έγκυρη:

```
python 2.7

# Έλεγχος εισαγωγής βαθμού (0-20)
choice = int(input("Δώσε βαθμό (0-20): "))

while choice < 0 or choice > 20:
    choice = int(input("Παρακαλώ δώστε έγκυρη τιμή (0-20): "))

print "Εγκυρος βαθμός:", choice
```

Παιχνίδι εικασίας αριθμού

```
python 2.7

import random

thenum = random.randint(1, 10)
print "Μάντεψε τον αριθμό που διάλεξα (1-10)"

guess = 0
while guess != thenum:
    guess = int(input("Δώσε αριθμό: "))
    if guess > thenum:
        print "Έδωσες μεγαλύτερο αριθμό"
    elif guess < thenum:
        print "Έδωσες μικρότερο αριθμό"
    else:
        print "Τον βρήκες!"
```

Εμφωλευμένες δομές επανάληψης

Συνδυασμός for και while: πρόκριση αθλητών στο μήκος με 3 προσπάθειες ο καθένας:

```python 2.7
# Πρόκριση αθλητών στο μήκος – καλύτερη επίδοση
import random

max_alma = 0
count_athletes = 0

for i in range(1, 21):
    max_epidosi = 0
    prospatheia = 1

    while prospatheia <= 3 and max_epidosi < 4.5:
        print "Αγωνίζεται ο {}ος αθλητής, {}η προσπάθεια".format(i, prospatheia)
        epidosi = float(input("Δώσε επίδοση: "))

        if max_epidosi < epidosi:
            max_epidosi = epidosi

        if epidosi >= 4.5:
            print "Ο {}ος αθλητής προκρίθηκε με {} μέτρα".format(i, epidosi)
            count_athletes = count_athletes + 1

        prospatheia = prospatheia + 1

    if max_epidosi < 4.5:
        print "Δεν προκρίθηκε. Καλύτερο άλμα: {}".format(max_epidosi)

    if max_alma < max_epidosi:
        max_alma = max_epidosi

print "Προκρίθηκαν {} αθλητές".format(count_athletes)
print "Καλύτερη επίδοση: {} μέτρα".format(max_alma)
```

Σύνοψη

`for`: προκαθορισμένες επαναλήψεις με `range()`
`range(αρχή, μέχρι, βήμα)` — αρχή και βήμα προαιρετικά
`while`: επανάληψη όσο η συνθήκη είναι True
Απαιτείται **αρχικοποίηση** μεταβλητής πριν το while και **ενημέρωση** μέσα στο βρόχο
Οι βρόχοι μπορούν να **εμφωλευτούν** (π.χ. for μέσα σε while)

2.2.1 Δημιουργώντας δικές μας συναρτήσεις

Τι είναι μια συνάρτηση;

Οι **συναρτήσεις** είναι επαναχρησιμοποιήσιμα μέρη προγραμμάτων. Μας επιτρέπουν να δίνουμε ένα όνομα σε ένα σύνολο εντολών και να το εκτελούμε καλώντας το όνομα αυτό, από οπουδήποτε στο πρόγραμμα και όσες φορές θέλουμε — διαδικασία που ονομάζεται **κλήση** (calling) της συνάρτησης.

Ορισμός συνάρτησης με def

Για να ορίσουμε μια δική μας συνάρτηση χρησιμοποιούμε τη λέξη-κλειδί `def`, ακολουθεί ένα όνομα που ταυτοποιεί τη συνάρτηση και ένα ζευγάρι παρενθέσεων που μπορούν να περικλείουν ονόματα μεταβλητών (παραμέτρους). Η γραμμή τελειώνει με άνω και κάτω τελεία `:`.

```python 2.7
def όνομα_συνάρτησης():
    # εντολές
    return αποτέλεσμα  # προαιρετικά
```

```python 2.7
# Ορισμός συνάρτησης
def hello():
    print "Γεια σου κόσμε!"
# Τέλος της συνάρτησης

# Κλήση της συνάρτησης
hello()
hello()  # κι άλλη μία κλήση
```

Κλήση συνάρτησης από άλλη συνάρτηση

Μια συνάρτηση μπορεί να καλεί άλλη συνάρτηση. Δημιουργούμε την `epanalave_hello()` που καλεί την `hello()` δύο φορές:

```python 2.7
def hello():
    print "Γεια σου κόσμε!"

def epanalave_hello():
    hello()
    hello()

def epanalave_4fores():
    epanalave_hello()
    epanalave_hello()

# Κλήση – θα εμφανίσει 4 φορές το μήνυμα
epanalave_4fores()
```

Σύνοψη

Συναρτήσεις: επαναχρησιμοποιήσιμα τμήματα κώδικα
Ορισμός με `def` + όνομα + `()` + `:`
Κλήση: `όνομα_συνάρτησης()`
Μια συνάρτηση μπορεί να καλεί άλλες συναρτήσεις

2.2.2 Παράμετροι συναρτήσεων

Παράμετροι και ορίσματα

Μια συνάρτηση δέχεται δεδομένα μέσω των **παραμέτρων** και επιστρέφει αποτελέσματα μέσω της εντολής `return`. Οι παράμετροι καθορίζονται μέσα στο ζευγάρι των παρενθέσεων στον ορισμό της συνάρτησης και διαχωρίζονται με κόμμα. Όταν καλούμε τη συνάρτηση, δίνουμε και τις τιμές με τον ίδιο τρόπο — αυτές οι τιμές ονομάζονται **ορίσματα** (arguments).

Κάποιες συναρτήσεις δεν απαιτούν ορίσματα (π.χ. `math.pi`), ενώ άλλες απαιτούν ένα ή περισσότερα (π.χ. `math.pow(x, y)` χρειάζεται δύο ορίσματα: βάση και εκθέτη).

```python 2.7
def όνομα(παράμετρος1, παράμετρος2, ...):
    # εντολές
    return αποτέλεσμα
```

```python 2.7
# Συνάρτηση με δύο παραμέτρους
def ginomeno(a, b):
    x = a * b
    return x

# Κλήση με ορίσματα 5 και 10
print ginomeno(5, 10)  # 50
```

Παράμετρος
Μεταβλητή που ορίζεται στην παρένθεση της def. Λειτουργεί ως «θέση» για την τιμή που θα δοθεί.

Όρισμα
Η πραγματική τιμή που περνάμε στη συνάρτηση όταν την καλούμε.

```python 2.7
# Συνάρτηση με επιστροφή αποτελέσματος
def embadon_orthogoniou(mikos, platos):
    return mikos * platos

# Χρήση της συνάρτησης
result = embadon_orthogoniou(5, 3)
print "Εμβαδόν:", result  # Εμβαδόν: 15
```

Περισσότερα για τις συναρτήσεις και την εμβέλεια των παραμέτρων θα μελετήσουμε στο Κεφάλαιο 7 (Αντικειμενοστρεφής Προγραμματισμός).

Σύνοψη
Παράμετροι: μεταβλητές στον ορισμό της συνάρτησης
Ορίσματα: πραγματικές τιμές κατά την κλήση
return: επιστρέφει τιμή από τη συνάρτηση
Μια συνάρτηση μπορεί να έχει πολλές παραμέτρους

Ενότητα 3: Κλασικοί Αλγόριθμοι II

3.1 Δυαδική αναζήτηση

Από τη σειριακή στη δυαδική αναζήτηση

Στην προηγούμενη τάξη λύσαμε το πρόβλημα της αναζήτησης με τον αλγόριθμο της **Σειριακής αναζήτησης**. Στη χειρότερη περίπτωση (το στοιχείο δεν υπάρχει ή είναι τελευταίο), ελέγχονται όλα τα στοιχεία. Υπάρχει όμως ταχύτερος τρόπος, αν εκμεταλλευτούμε τη **διάταξη** των δεδομένων.

Παιχνίδι: Σκέψου έναν αριθμό από το 1 έως το 1000. Είναι μεγαλύτερος ή μικρότερος από το 500; Με μία ερώτηση κόβουμε τον χώρο αναζήτησης στο μισό! Σε κάθε βήμα το πρόβλημα περιορίζεται στο μισό του προηγούμενου — από 1000 → 500 → 250 → 125 → ... χρειάζονται μόλις **10 ερωτήσεις** για 1000 αριθμούς.

Δυαδική αναζήτηση σε λίστα

Ο αλγόριθμος εφαρμόζεται και σε λίστες με διατεταγμένα στοιχεία (αριθμούς ή αλφαριθμητικά). Η λογική είναι ίδια με το παιχνίδι: σε κάθε βήμα συγκρίνουμε το ζητούμενο με το μεσαίο στοιχείο και περιορίζουμε το διάστημα στο μισό.

Θέση:

0

1

2

3

4

5

6

7

8

9

10

11

12

13

Τιμή:

5 10 17 23 28 30 35 40 45 50 60 63 68 70

Αναζήτηση του 45: mid=35 (θέση 6) → 45>35 → first=7 → mid=50 (θέση 10) → 45<50 → last=9 → mid=45 (θέση 8) → βρέθηκε!

