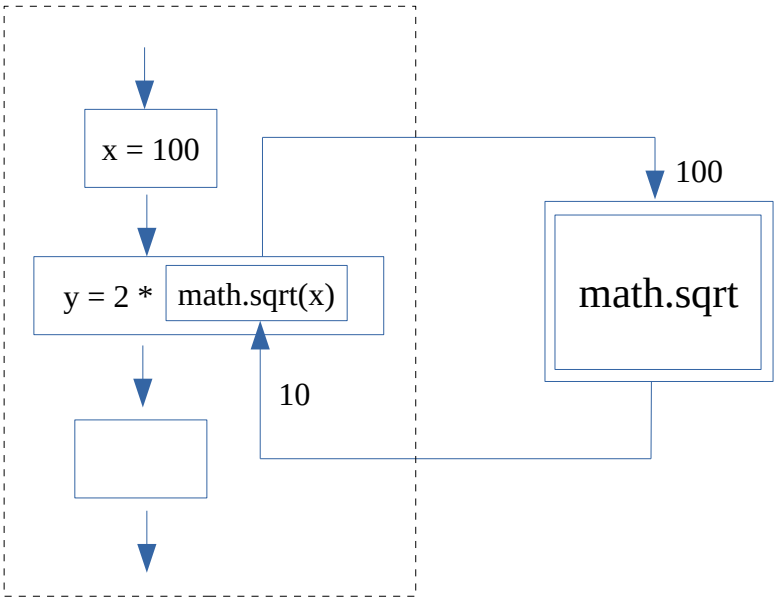
ΠΛΗΡΟΦΟΡΙΚΗ Ι (Python)
Συναρτήσεις

Συναρτήσεις

Συνάρτηση ονομάζεται ένα τμήμα κώδικα (ή υποπρόγραμμα) το οποίο κάνει μια συγκεκριμένη διεργασία και μπορεί να καλείται από ένα πρόγραμμα (ή από μια άλλη συνάρτηση).

Π.χ.



Ουσιαστικά: $x \longrightarrow$ math.sqrt $\longrightarrow \sqrt{x}$

(Η `sqrt` είναι μια συνάρτηση της "βιβλιοθήκης" `math` της Python – Περισσότερα στη συνέχεια)

Η διαδικασία που ακολουθείται είναι η εξής:

1. Ένα πρόγραμμα καλεί μια συνάρτηση
2. Η ροή του προγράμματος μεταφέρεται προσωρινά στις εντολές της συνάρτησης
3. Εκτελούνται οι εντολές της συνάρτησης
4. Η ροή του προγράμματος επιστρέφει στο πρόγραμμα

Όπως προαναφέρθηκε, μια συνάρτηση μπορεί να καλεί με τη σειρά της άλλες συναρτήσεις.

→ Ο *τρόπος λειτουργίας* μιας συνάρτησης δεν ενδιαφέρει τον χρήστη. Αυτό που τον ενδιαφέρει είναι ο *τρόπος χρήσης* της, δηλαδή:

- το όνομά της
- τι δέχεται (αν δέχεται κάτι)
- τι επιστρέφει (αν επιστρέφει κάτι)

→ <u>Λόγοι ύπαρξης συναρτήσεων:</u>

- Βοηθούν στη λογική σχεδίαση των προγραμμάτων
- Είναι «μαύρα κουτιά» που απλά χρησιμοποιούνται χωρίς να είναι απαραίτητη η γνώση της λειτουργίας τους
- Αποφυγή επανάληψης κώδικα στο ίδιο πρόγραμμα
- Επαναχρησιμοποίηση κώδικα σε άλλα προγράμματα
- Ευκολότερη τροποποίηση κώδικα

→ Είδη συναρτήσεων:

- Ενσωματωμένες συναρτήσεις της Python σε βιβλιοθήκες (modules)
- Συναρτήσεις ορισμένες από τον προγραμματιστή

→ Σχεδιασμός συναρτήσεων:

- Όνομα
- Είσοδος / έξοδος ⟶ interface (περιβάλλον επικοινωνίας)
- Αλγόριθμος

→ Ονόματα συναρτήσεων: Ισχύουν οι ίδιοι κανόνες με τα ονόματα μεταβλητών.

Ορισμός συνάρτησης

Η γενική μορφή δημιουργίας μιας συνάρτησης (ορισμού της συνάρτησης) είναι η εξής:

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

Η πρώτη γραμμή της συνάρτησης ονομάζεται επικεφαλίδα της συνάρτησης (function header).