Αλγόριθμος δυαδικής αναζήτησης

Η συνάρτηση επιστρέφει True αν το στοιχείο υπάρχει, False διαφορετικά:

```python
# python 2.7

def binarySearch(array, key):
    first = 0
    last = len(array) - 1
    found = False

    while first <= last and not found:
        mid = (first + last) // 2
        if array[mid] == key:
            found = True
        elif array[mid] < key:
            first = mid + 1
        else:
            last = mid - 1

    return found
```

Η επόμενη έκδοση επιστρέφει τη **θέση** του στοιχείου ή -1 αν δεν υπάρχει:

```python
# python 2.7

def binarySearch(array, key):
    first = 0
    last = len(array) - 1
    pos = -1

    while first <= last and pos == -1:
        mid = (first + last) // 2
        if array[mid] == key:
            pos = mid
        elif array[mid] < key:
            first = mid + 1
        else:
            last = mid - 1

    return pos
```

Οι αλγόριθμοι λειτουργούν για κάθε τύπο δεδομένων που υποστηρίζει τους τελεστές == και < (ακέραιοι, πραγματικοί, αλφαριθμητικά κ.λπ.). Αυτό το χαρακτηριστικό ονομάζεται **πολυμορφισμός** και είναι βασικό πλεονέκτημα της Python.

Σύνοψη

Η **δυαδική αναζήτηση** εκμεταλλεύεται τη διάταξη των δεδομένων
Σε κάθε βήμα ο χώρος αναζήτησης **μειώνεται στο μισό**
Ισχύει **μόνο** για ταξινομημένες συλλογές δεδομένων
Πολύ ταχύτερη από τη σειριακή αναζήτηση
Ισχύει για αριθμούς, αλφαριθμητικά και άλλους συγκρίσιμους τύπους

3.2 Ταξινόμηση Ευθείας ανταλλαγής (Bubble Sort)

Εισαγωγή

Στην προηγούμενη τάξη παρουσιάστηκε ο αλγόριθμος **ταξινόμησης με επιλογή** (selection sort). Ένα χαρακτηριστικό του είναι ότι εκτελεί πάντα τον ίδιο αριθμό συγκρίσεων, ακόμα και για ήδη ταξινομημένες λίστες. Ο αλγόριθμος **ταξινόμησης ευθείας ανταλλαγής** (bubble sort) έχει το πλεονέκτημα ότι μπορεί να τερματίσει νωρίτερα αν η λίστα είναι ήδη ταξινομημένη.

Εισαγωγική δραστηριότητα

Πέντε μαθητές είναι στη σειρά: **Δημήτρης, Ευγενία, Ναταλία, Ρένια, Αλέξανδρος**. Ο Αλέξανδρος ήρθε τελευταίος, αλλά αλφαβητικά είναι πρώτος. Κοιτάει κάθε φορά τον μπροστινό του και αν είναι μικρότερος, αλλάζουν θέσεις — συνεχίζει μέχρι να βρεθεί στη σωστή θέση:

Αρχική:

Δημήτρης

→

Ευγενία

→

Ναταλία

→

Ρένια

→

Αλέξανδρος

Βήμα 1:

Δημήτρης

→

Ευγενία

→

Ναταλία

→

Αλέξανδρος

→

Ρένια

(ανταλλαγή με Ρένια)

Βήμα 2:

Δημήτρης

→

Ευγενία

→

Αλέξανδρος

→

Ναταλία

→

Ρένια

(ανταλλαγή με Ναταλία)

Βήμα 3:

Δημήτρης

→

Αλέξανδρος

→

Ευγενία

→

Ναταλία

→

Ρένια

(ανταλλαγή με Ευγενία)

Βήμα 4:

Αλέξανδρος

→

Δημήτρης

→

Ευγενία

→

Ναταλία

→

Ρένια

(ανταλλαγή με Δημήτρη)

```python 2.7
students = ["Δημήτρης", "Ευγενία", "Ναταλία", "Ρένια", "Αλέξανδρος"]

# Μία προς μία συγκρίσεις και ανταλλαγές
for j in range(4, 0, -1):
    if students[j] < students[j-1]:
        students[j], students[j-1] = students[j-1], students[j]

print students  # ['Αλέξανδρος', 'Δημήτρης', 'Ευγενία', 'Ναταλία', 'Ρένια']
```

Αντιμετάθεση τιμών

Για να ανταλλάξουμε δύο μεταβλητές:

Με προσωρινή μεταβλητή

```python 2.7
temp = a
a = b
b = temp
```

Χωρίς επιπλέον μεταβλητή

```python 2.7
a, b = b, a
```

Ο αλγόριθμος bubble sort

Γενικεύοντας για λίστα A με N αριθμούς, κάθε πέρασμα φέρνει το μικρότερο στοιχείο στην αρχή. Χρειάζονται N-1 περάσματα:

```python 2.7
# Αλγόριθμος ταξινόμησης ευθείας ανταλλαγής (bubble sort)
def bubbleSort(A):
    N = len(A)
    for i in range(N - 1):
        for j in range(N - 1, i, -1):
            if A[j] < A[j - 1]:
                A[j], A[j - 1] = A[j - 1], A[j]
    return A
```

Για λίστα `[21, 13, 8, 5, 3, 2]` η εκτέλεση βήμα-βήμα:

Αρχικά: [21, 13, 8, 5, 3, 2]
i=0: [2, 21, 13, 8, 5, 3] — το 2 ανεβαίνει στη θέση 0
i=1: [2, 3, 21, 13, 8, 5] — το 3 στη θέση 1
i=2: [2, 3, 5, 21, 13, 8] — το 5 στη θέση 2
i=3: [2, 3, 5, 8, 21, 13] — το 8 στη θέση 3
i=4: [2, 3, 5, 8, 13, 21] — ταξινομημένο!

Βελτιστοποίηση

Ο αλγόριθμος μπορεί να σταματήσει νωρίτερα αν διαπιστωθεί ότι η λίστα είναι ήδη ταξινομημένη, με μια λογική μεταβλητή `swapped`:

```python 2.7
def bubbleSortOptimized(A):
    N = len(A)
    swapped = True
    i = 0

    while swapped:
        swapped = False
        for j in range(N - 1, i, -1):
            if A[j] < A[j - 1]:
                A[j], A[j - 1] = A[j - 1], A[j]
                swapped = True
        i = i + 1

    return A
```

Για παράδειγμα, στη λίστα `[3, 5, 8, 13, 21, 34, 55, 2]`, αρκεί **μόνο ένα πέρασμα** για να έρθει το 2 στην πρώτη θέση, αφού τα υπόλοιπα είναι ήδη στη σωστή σειρά.

Σύνοψη

Συγκρίνει διαδοχικά ζεύγη και ανταλλάσσει όταν δεν είναι στη σωστή σειρά
Σε κάθε πέρασμα το μικρότερο στοιχείο «ανεβαίνει» σαν φυσαλίδα
Χρειάζονται **N-1** περάσματα για N στοιχεία
Βελτιστοποίηση: τερματισμός αν δεν γίνει καμία ανταλλαγή

Ενότητα 4: Διαχείριση Αρχείων

4.1 Εισαγωγή — Δημιουργία, άνοιγμα, κλείσιμο αρχείων

Γιατί αρχεία;

Η δυνατότητα να δημιουργούμε, να διαβάζουμε και να γράφουμε σε αρχεία αποτελεί βασική εργασία σε πολλά προγράμματα και παρέχεται από όλες τις γλώσσες προγραμματισμού. Τα περισσότερα προγράμματα που έχουμε δει μέχρι τώρα μπορούν να χαρακτηριστούν ως **προσωρινά** — τρέχουν για ένα μικρό χρονικό διάστημα και παράγουν κάποια έξοδο, με τα δεδομένα τους να χάνονται. Αυτό συμβαίνει, διότι τα δεδομένα ήταν αποθηκευμένα προσωρινά στην κύρια μνήμη του υπολογιστή, οπότε διαρκούσαν μόνο κατά την εκτέλεση του προγράμματος.

Σε πολλές όμως περιπτώσεις θέλουμε τα δεδομένα να μη χάνονται. Θέλουμε να διαβάζουμε δεδομένα από ένα αρχείο του υπολογιστή στο οποίο βρίσκονται αποθηκευμένα και να γράφουμε ένα αποτέλεσμα στο ίδιο ή εναλλακτικά σε άλλο αρχείο. Αυτή η διεργασία ανάγνωσης και εγγραφής ονομάζεται **Είσοδος/Έξοδος Αρχείου** και στην Python υλοποιείται μέσω ενσωματωμένων συναρτήσεων.

Η συνάρτηση `open()`

Για να χρησιμοποιήσουμε ένα αρχείο, πρέπει πρώτα να το **ανοίξουμε** με την ενσωματωμένη συνάρτηση `open()` και στο τέλος να το **κλείσουμε** με τη συνάρτηση `close()`.

Η συνάρτηση `open()` είναι ενσωματωμένη στην Python — δε χρειάζεται να φορτώσουμε κάποια βιβλιοθήκη. Η σύνταξή της είναι:

```
python 2.7

open("όνομα_αρχείου", "τρόπος_προσπέλασης")
```

Η `open()` δέχεται δύο ορίσματα: Το πρώτο είναι το **όνομα του αρχείου**, με το οποίο το αναγνωρίζει το λειτουργικό σύστημα. Το δεύτερο είναι ένα ειδικό σύμβολο (σημαία — flag) που καθορίζει τον **τρόπο προσπέλασης** του αρχείου. Αν παραλειφθεί, θεωρείται εξ ορισμού `"r"`.

Τρόποι προσπέλασης αρχείου

"r"

Ανάγνωση (read)

"w"

Εγγραφή (write) — διαγράφει προηγούμενα περιεχόμενα, αν υπάρχουν

"a"

Προσθήκη (append) — διατηρεί προηγούμενα περιεχόμενα

"r+"

Άνοιγμα και για ανάγνωση και για εγγραφή

Δημιουργία αρχείου

Μπορούμε να δημιουργήσουμε ένα αρχείο δεδομένων είτε χρησιμοποιώντας ένα συντάκτη (editor), όπως το Notepad, είτε τη συνάρτηση `open()` με το όρισμα `"w"`:

```
python 2.7

fin = open('words.txt', 'w')
print fin
```

Αν δεν υπάρχει το αρχείο `words.txt`, η `open()` το δημιουργεί. Αν υπάρχει, τα περιεχόμενά του χάνονται. Με το όρισμα `"a"`, αν δεν υπάρχει το αρχείο, δημιουργείται, ενώ αν υπάρχει, ανοίγει σε κατάσταση προσθήκης δεδομένων στο τέλος του.

Κλείσιμο αρχείου — `close()`

Όταν ολοκληρώσουμε τις λειτουργίες στο αρχείο, καλούμε **απαραίτητα** τη συνάρτηση `close()`. Αυτό οδηγεί το Λειτουργικό Σύστημα στην αποθήκευση δεδομένων που βρίσκονται ακόμη στη μνήμη:

```
python 2.7

fin = open("workfile.txt", "w")
# ... εργασίες στο αρχείο ...
fin.close()
```

Για να ελέγξουμε αν ένα αρχείο έκλεισε, χρησιμοποιούμε την ιδιότητα `closed`, η οποία επιστρέφει `True` ή `False`:

```
python 2.7

fin = open("workfile.txt", "w")
print fin.closed  # False
fin.close()
print fin.closed  # True
```

Σύνοψη

`open("όνομα", "τρόπος")` ανοίγει ή δημιουργεί αρχείο
Τρόποι: `"r"` (ανάγνωση), `"w"` (εγγραφή), `"a"` (προσθήκη), `"r+"` (ανάγνωση + εγγραφή)
Η `close()` κλείνει το αρχείο και αποθηκεύει τα δεδομένα
Η ιδιότητα `.closed` ελέγχει αν το αρχείο είναι κλειστό

4.2 Ανάγνωση και εγγραφή σε αρχείο

Το αρχείο παραδείγματος

Για τα παραδείγματα χρησιμοποιούμε ένα αρχείο `words.txt` με περιεχόμενο:

```
python 2.7

This is line 1
This is line 2
This is line 3
```

Εγγραφή σε αρχείο — `write()`

Για να γράψουμε σε ένα αρχείο, το ανοίγουμε πρώτα με το κατάλληλο όρισμα. Με `'w'` διαγράφονται τα υπάρχοντα περιεχόμενα, ενώ με `'a'` διατηρούνται. Χρησιμοποιούμε τη μέθοδο `write()` με όρισμα τη συμβολοσειρά που θέλουμε να εισάγουμε:

```python 2.7
fin = open("words.txt", "a")
fin.write("This is line 4\n")
fin.close()
```

Το αρχείο `words.txt` έχει τώρα τη μορφή:

```python 2.7
This is line 1
This is line 2
This is line 3
This is line 4
```

Σε ένα αρχείο κειμένου, αν θέλουμε να υπάρχουν γραμμές και όχι συνεχόμενοι χαρακτήρες, πρέπει να σημειώνουμε την αλλαγή γραμμής με το χαρακτήρα `"\n"`.

Το όρισμα της `write()` πρέπει να είναι **συμβολοσειρά**. Αν θέλουμε να εισάγουμε έναν αριθμό, τον μετατρέπουμε με `str()`:

```python 2.7
x = 52
fin.write(str(x))
```

Ανάγνωση αρχείου — `read()`

Οι πιο διαδεδομένες μέθοδοι για διάβασμα περιεχομένων ενός αρχείου είναι η `read()` και η `readline()`.

Η `read()` διαβάζει ένα πλήθος χαρακτήρων από την αρχή του αρχείου:

Σύνταξη: fileObject.read([count])

Η παράμετρος `count` καθορίζει τον αριθμό των χαρακτήρων που θα διαβαστούν. Αν λείπει, γίνεται ανάγνωση μέχρι το τέλος του αρχείου.

```python 2.7
fin = open('words.txt', 'r')

# Ανάγνωση και εμφάνιση ενός μόνο χαρακτήρα
print fin.read(1)  # T

# Διάβασμα και εμφάνιση των επόμενων 13 χαρακτήρων
print fin.read(13) # his is line 1

# Εμφάνιση ολόκληρου του υπόλοιπου αρχείου
print fin.read()   # This is line 2\nThis is line 3

fin.close()
```

Ανάγνωση κατά γραμμή — `readline()`

Η `readline()` διαβάζει μία γραμμή του αρχείου, δηλαδή διαδοχικούς χαρακτήρες μέχρι να συναντήσει το χαρακτήρα νέας γραμμής (`\n`):

```
python 2.7

fin = open("words.txt")
print fin.readline()  # This is line 1
print fin.readline()  # This is line 2
fin.close()
```

Το `fin` καταγράφει τη θέση του μέσα στο αρχείο, οπότε κάθε επόμενη κλήση της `readline()` διαβάζει την επόμενη γραμμή.

Σάρωση αρχείου με `for`

Για ανάγνωση ή εγγραφή κατά γραμμές, μπορούμε να σαρώσουμε το αρχείο με μια δομή επανάληψης:

```
python 2.7

fin = open("words.txt")
for line in fin:
    print line,
fin.close()
```

Εντοπισμός θέσης στο αρχείο

Η μέθοδος `tell()` επιστρέφει έναν ακέραιο που περιέχει την τρέχουσα θέση στο αρχείο, υπολογισμένη σε bytes από την αρχή του. Η επόμενη ανάγνωση ή εγγραφή θα γίνει σε εκείνη τη θέση.

Για να αλλάξουμε την τρέχουσα θέση, χρησιμοποιούμε τη `seek()`:

```
python 2.7

fin = open("workfile.txt", "r+")
fin.write("0123456789abcdef")
fin.seek(5)        # πηγαίνει στο 6ο byte
print fin.read(1)  # 5
fin.seek(-3, 2)     # πηγαίνει στο 3ο byte πριν το τέλος
print fin.read(1)  # d
fin.close()
```

Σύνταξη seek(): file.seek(offset [, from_what])

Η θέση υπολογίζεται προσθέτοντας `offset` (πλήθος bytes) σε ένα σημείο αναφοράς:

- `from_what = 0`: από την αρχή του αρχείου
- `from_what = 1`: από την τρέχουσα θέση
- `from_what = 2`: από το τέλος του αρχείου

Σύνοψη

`write()` γράφει συμβολοσειρά στο αρχείο
`read(n)` διαβάζει `n` χαρακτήρες, `readline()` διαβάζει μία γραμμή
`for line in fin` σαρώνει όλο το αρχείο γραμμή-γραμμή
`tell()` δείχνει τη θέση, `seek()` την αλλάζει

Ενότητα 5: Προηγμένα στοιχεία γλώσσας προγραμματισμού

5.1.1 Υποπρογράμματα

Τι είναι τα Υποπρογράμματα;

Μεταξύ των θεμάτων της αλγοριθμικής επίλυσης προβλημάτων που έχουμε ήδη εξετάσει, είναι και εκείνα της ανάγκης επανάληψης κώδικα με τη χρήση μιας δομής επανάληψης ή ομαδοποίησης και χειρισμού πολλών δεδομένων με τη χρήση άλλων δομών δεδομένων, όπως η Λίστα.

Προχωρώντας σε πιο πολύπλοκα προβλήματα, θα δούμε ότι συχνά κάποια ενέργεια ή και ολόκληρο αλγοριθμικό τμήμα, είναι αναγκαίο να επαναλαμβάνεται, σχεδόν αυτούσια ως λογική επεξεργασία, σε πολλά διαφορετικά σημεία του αλγορίθμου, αλλά κάθε φορά να επεξεργάζεται διαφορετικά δεδομένα.

Φυσικά δεν είναι προγραμματιστικά ορθό να επαναλαμβάνουμε το τμήμα κώδικα με άλλα δεδομένα κάθε φορά. Η γλώσσα προγραμματισμού πρέπει να επιτρέπει τη δημιουργία ομάδας εντολών, ως οντότητα στο πρόγραμμα, που να δέχεται ως είσοδο παραμέτρους και να μπορεί να καλείται από σημεία του προγράμματος.