Κλήση συνάρτησης

Όταν δημιουργείται μια συνάρτηση σε ένα πρόγραμμα, ο κώδικας που περιέχεται στο μπλοκ της συνάρτησης, δηλαδή το περιεχόμενό της, δεν εκτελείται, παρά μόνο όταν υπάρξει κάποια εντολή κλήσης της συνάρτησης. Άρα, οι συναρτήσεις καλούνται με κατάλληλες εντολές κλήσης. Η γενική μορφή κλήσης μιας συνάρτησης είναι η εξής:

```
όνομα_συνάρτησης()
```

Παράδειγμα δημιουργίας συνάρτησης και κλήσης της μέσα σε ένα πρόγραμμα:

```
def kalimera():
    print('Καλημέρα!')
# Τέλος συνάρτησης
print('Είναι πρωί, άρα θα πω:', end = ' ')
kalimera()
```

Το πρόγραμμα αυτό θα εκτυπώσει:

```
Είναι πρωί, άρα θα πω: Καλημέρα!
```

Δηλαδή, η πρώτη εντολή που εκτελείται στο πρόγραμμα είναι αυτή που βρίσκεται μετά το σώμα της συνάρτησης. Στη συνέχεια υπάρχει η εντολή κλήσης της συνάρτησης, οπότε τότε η ροή του κώδικα πηγαίνει στο εσωτερικό της συνάρτησης και εκτελείται η εντολή που εκτυπώνει τη λέξη “Καλημέρα!”.

Στο παράδειγμα αυτό, η κλήση της συνάρτησης γίνεται από το κυρίως τμήμα του προγράμματος. Θα μπορούσε όμως να γίνει και μέσα από κάποια άλλη συνάρτηση. Συνήθως, τα προγράμματα που περιέχουν συναρτήσεις, έχουν μία βασική συνάρτηση, η οποία ονομάζεται `main` και η οποία περιέχει όσες εντολές του προγράμματος δεν θα περιέχονταν σε κάποια συγκεκριμένη συνάρτηση. Οι εντολές αυτές δηλαδή, αποτελούν το βασικό μέρος του προγράμματος. Η συνάρτηση αυτή όμως δεν καλείται αυτόματα, οπότε (συνήθως) στο τέλος του προγράμματος θα πρέπει να υπάρχει η κλήση της. Π.χ.,

```
# Ορισμός της συνάρτησης main
def main():
    print('Αρχή προγράμματος')
    print('Ένα', end = ' ')
    fun1()
    fun2()
    print('Δύο')
    print('Τέλος προγράμματος')

# Ορισμός της συνάρτησης fun1
def fun1():
    print('Τρία', end = ' ')

# Ορισμός της συνάρτησης fun2
def fun2():
    print('Τέσσερα', end = ' ')

# Κλήση της συνάρτησης main για έναρξη του προγράμματος
main()
```

Το πρόγραμμα αυτό θα εκτυπώσει:

```
Αρχή προγράμματος
Ένα Τρία Τέσσερα Δύο
Τέλος προγράμματος
```

Γίνεται σαφές ότι μετά την ολοκλήρωση εκτέλεσης κάποια συνάρτησης, η ροή του κώδικα επιστρέφει στην επόμενη εντολή μετά την εντολή κλήσης της.

Σημείωση 1: Η σειρά με την οποία συντάσσονται οι συναρτήσεις ενός προγράμματος ή η θέση ορισμού τους σε σχέση με τη θέση της εντολής κλήσης τους, δεν έχουν καμία σημασία.

Σημείωση 2: Μια συνάρτηση μπορεί να κληθεί και από το κέλυφος (γραμμή εντολών) του IDLE (στην προτροπή >>>), με την προϋπόθεση ότι το αρχείο προγράμματος που την περιέχει έχει πρώτα μεταγλωττιστεί (με F5-Run) κατά την τρέχουσα λειτουργία τού κελύφους. Η κλήση της γίνεται όπως ακριβώς θα γινόταν και από κάποιο πρόγραμμα ή άλλη συνάρτηση.

Είσοδος συναρτήσεων – Παράμετροι εισόδου

Μια συνάρτηση μπορεί να δεχτεί μία ή περισσότερες τιμές εισόδου, τις οποίες χρησιμοποιεί κατά την εκτέλεσή της. Οι συναρτήσεις που δέχονται τιμές εισόδου, περιέχουν αντίστοιχες ***παραμέτρους εισόδου*** στον ορισμό τους, ως εξής:

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

Κατά την *κλήση* τέτοιων συναρτήσεων, μέσα στις παρενθέσεις αποστέλλονται οι συγκεκριμένες τιμές των παραμέτρων εισόδου, οι οποίες λέγονται ***ορίσματα***, ως εξής:

```
όνομα_συνάρτησης(όρισμα_1, όρισμα_2, κτλ.)
```

Τα ορίσματα αυτά λέγονται “ορίσματα κατά θέση”, αφού κατά την κλήση της συνάρτησης, εκχωρούνται στις παραμέτρους εισόδου που βρίσκονται στις αντίστοιχες θέσεις. Λέγονται επίσης και “κανονικά ορίσματα”, σε αντιδιαστολή με άλλους τύπους ορισμάτων που θα αναφερθούν στη συνέχεια.