Τμηματικός Προγραμματισμός

Μια από τις βασικότερες τεχνικές του διαδικαστικού προγραμματισμού είναι ο **Τμηματικός Προγραμματισμός**. Σύμφωνα με την τεχνική αυτή, μπορούμε να γράψουμε ένα πρόγραμμα ως ένα σύνολο από μικρότερα κομμάτια προγράμματος.

Ένα **υποπρόγραμμα** είναι ένα κομμάτι προγράμματος που έχει γραφεί ξεχωριστά από το υπόλοιπο πρόγραμμα και επιτελεί ένα αυτόνομο έργο. Το γράφουμε μία φορά και το χρησιμοποιούμε όσες φορές θέλουμε, καλώντας το αντίστοιχο υποπρόγραμμα.

Βασικά χαρακτηριστικά υποπρογραμμάτων

- Έχει **μόνο ένα σημείο εισόδου** από το οποίο δέχεται τα δεδομένα του.
- Το πρόγραμμα που καλεί ένα άλλο υποπρόγραμμα **σταματάει την εκτέλεσή του** όσο εκτελείται το καλούμενο υποπρόγραμμα. Μόνο ένα υποπρόγραμμα μπορεί να εκτελείται σε μια χρονική στιγμή.
- Ο **έλεγχος επιστρέφει** στο πρόγραμμα που κάλεσε, όταν το καλούμενο υποπρόγραμμα σταματήσει να εκτελείται.

Καλές πρακτικές

Πριν ξεκινήσουμε να γράφουμε ένα πρόγραμμα, μελετάμε πώς αυτό μπορεί να αναλυθεί σε επιμέρους τμήματα και αποφασίζουμε για τα αντίστοιχα υποπρογράμματα. Εξετάζουμε αν κάποια υποπρογράμματα που υπάρχουν σε έτοιμες βιβλιοθήκες μπορούν να χρησιμοποιηθούν. Προσπαθούμε κάθε υποπρόγραμμα να είναι όσο το δυνατόν πιο ανεξάρτητο.

Σύνοψη

Τα υποπρογράμματα επιτρέπουν την επαναχρησιμοποίηση κώδικα
Ο Τμηματικός Προγραμματισμός αναλύει το πρόγραμμα σε μικρότερα αυτόνομα τμήματα
Κάθε υποπρόγραμμα έχει ένα σημείο εισόδου και επιστρέφει τον έλεγχο όταν ολοκληρωθεί
Στην Python, τα υποπρογράμματα υλοποιούνται ως συναρτήσεις

5.1.2 Συναρτήσεις στην Python

Συναρτήσεις στην Python

Κάθε γλώσσα προγραμματισμού διαθέτει ένα δικό της λεξιλόγιο για τις εντολές και τις ενσωματωμένες συναρτήσεις. Η Python παρέχει **ένα μόνο τύπο υποπρογραμμάτων**, τις συναρτήσεις, τις οποίες τις θεωρεί ως αντικείμενα.

Για να εξασκηθούμε μπορούμε να δημιουργήσουμε τις δικές μας συναρτήσεις και έτσι να κατανοήσουμε τη λειτουργία και τη χρήση τους. Σε υψηλότερο επίπεδο, μπορούμε να χρησιμοποιούμε έτοιμες βιβλιοθήκες.

Ορισμός και κλήση Συνάρτησης

Ο ορισμός μιας συνάρτησης γίνεται με τη λέξη κλειδί `def` που την ακολουθεί ένα όνομα, ένα ζεύγος παρενθέσεων και η δήλωση τελειώνει με διπλή τελεία (:). Κάτω από τη γραμμή αυτή τοποθετούνται, σε εσοχή, οι εντολές της συνάρτησης.

Παράδειγμα — χωρίς παραμέτρους

```
python 2.7

def fun_name():
    print "hello"
```

Η συνάρτηση αυτή, όταν κληθεί, εμφανίζει στην οθόνη τη λέξη hello.

Παράδειγμα — με δύο παραμέτρους

```
python 2.7

def find_sum(par1, par2):
    result = par1 + par2
    return result
```

Η συνάρτηση αυτή επιστρέφει το άθροισμα των τιμών των δύο παραμέτρων.

Κλήση συνάρτησης

```
python 2.7

>>> type(45)
< type 'int'>
```

Εδώ, η συνάρτηση `type` δέχεται το όρισμα 45 και επιστρέφει τον τύπο του.

```
python 2.7

>>> find_sum(3, 4)
7
>>> find_sum("well", "come")
"wellcome"
```

Η συνάρτηση καλείται με ορίσματα 3 και 4 και επιστρέφει 7. Με συμβολοσειρές, ο τελεστής + λειτουργεί ως συνένωση λόγω πολυμορφισμού.

Μια συνάρτηση πρέπει να έχει οριστεί πριν χρησιμοποιηθεί. Οι δηλώσεις μέσα στη συνάρτηση δεν εκτελούνται μέχρι αυτή να κληθεί.

Ροή εκτέλεσης

Μια κλήση συνάρτησης είναι σαν μια παράκαμψη στη ροή της εκτέλεσης. Αντί η ροή να πάει στην επόμενη δήλωση, περνάει στο σώμα της συνάρτησης, εκτελεί όλες τις δηλώσεις εκεί και μετά επιστρέφει για να συνεχίσει από εκεί που σταμάτησε.

Κατηγορίες συναρτήσεων

Μια πρώτη κατηγοριοποίηση:

- **α)** Αυτές που **δεν τροποποιούν** το αντικείμενο στο οποίο εφαρμόζονται
- **β)** Εκείνες που **τροποποιούν** το αντικείμενο στο οποίο καλούνται

Παράδειγμα μη τροποποίησης: `str.upper()` επιστρέφει νέα συμβολοσειρά χωρίς να αλλάζει την αρχική.

```
python 2.7

>>> a = 'Python'
>>> print a.upper()
PYTHON
>>> print a
Python
```

Παράδειγμα τροποποίησης: `list.append()` αλλάζει τη λίστα επί τόπου.

```
python 2.7

>>> b = ['a', 'b', 'c', 'd']
>>> print b.append('e')
None
>>> print b
['a', 'b', 'c', 'd', 'e']
```

Σύνοψη

Η Python παρέχει ένα τύπο υποπρογραμμάτων: τις συναρτήσεις
Ορισμός με `def`, κλήση με το όνομα και παρενθέσεις
Μπορεί να δέχεται παραμέτρους και να επιστρέφει τιμή με `return`
Η συνάρτηση πρέπει να έχει οριστεί πριν χρησιμοποιηθεί

5.2.1 Παράμετροι συναρτήσεων

Παράμετροι συναρτήσεων

Ένα υποπρόγραμμα αποτελεί ένα ανεξάρτητο τμήμα προγράμματος που μπορεί να καλείται από σημεία του προγράμματος. Δέχεται τιμές από το τμήμα προγράμματος που το καλεί και μετά την εκτέλεση των εντολών του επιστρέφει σε αυτό νέες τιμές.

Οι τιμές αυτές που μεταβιβάζονται λέγονται **παράμετροι** και διακρίνονται σε παραμέτρους Εισόδου και Εξόδου. Οι παράμετροι είναι μεταβλητές για τη μεταβίβαση τιμών μεταξύ υποπρογραμμάτων.

Παράδειγμα printMax

```
python 2.7
```

```
def printMax(a, b):
    if a > b:
        print a, 'είναι το μέγιστο'
    elif a == b:
        print a, 'είναι ίσο με το', b
    else:
        print b, 'είναι το μέγιστο'

printMax(3, 4)  # σταθερές
x = 5
y = 7
printMax(x, y)  # μεταβλητές
```

```
python 2.7
```

```
4 είναι το μέγιστο
7 είναι το μέγιστο
```

Παράδειγμα print_twice

```
python 2.7
```

```
def print_twice(x):
    print x
    print x

>>> print_twice('Spam')
Spam
Spam
>>> print_twice(25)
25
25
```

Pass by Reference

Όταν οι παράμετροι περνάνε με αναφορά, αν αλλάξουμε μια παράμετρο μέσα στη συνάρτηση, η αλλαγή είναι μόνιμη.

```
python 2.7
```

```
# Ορισμός συνάρτησης
def changeme(mylist):
    mylist.append([1, 2, 3, 4])
    print "Τιμές μέσα: ", mylist

# Κλήση
mylist = [10, 20, 30]
changeme(mylist)
print "Τιμές έξω: ", mylist
```

```
python 2.7
```

```
Τιμές μέσα: [10, 20, 30, [1, 2, 3, 4]]
Τιμές έξω: [10, 20, 30, [1, 2, 3, 4]]
```

Παράκαμψη παραμέτρου

Αν η παράμετρος αποκτήσει νέα αναφορά μέσα στη συνάρτηση, η αλλαγή ισχύει μόνο τοπικά:

```
python 2.7

def changeme(mylist):
    mylist = [1, 2, 3, 4]  # Νέα αναφορά
    print "Τιμές μέσα: ", mylist

mylist = [10, 20, 30]
changeme(mylist)
print "Τιμές έξω: ", mylist
```

```
python 2.7

Τιμές μέσα: [1, 2, 3, 4]
Τιμές έξω: [10, 20, 30]
```

Η παράμετρος mylist είναι τοπική στη συνάρτηση changeme. Αλλάζοντας την τιμή της, η αλλαγή εφαρμόζεται μόνο στην "περιοχή" της συνάρτησης.

Σύνοψη

Οι παράμετροι μεταβιβάζουν τιμές σε μία συνάρτηση
Τα ορίσματα εκχωρούνται σε μεταβλητές-παραμέτρους
Οι μεταβλητές λίστες περνάνε με αναφορά
Η εκχώρηση νέας τιμής σε παράμετρο δημιουργεί τοπική μεταβλητή

5.2.2 Εμβέλεια των μεταβλητών

Εμβέλεια των μεταβλητών

Όλες οι μεταβλητές σε ένα πρόγραμμα δεν μπορούν να είναι προσπελάσιμες από όλα τα μέρη του προγράμματος. Η **εμβέλεια (scope)** μιας μεταβλητής αναφέρεται στο τμήμα του προγράμματος που μπορεί αυτή να έχει πρόσβαση.

- **Απεριόριστη εμβέλεια:** Οι **καθολικές (global)** μεταβλητές είναι ορατές παντού. Μειονέκτημα: περιορίζεται η ανεξαρτησία των υποπρογραμμάτων.
- **Περιορισμένη εμβέλεια:** Οι **τοπικές (local)** μεταβλητές ισχύουν μόνο για το υποπρόγραμμα στο οποίο δηλώθηκαν.

Καθολικές και Τοπικές μεταβλητές

Οι μεταβλητές μέσα στο σώμα της συνάρτησης έχουν τοπική εμβέλεια, ενώ αυτές έξω έχουν καθολική εμβέλεια:

```
python 2.7

total = 0  # Καθολική

def sum(arg1, arg2):
    total = arg1 + arg2  # Τοπική
    print "Μέσα: ", total
    return total

sum(10, 20)
print "Έξω: ", total
```

```
python 2.7

Μέσα: 30
Έξω: 0
```

Παράδειγμα τοπικών μεταβλητών

```
python 2.7

x = 50

def func(x):
    print 'Το x είναι', x
    x = 2
    print 'Το τοπικό x άλλαξε σε', x

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

```
python 2.7

Το x είναι 50
Το τοπικό x άλλαξε σε 2
Το x είναι ακόμα 50
```

Χρήση της εντολής global

Για να αλλάξουμε καθολική μεταβλητή μέσα σε συνάρτηση, χρησιμοποιούμε `global`:

```
python 2.7

x = 50

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

func()
print 'Η τιμή του x είναι', x
```

```
python 2.7

Το x είναι 50
Το καθολικό x άλλαξε σε 2
Η τιμή του x είναι 2
```

Παράδειγμα επίδειξης

Η Python θεωρεί ότι αν μέσα σε μια συνάρτηση εκχωρηθεί τιμή σε μεταβλητή, αυτή είναι τοπική:

```
python 2.7

>>> myGlobal = 5
def func1():
    myGlobal = 42

def func2():
    print myGlobal

>>> func1()
>>> func2()
5
```

Με `global` στη func1(), η func2() τυπώνει 42:

```
python 2.7

>>> myGlobal = 5
def func1():
    global myGlobal
    myGlobal = 42

def func2():
    print myGlobal

>>> func1()
>>> func2()
42
```

Αν η Python διαβάζει τιμή από μεταβλητή που δεν υπάρχει τοπικά, την αναζητά σε εξωτερική εμβέλεια (π.χ. καθολική).

Σύνοψη

Οι μεταβλητές μέσα σε συνάρτηση είναι **τοπικές**
Οι μεταβλητές έξω από συναρτήσεις είναι **καθολικές**
Η `global` επιτρέπει σε συνάρτηση να τροποποιήσει καθολική μεταβλητή

5.3.1 Εισαγωγή

Εισαγωγή στα Αρθρώματα (Modules)

Έχουμε δει πώς μπορούμε να επαναχρησιμοποιήσουμε κώδικα ορίζοντας συναρτήσεις. Τι κάνουμε όμως αν θέλουμε να επαναχρησιμοποιήσουμε έναν αριθμό συναρτήσεων σε άλλα προγράμματα; Η λύση είναι τα **αρθρώματα (modules)**.

Ένα άρθρωμα είναι ένα αρχείο .py που μπορεί να ορίζει συναρτήσεις, κλάσεις και μεταβλητές. Η ομαδοποίηση σχετικού κώδικα σε ένα module τον κάνει ευκολότερο στην κατανόηση και χρήση.

Η δήλωση import

```
python 2.7

import module1[, module2[, ... moduleN]]
```

Για πρόσβαση σε συνάρτηση χρησιμοποιούμε **συμβολισμό με τελεία**:

```
python 2.7

>>> import math
>>> print math.pi
3.14159265359
```

Η from...import

Εισάγει συγκεκριμένα στοιχεία χωρίς να φορτώνει ολόκληρο το module:

```
python 2.7

from modname import name1[, name2[, ... nameN]]
```

Η from...import *

```
python 2.7

from modname import *
```

Δεν ενδείκνυται η συχνή χρήση του from...import *, γιατί μπορεί να προκαλέσει συγκρούσεις ονομάτων.

Κατηγορίες συναρτήσεων

- **Ενσωματωμένες (built-in)** — πάντα διαθέσιμες (π.χ. `abs()`, `type()`)
- **Από εξωτερικά αρθρώματα** — εισάγονται με `import` (π.χ. `math.sqrt()`)
- **Οριζόμενες από τον προγραμματιστή** — με `def`

Παράδειγμα

```
python 2.7

from math import sqrt

def cube(x):
    return x * x * x

>>> print abs(-1)
1
>>> print cube(9)
729
>>> print sqrt(81)
9.0
```

Σύνοψη

Module = αρχείο .py με συναρτήσεις και μεταβλητές
`import module` εισάγει ολόκληρο το άρθρωμα
`from module import name` εισάγει συγκεκριμένα στοιχεία
Dot notation: `module.func()`

5.3.2 Σύντομη περιγραφή της Πρότυπης βιβλιοθήκης (Standard Library)

Πρότυπη βιβλιοθήκη (Standard Library)

Μια **βιβλιοθήκη (library)** είναι μια συλλογή εργαλείων που μπορεί να έχουν γραφτεί και από άλλους προγραμματιστές. Η πρότυπη βιβλιοθήκη της Python περιέχει τεράστιο αριθμό χρήσιμων αρθρωμάτων και είναι μέρος κάθε πρότυπης εγκατάστασης.

Περιλαμβάνει τμήματα για προγραμματισμό γραφικών (Tkinter), αριθμητική επεξεργασία, web συνδεσιμότητα, βάσεις δεδομένων (Sqlite3), Βιοπληροφορική (Biopython) κ.ά.

Math module

```
python 2.7

>>> import math
>>> print math
<module 'math' (built-in)>
>>> print math.pi
3.14159265359
>>> print math.cos(math.pi / 4.0)
0.70710678118654757
```

Αν εισάγουμε απευθείας:

```
python 2.7

>>> from math import pi
>>> print pi
3.14159265359
```

```
python 2.7

>>> from math import *
>>> cos(pi)
-1.0
```

Random module

```
python 2.7

>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.random()
0.17970987693706186
>>> random.randrange(6)
5
```

Σύνοψη

Η Πρότυπη βιβλιοθήκη περιέχει πολλά χρήσιμα αρθρώματα
Το `math` περιέχει μαθηματικές σταθερές και συναρτήσεις
Το `random` περιέχει συναρτήσεις τυχαίων αριθμών
Πρέπει να εισάγουμε ένα module πριν το χρησιμοποιήσουμε

5.3.3 Πακέτα (Packages)

Πακέτα (Packages)

Μέχρι τώρα, πρέπει να έχουμε αρχίσει να παρατηρούμε την ιεραρχία με την οποία οργανώνονται τα προγράμματά μας:

- Οι **μεταβλητές** πηγαίνουν μέσα στις συναρτήσεις
- Οι **συναρτήσεις** και οι καθολικές μεταβλητές πηγαίνουν μέσα στα αρθρώματα
- Τα **πακέτα** οργανώνουν ιεραρχικά τα αρθρώματα

Κάθε κατάλογος με ένα `__init__.py` αρχείο αναφέρεται ως Python πακέτο.

Δομή πακέτου

```
python 2.7

my_package/
    __init__.py
    module1.py
    module2.py
```

Παράδειγμα

```
python 2.7

from my_package import module1
module1.some_function()
```

Σύνοψη

Package = κατάλογος με αρχεία .py και __init__.py
Επιτρέπει ιεραρχική οργάνωση των modules
Το __init__.py εκτελείται όταν εισάγεται το πακέτο

Ενότητα 6: Δομές Δεδομένων II

6.1 Συμβολοσειρές (Strings)

Βασικές έννοιες

Τα αλφαριθμητικά ή **συμβολοσειρές** (strings) στην Python είναι ακολουθίες από χαρακτήρες που έχουν **σταθερό μέγεθος** και **μη μεταβαλλόμενα περιεχόμενα**. Δεν μπορούμε να προσθέσουμε ή να αφαιρέσουμε χαρακτήρες, ούτε να τροποποιήσουμε τα περιεχόμενα του αλφαριθμητικού. Γι' αυτό λέμε ότι η δομή αυτή ανήκει στις **μη μεταβαλλόμενες (immutable)** δομές της Python.

Η αρίθμηση των χαρακτήρων σε ένα αλφαριθμητικό ξεκινάει από το 0. Ο τύπος των αλφαριθμητικών στην Python ονομάζεται `str`.

Παραδείγματα

`word = "PYTHON"` — η αναπαράσταση: word[0]="P", word[1]="Y", word[2]="T", word[3]="H", word[4]="O", word[5]="N"

python 2.7

```
>>> word = "PYTHON"
>>> len(word)
6
>>> print word[5] + word[0]
NP
>>> str(28) == '28'
True
>>> print int('496') + 4
500
```

- Η `len()` επιστρέφει το μήκος, δηλαδή το πλήθος των χαρακτήρων
- Ο τελεστής `+` σε strings κάνει συνένωση
- Η `str()` μετατρέπει μια τιμή σε συμβολοσειρά
- Η `int()` μετατρέπει αλφαριθμητικό σε ακέραιο

Έλεγχος ύπαρξης — τελεστής in

Ο υπαρξιακός τελεστής `in` ελέγχει αν ένα αντικείμενο ανήκει σε ένα σύνολο αντικειμένων:

python 2.7

```
>>> "Py" in "Python"
True
>>> "a" in "Python"
False
>>> "a" not in "Python"
True
```

Οι συγκριτικοί τελεστές (<, <=, >, >=, ==, !=) ισχύουν και στις συμβολοσειρές, με βάση τη λεξικογραφική διάταξη:

```
python 2.7

>>> 'antonis' > 'antonia'
True
>>> '1000' < '2'
True
```

Σάρωση συμβολοσειράς με for

Μια συμβολοσειρά είναι ακολουθία (sequence). Μπορούμε να σαρώσουμε τους χαρακτήρες της με `for`:

```
python 2.7

def trimSpaces(sentence):
    result = ""
    for char in sentence:
        if char != " ":
            result += char
    return result

>>> phrase = "Houston we have a problem"
>>> trimSpaces(phrase)
'Houstonwehaveaproblem'
```

Καταμέτρηση φωνηέντων

```
python 2.7

def count_vowels(word):
    vowels = "AEIOUaeiou"
    count = 0
    for letter in word:
        if letter in vowels:
            count += 1
    return count
```

Ο τελεστής `in` χρησιμοποιείται με δύο τρόπους: για τον καθορισμό του εύρους επανάληψης (`for letter in word`) και για έλεγχο ύπαρξης (`if letter in vowels`).

Σύνοψη

Οι συμβολοσειρές είναι immutable (μη μεταβαλλόμενες)
Η αρίθμηση ξεκινά από το 0
Τελεστής `in` για έλεγχο ύπαρξης
Σάρωση με `for char in string`
`len()`, `str()`, `int()`

6.2 Λίστες

Βασικές έννοιες

Η **λίστα** είναι μια διατεταγμένη ακολουθία αντικειμένων, όχι απαραίτητα του ίδιου τύπου και αποτελεί τη βασική δομή δεδομένων της Python. Σε αντίθεση με τη συμβολοσειρά, είναι **δυναμική** δομή (mutable) — μπορούμε να προσθέτουμε ή να αφαιρούμε στοιχεία.

Η αρίθμηση των στοιχείων ξεκινάει από το 0:

```
python 2.7

>>> L = [3, 5, 8, 13, 21, 34]
>>> print L[0]
3
>>> print L[5]
34
```

Προσθήκη και τροποποίηση

```
python 2.7

>>> daysofweek = ["Δευτέρα", "Τρίτη", "Τετάρτη", "Πέμπτη", "Παρασκευή"]
>>> daysofweek = daysofweek + ["Σάββατο"]
>>> daysofweek[0] = daysofweek[1] = "Κυριακή"
>>> print daysofweek
["Κυριακή", "Κυριακή", "Τετάρτη", "Πέμπτη", "Παρασκευή", "Σάββατο"]
```

Οι παραπάνω εντολές δεν προσθέτουν στοιχείο στην υπάρχουσα λίστα, αλλά δημιουργούν νέα κάθε φορά. Για αποδοτική προσθήκη στο τέλος προτιμούμε +=: `Lista += [stoixeio]`

Χαρακτηριστικά λιστών

- Δεν έχουν σταθερό μέγεθος — αυξάνονται και μειώνονται δυναμικά
- Αρίθμηση δεικτών από το 0
- Μπορούν να περιέχουν στοιχεία διαφορετικού τύπου
- Υποστηρίζουν `in`, `len()`, `+` (συνένωση)

```
python 2.7

>>> mix = [6, 3.14159, True, "Guido Van Rossum"]
>>> len(mix)
4
```

Μέθοδοι λιστών

Οι βασικές μέθοδοι που θα χρησιμοποιήσουμε:

- `L.append(object)` — προσθήκη στοιχείου στο τέλος
- `L.insert(index, object)` — προσθήκη σε συγκεκριμένη θέση
- `L.pop([index])` — αφαίρεση από θέση index (ή τελευταίου)

```
python 2.7

>>> fib = [5, 8, 13, 21, 34]
>>> fib.pop(1)    # fib = [5, 13, 21, 34]
>>> fib.append(55) # fib = [5, 13, 21, 34, 55]
>>> fib.pop()     # fib = [5, 13, 21, 34]
>>> fib.insert(2, 89) # fib = [5, 13, 89, 21, 34]
```

Διάσχιση Λίστας

Με το ιδίωμα `for item in List`:

```python 2.7
L = [1, 2, 3, 4, 5, 6]
for number in L:
    print number
```

Μέσος όρος

```python 2.7
sum = 0.0
for number in L:
    sum = sum + number
average = sum / len(L)
print average
```

Μέγιστη τιμή

```python 2.7
maximum = L[0]
for number in L:
    if number > maximum:
        maximum = number
print maximum
```

Παράδειγμα: Ρέστα

Ταμειακή μηχανή — ελάχιστο πλήθος κερμάτων/χαρτονομισμάτων για ρέστα:

```python 2.7
values = [100, 50, 20, 10, 5, 2, 1]
cost = input("Δώσε το κόστος των αγορών")
payment = input("Δώσε το ποσό της πληρωμής")
change = payment - cost
counter = 0
for value in values:
    counter = counter + (change / value)
    change = change % value
print counter
```

Εφαρμογή: Διαχωρισμός λίστας

Διαχωρισμός αριθμών σε θετικούς και αρνητικούς:

```python 2.7
positives = []
negatives = []
for number in numbers:
    if number > 0:
        positives.append(number)
    else:
        negatives.append(number)
print positives
print negatives
```

Εφαρμογή: Συγχώνευση διατεταγμένων λιστών

Συγχώνευση δύο ταξινομημένων λιστών σε μία νέα ταξινομημένη:

```
python 2.7

def merge(A, B):
    L = []
    while A != [] and B != []:
        if A[0] < B[0]:
            L.append(A.pop(0))
        else:
            L.append(B.pop(0))
    return L + A + B
```

Η συνάρτηση range()

Η `range()` επιστρέφει λίστα αριθμών:

```
python 2.7

>>> range(4)
[0, 1, 2, 3]
>>> range(0, 4)
[0, 1, 2, 3]
>>> range(0, 4, 1)
[0, 1, 2, 3]
>>> range(10, 30, 5)
[10, 15, 20, 25]
>>> range(30, 10, -5)
[30, 25, 20, 15]
```

Η `range(A, M, B)` επιστρέφει λίστα από A έως M (χωρίς το M) με βήμα B.

Τα παρακάτω τμήματα κώδικα κάνουν την ίδια λειτουργία:

```
python 2.7

>>> L = [6, 28, 496, 8128]
>>> for item in L:
...     print item,
6 28 496 8128
```

```
python 2.7

>>> for index in range(0, 4):
...     print L[index],
6 28 496 8128
```

Ο τελεστής διαμέρισης : (slice)

Η έκφραση `word[a:b]` επιστρέφει τμήμα από το a έως το b-1:

```
python 2.7

>>> word = "zanneio gymnasio"
>>> word[:7]
'zanneio'
>>> word[8:]
'gymnasio'
>>> word[3:11]
'neio gym'
```

Ο τελεστής `[:]` δημιουργεί αντίγραφο λίστας: `fib = fibonacci[:]` — δεν ισχύει όμως για συμβολοσειρές.

Σύνοψη

Οι λίστες είναι mutable (δυναμικές δομές)
Υποστηρίζουν `append()`, `insert()`, `pop()`
Διάσχιση με `for item in List`
`range()` για δημιουργία λιστών αριθμών
Τελεστής `:` για διαμέριση (slice)

6.3 Στοίβα

Εισαγωγή

Η **στοίβα** (stack) είναι μια δομή δεδομένων όπου οι εισαγωγές και οι διαγραφές στοιχείων γίνονται μόνο από το ένα άκρο (κορυφή). Λειτουργεί με βάση την αρχή **LIFO** (Last In First Out) — το τελευταίο στοιχείο που εισέρχεται είναι το πρώτο που εξάγεται.

Χαρακτηριστικό παράδειγμα: το κουμπί "Πίσω" (Back) στους φυλλομετρητές ιστού επιστρέφει στην προηγούμενη σελίδα με βάση το ιστορικό επισκέψεων, το οποίο λειτουργεί ως στοίβα.

Βασικές λειτουργίες

- **Δημιουργία** μιας κενής στοίβας
- **Έλεγχος** αν η στοίβα είναι κενή
- **Ώθηση (push)** — εισαγωγή στοιχείου στην κορυφή
- **Απώθηση (pop)** — εξαγωγή στοιχείου από την κορυφή

Όταν απωθούμε ένα στοιχείο από τη στοίβα, πρέπει προηγουμένως να έχουμε εξασφαλίσει ότι η στοίβα δεν είναι κενή.

Υλοποίηση Στοίβας στην Python

Υλοποίηση με εισαγωγές/διαγραφές στο **τέλος** της λίστας:

```python 2.7
def push(stack, item):
    stack.append(item)

def pop(stack):
    return stack.pop()

def isEmpty(stack):
    return len(stack) == 0

def createStack():
    return []
```

Υλοποίηση με εισαγωγές/διαγραφές στην **αρχή** της λίστας:

```python 2.7
def push(stack, item):
    stack.insert(0, item)

def pop(stack):
    return stack.pop(0)

def isEmpty(stack):
    return len(stack) == 0

def createStack():
    return []
```

Διεπαφή (Interface)

Το σύνολο των επικεφαλίδων των συναρτήσεων ονομάζεται **διεπαφή** (interface). Ορίζει **τι** μπορεί να κάνει η δομή και όχι **πώς** το κάνει:

```python 2.7
def push(stack, item)
def pop(stack)
def isEmpty(stack)
def createStack()
```

Διαχωρισμός διεπαφής-υλοποίησης: οι εφαρμογές που χρησιμοποιούν στοίβα είναι ανεξάρτητες από την υλοποίηση. Αλλάζοντας την υλοποίηση, δε χρειάζεται να αλλάξουμε το πρόγραμμα.

Εφαρμογή: Αντιστροφή αριθμών

Το πρόγραμμα διαβάζει αριθμούς μέχρι να δοθεί 0 και τους εμφανίζει σε αντίστροφη σειρά:

```python 2.7
stack = createStack()
number = int(raw_input())
while number != 0:
    push(stack, number)
    number = int(raw_input())
while not isEmpty(stack):
    number = pop(stack)
    print number
```

Σύνοψη

Στοίβα: δομή LIFO (Last In First Out)
Βασικές λειτουργίες: push, pop, isEmpty, createStack
Υλοποίηση με λίστα (append/pop ή insert(0)/pop(0))
Διεπαφή vs Υλοποίηση — ανεξαρτησία προγράμματος

6.4 Ουρά

Εισαγωγή

Η **ουρά** (queue) είναι μια δομή δεδομένων όπου τα στοιχεία εισάγονται από το ένα άκρο (πίσω) και εξάγονται από το άλλο (μπροστά). Λειτουργεί με βάση την αρχή **FIFO** (First In First Out) — το πρώτο στοιχείο που εισέρχεται είναι το πρώτο που εξάγεται.

Παραδείγματα ουρών στην καθημερινότητα:

- Οι ουρές στις τράπεζες και τα σούπερ-μάρκετ
- Η ουρά προγραμμάτων που περιμένουν τον επεξεργαστή
- Η ουρά αιτήσεων προς έναν web server

Βασικές λειτουργίες

- **Δημιουργία** μιας κενής ουράς
- **Έλεγχος** αν η ουρά είναι κενή
- **Εισαγωγή (enqueue)** — προσθήκη στοιχείου στο πίσω μέρος
- **Εξαγωγή (dequeue)** — αφαίρεση στοιχείου από το μπροστινό μέρος

Υλοποίηση Ουράς στην Python

Υλοποίηση με λίστα — εισαγωγή στο τέλος, εξαγωγή από την αρχή:

```python 2.7

def enqueue(queue, item):
    queue.append(item)

def dequeue(queue):
    return queue.pop(0)

def isEmpty(queue):
    return len(queue) == 0

def createQueue():
    return []
```

Παράδειγμα χρήσης ουράς

Προσομοίωση εξυπηρέτησης πελατών:

```python 2.7

queue = createQueue()
enqueue(queue, "Πελάτης Α")
enqueue(queue, "Πελάτης Β")
enqueue(queue, "Πελάτης Γ")

while not isEmpty(queue):
    customer = dequeue(queue)
    print "Εξυπηρετείται: ", customer
```

Σύγκριση Στοίβας και Ουράς

Κύριες διαφορές:

- Στοίβα: LIFO — τελευταίος μπαίνει, πρώτος βγαίνει
- Ουρά: FIFO — πρώτος μπαίνει, πρώτος βγαίνει
- Στοίβα: push/pop στο ίδιο άκρο
- Ουρά: enqueue στο πίσω άκρο, dequeue από το μπροστινό

Σύνοψη

Ουρά: δομή FIFO (First In First Out)
Βασικές λειτουργίες: enqueue, dequeue, isEmpty, createQueue
Υλοποίηση: append για εισαγωγή, pop(0) για εξαγωγή
Χρήση σε προσομοιώσεις και εξυπηρέτηση αιτήσεων

Ενότητα 7: Αντικειμενοστρεφής Προγραμματισμός

7.1 Αντικείμενα και Κλάσεις

Εισαγωγή στον Αντικειμενοστρεφή Προγραμματισμό

Πριν την εμφάνιση του αντικειμενοστρεφούς προγραμματισμού, τα προγράμματα δομούνταν πάνω στην ιδέα της προτεραιότητας των ενεργειών έναντι των δεδομένων. Αυτή η προσέγγιση ονομάζεται **Διαδικαστικός Προγραμματισμός** (procedural programming).

Ο **Αντικειμενοστρεφής Προγραμματισμός** (Object-Oriented Programming — OOP) αλλάζει την εστίαση από τις διαδικασίες στις **έννοιες**. Σε αυτές αναθέτει χαρακτηριστικά, τα οποία ονομάζουμε **ιδιότητες** (attributes), και τα επεξεργάζεται μέσω ειδικών συναρτήσεων που ονομάζουμε **μεθόδους** (methods).

Η Python είναι στη βάση της μια αντικειμενοστρεφής γλώσσα. Οι βασικές βιβλιοθήκες της έχουν δομηθεί με αντικειμενοστρεφή λογική και όλα στην Python αποτελούν στιγμιότυπα μιας κλάσης.

Κλάσεις και Αντικείμενα

Μια **κλάση** (class) είναι ένα πρότυπο που περιγράφει μια έννοια. Ένα **αντικείμενο** (object) είναι μια συγκεκριμένη υλοποίηση της κλάσης.

Παράδειγμα: Η κλάση **όχημα** είναι ένα σχέδιο. Το συγκεκριμένο αυτοκίνητο που βλέπεις είναι ένα **αντικείμενο** (στιγμιότυπο) αυτής της κλάσης.

Ένα αντικείμενο έχει:

- **Ιδιότητες (attributes)** — χαρακτηριστικά (π.χ. μάρκα, χρώμα, ταχύτητα)
- **Μεθόδους (methods)** — ενέργειες (π.χ. επιτάχυνση, φρενάρισμα)

Ορισμός κλάσης στην Python

Ο ορισμός κλάσης γίνεται με τη λέξη κλειδί `class`. Οι ιδιότητες ορίζονται μέσα στη μέθοδο `__init__`:

```python
class Vehicle:
    def __init__(self, color, price, wheels, speed):
        self.color = color
        self.price = price
        self.wheels = wheels
        self.speed = speed

    def accelerate(self, amount):
        self.speed += amount
        return self.speed

    def decelerate(self, amount):
        self.speed -= amount
        return self.speed
```

Η `__init__` είναι μια ειδική μέθοδος (κατασκευαστής) που καλείται αυτόματα όταν δημιουργείται ένα νέο αντικείμενο.

Βασικές έννοιες OOP

- **Κληρονομικότητα (inheritance):** Μια κλάση μπορεί να κληρονομήσει ιδιότητες και μεθόδους από μια άλλη κλάση (γονέας → παιδί)
- **Πολυμορφισμός (polymorphism):** Διαφορετικές κλάσεις μπορούν να υλοποιούν την ίδια μέθοδο με διαφορετικό τρόπο
- **Ενθυλάκωση (encapsulation):** Οι ιδιότητες και οι μέθοδοι ομαδοποιούνται μέσα στην κλάση

Σύνοψη

OOP εστιάζει στις έννοιες/αντικείμενα, όχι στις διαδικασίες
Κλάση = πρότυπο, Αντικείμενο = στιγμιότυπο της κλάσης
Ιδιότητες (attributes): χαρακτηριστικά του αντικειμένου
Μέθοδοι (methods): ενέργειες που μπορεί να κάνει το αντικείμενο
Ορισμός με `class`, κατασκευαστής με `__init__`

7.2 Στιγμιότυπα (αυτόματη αρχικοποίηση αντικειμένων)

Στιγμιότυπα (Instances)

Η ύπαρξη της κλάσης **δεν σημαίνει** και την αυτόματη ύπαρξη ενός αντικειμένου. Σημαίνει μόνο ότι υπάρχει η περιγραφή με την οποία μπορούμε να δημιουργήσουμε αντικείμενα.

Ένα **στιγμιότυπο** (instance) είναι ένα συγκεκριμένο αντικείμενο που δημιουργείται από μια κλάση, με συγκεκριμένες τιμές για τις ιδιότητές του.

Η μέθοδος __init__ και ο κατασκευαστής

Η `__init__` είναι η μέθοδος που αποτελεί τον **κατασκευαστή** (constructor) των αντικειμένων. Καλείται αυτόματα με κάθε νέα δημιουργία αντικειμένου:

```
python 2.7

class Vehicle:
    def __init__(self, color, price, wheels, speed):
        self.color = color
        self.price = price
        self.wheels = wheels
        self.speed = speed
```

Για να δημιουργήσουμε ένα αντικείμενο, καλούμε την κλάση με ορίσματα που αντιστοιχούν στις παραμέτρους της `__init__`:

```
python 2.7

>>> mybeetle = Vehicle('yellow', 2000.00, 4, 80)
```

Το `mybeetle` είναι ένα στιγμιότυπο της κλάσης Vehicle, με χρώμα κίτρινο, τιμή 2000, 4 τροχούς και ταχύτητα 80.

Η παράμετρος self

Η `self` εμφανίζεται ως πρώτη παράμετρος σε όλες τις μεθόδους. Επιτρέπει στη μέθοδο να αναφέρεται στο **ίδιο το αντικείμενο** και όχι σε ολόκληρη την κλάση.

Η λέξη `self` δεν είναι δεσμευμένη λέξη της Python, αλλά σύμβαση μεταξύ προγραμματιστών. Πάντα χρησιμοποιείται ως πρώτη παράμετρος.

Με κλήση όπως `mybeetle.accelerate(10)`, η Python αντιλαμβάνεται ως `self` το `mybeetle`. Έτσι η μέθοδος επιδρά μόνο στο συγκεκριμένο αντικείμενο.

Παράδειγμα: Κλάση Dog

Δημιουργία κλάσης με δύο στιγμιότυπα:

python 2.7

```
class Dog:
    def __init__(self, breed, size, color):
        self.breed = breed
        self.size = size
        self.color = color
        print "breed:", self.breed, "size:", self.size, "color:", self.color

    def eat(self, food):
        self.food = food
        print "I am eating", food

    def bark(self):
        print "I am barking"

Max = Dog("terrier", "medium", "brown")
Rocky = Dog("labrador", "large", "light brown")
Max.eat("bone")
Rocky.eat("meat")
```

Σωστή αρχικοποίηση ιδιοτήτων

Κάθε ιδιότητα που χρησιμοποιείται από άλλες μεθόδους πρέπει να αρχικοποιείται πριν χρησιμοποιηθεί. Ο πιο εύκολος τρόπος είναι να ορίζονται **όλες** μέσα στην `__init__`:

python 2.7

```
class Vehicle1:
    def __init__(self, color, price, wheels):
        self.color = color
        self.price = price
        self.wheels = wheels

    def set_speed(self, speed=100):
        self.speed = speed

    def accelerate(self, amount):
        self.speed += amount   # ΛΑΘΟΣ: speed δεν αρχικοποιήθηκε
        return self.speed
```

Η αρχικοποίηση ιδιοτήτων έξω από την `__init__` οδηγεί εύκολα σε λογικά λάθη. Σωστή πρακτική: όλες οι ιδιότητες αρχικοποιούνται μέσα στην `__init__`.

Δραστηριότητα

Δίνεται η παρακάτω κλάση:

```
python 2.7

class Car:
    def __init__(self, make):
        self.make = make
        self.speed = 60

    def speed_up(self, speed):
        self.speed = speed
        print "I am driving at", self.speed, "km/h"

    def turn(self):
        print "I am turning..."
```

Ερωτήσεις:

- Ποιος είναι ο κατασκευαστής;
- Ποιες είναι οι ιδιότητες και οι μέθοδοι;
- Προσθέστε ιδιότητες `color` και `year` στον κατασκευαστή
- Αλλάξτε τη μέθοδο `turn` να δέχεται παράμετρο στροφής ("αριστερά"/"δεξιά")
- Δημιουργήστε στιγμιότυπα `convertible` ("bmw", "μαύρο", 2013) και `sedan` ("toyota", "κόκκινο", 2009)

Σύνοψη

Στιγμιότυπο = συγκεκριμένο αντικείμενο από μια κλάση
Η __init__ είναι ο κατασκευαστής (constructor)
Η self αναφέρεται στο τρέχον αντικείμενο
Όλες οι ιδιότητες πρέπει να αρχικοποιούνται στην __init__

7.3 Ιδιότητες και Μέθοδοι

Ιδιότητες (Attributes)

Οι **ιδιότητες** (attributes) είναι τα χαρακτηριστικά ενός αντικειμένου. Μπορούν να δημιουργηθούν είτε μέσα στην κλάση είτε από το στιγμιότυπο.

Δημιουργία ιδιότητας στην κλάση:

```
python 2.7

class C:
    classattr = "attr on class"
```

Δημιουργία ιδιότητας από το στιγμιότυπο:

```
python 2.7

>>> cobj = C()
>>> cobj.instattr = "attr on instance"
```

Πρόσβαση σε ιδιότητες — Dot Notation

Η πρόσβαση γίνεται με **dot notation**: `όνομα_αντικειμένου.ιδιότητα`

```
python 2.7

>>> cobj = C()
>>> cobj.instattr = "attr on instance"
>>> cobj.instattr
'attr on instance'
>>> cobj.classattr
'attr on class'
```

Η έκφραση `cobj.classattr` σημαίνει: "πήγαινε στο αντικείμενο cobj και πάρε την τιμή της ιδιότητας classattr".

Μέθοδοι (Methods)

Οι **μέθοδοι** είναι συναρτήσεις που ανήκουν σε ένα αντικείμενο. Δύο συντακτικές διαφορές από τις συναρτήσεις:

- Ορίζονται **μέσα** στον ορισμό της κλάσης
- Η σύνταξη κλήσης είναι διαφορετική (`αντικείμενο.μέθοδος()`)

Ως **υπογραφή** μεθόδου ορίζεται το όνομα, τα ορίσματα και η τιμή επιστροφής.

```
python 2.7

class C:
    classattr = "attr on class"

    def my_print(self):
        print "my attribute is", self.classattr
```

```
python 2.7

>>> cobj = C()
>>> cobj.my_print()
my attribute is attr on class
```

Στατικές Μέθοδοι (@staticmethod)

Υπάρχουν μέθοδοι που δεν χρειάζονται την `self` — το αποτέλεσμά τους είναι το ίδιο ανεξάρτητα από το ποιο στιγμιότυπο τις καλεί.

Χρησιμοποιούμε το διακριτικό `@staticmethod`:

```
python 2.7

class Dog:
    def __init__(self, breed, size, color):
        self.breed = breed
        self.size = size
        self.color = color

    @staticmethod
    def bark():
        print "I am barking"
```

Στις στατικές μεθόδους δε χρησιμοποιούμε την `self`. Η `@staticmethod` είναι πρακτικά μια κλασική συνάρτηση τοποθετημένη μέσα σε κλάση.

Μέθοδοι Κλάσης (@classmethod)

Οι μέθοδοι κλάσης κληρονομούνται και χρησιμοποιούν την κλάση (όχι το αντικείμενο) ως πρώτη παράμετρο (`cls`):

python 2.7

```
class Dog:
    no_inst = 0

    def __init__(self, breed, size, color):
        self.breed = breed
        self.size = size
        self.color = color
        Dog.no_inst += 1

    @classmethod
    def get_no_of_dogs(cls):
        return cls.no_inst
```

Χρήση: `Dog.get_no_of_dogs()` επιστρέφει τον αριθμό των στιγμιοτύπων που έχουν δημιουργηθεί.

Σύνοψη

Ιδιότητες = χαρακτηριστικά αντικειμένου (attributes)
Μέθοδοι = συναρτήσεις που ανήκουν σε αντικείμενο
Dot notation: `αντικείμενο.ιδιότητα` ή `αντικείμενο.μέθοδος()`
`@staticmethod`: ανεξάρτητη από στιγμιότυπο
`@classmethod`: χρησιμοποιεί την κλάση (cls)