Π.χ., η ακόλουθη συνάρτηση `print_max` έχει δύο παραμέτρους εισόδου και εκτυπώνει το μέγιστο των δύο ορισμάτων της:

```
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')

def main():
    print_max(10,15) # Κλήση της συνάρτησης με ορίσματα συγκεκριμένες τιμές
    x = float(input('Δώσε τον 1ο αριθμό: '))
    y = float(input('Δώσε τον 2ο αριθμό: '))
    # Κλήση της συνάρτησης με ορίσματα τα x και y:
    print_max(x, y)

main()
```

Άλλο παράδειγμα:

```
def main():
    value = float(input('Δώσε έναν αριθμό: '))
    print('Το διπλάσιό του ισούται με', end = ' ')
    show_double(value)

# Η συνάρτηση show_double δέχεται ένα όρισμα
# και εμφανίζει το διπλάσιο της τιμής του.
def show_double(number):
    result = number * 2
    print(result)
main()
```

Έξοδος συναρτήσεων – Επιστροφή τιμής

Η "είσοδος συναρτήσεων" που προαναφέρθηκε αποτελεί τον τρόπο επικοινωνίας της συνάρτησης που κάνει την κλήση, προς τη συνάρτηση που καλείται. Στην Python, η συνάρτηση που καλείται *δεν* έχει τη δυνατότητα της αντίστροφης επικοινωνίας (προς τη συνάρτηση που την καλεί) μέσω των παραμέτρων εισόδου της. Αυτό μπορεί να φανεί από το ακόλουθο παράδειγμα:

```
def main():
    value = 99
    print('Η τιμή είναι', value)
    change_me(value)
    print(Πίσω στη main η τιμή είναι, value)

def change_me(arg):
    print('Αλλάζω την τιμή.')
    arg = 0
    print('Τώρα η τιμή είναι', arg)
```

Το πρόγραμμα αυτό θα εκτυπώσει:

```
Η τιμή είναι 99
Αλλάζω την τιμή.
Τώρα η τιμή είναι 0
Πίσω στη main η τιμή είναι 99
```

Δηλαδή, ενώ η τιμή της παραμέτρου εισόδου της συνάρτησης άλλαξε μέσα στη συνάρτηση `change_me`, η αλλαγή αυτή δεν μεταβιβάστηκε πίσω στο όρισμα στη συνάρτηση `main`. Αυτό συμβαίνει γιατί η Python είναι μία από τις γλώσσες προγραμματισμού όπου η μέθοδος μεταβίβασης ορισμάτων σε συναρτήσεις γίνεται με το λεγόμενο *"πέρασμα με τιμή"* ή *"μεταβίβαση με τιμή"* (pass-by-value). Άρα, ουσιαστικά μέσω των παραμέτρων εισόδου η συνάρτηση δεν μπορεί να "επικοινωνήσει" με τη συνάρτηση που την καλεί, παρά μόνο να δεχτεί δεδομένα από αυτήν.

Η επικοινωνία της καλούμενης συνάρτησης με τη συνάρτηση που την κάλεσε, γίνεται μόνο μέσω της δυνατότητας που έχει η συνάρτηση να επιστέφει κάποια τιμή (ή κάποιες τιμές) στη συνάρτηση που την κάλεσε. Αυτό πραγματοποιείται με την εντολή `return`, ως εξής:

```
def όνομα_συνάρτησης():
    εντολή
    εντολή
    κτλ.
    return έκφραση
```

Σημείωση 1: Στον ορισμό της συνάρτησης δεν υπάρχει κάτι που να σηματοδοτεί ότι η συνάρτηση αυτή θα επιστρέφει κάποια τιμή.
Σημείωση 2: Η εντολή `return` μπορεί να περιέχει μεταβλητή επιστρέφοντας την τιμή της ή έκφραση, επιστρέφοντας την τιμή του αποτελέσματός της.
Σημείωση 3: Μια συνάρτηση μπορεί να περιέχει το πολύ μία εντολή `return`.

Η κλήση συναρτήσεων που επιστρέφουν τιμή γίνεται με διαφορετικό τρόπο από την κλήση των συναρτήσεων που δεν επιστρέφουν τιμή. Η διαφορά είναι ότι από τη στιγμή που η συνάρτηση επιστρέφει κάποια τιμή, η τιμή αυτή πρέπει να εκχωρηθεί σε κάποια μεταβλητή. Άρα η κλήση τέτοιων συναρτήσεων γίνεται ως εξής:

```
μεταβλητή = όνομα_συνάρτησης()
```

Προφανώς, το αν δέχεται ή αν επιστρέφει κάτι κάποια συνάρτηση, δεν σχετίζονται. Επομένως μία συνάρτηση που επιστρέφει τιμή, μπορεί και να δέχεται ορίσματα.

Παράδειγμα 1:

```
def double(x):
    result = 2 * x
    return result
```

Παράδειγμα 2:

```
def find_max(a, b):
    if a>b:
        max = a
    else:
        max = b
    return max
def main():
    x = float(input('Δώσε τον 1ο αριθμό: '))
    y = float(input('Δώσε τον 2ο αριθμό: '))
    # Κλήση της συνάρτησης με ορίσματα τα x και y:
    maximum = find_max(x, y)
    print(maximum)
main()
```

Προσοχή: Αν η συνάρτηση `find_max` είχε γραφεί έτσι:

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

δεν θα ήταν λάθος, παρόλο που φαινομενικά έχει δύο εντολές `return`. Ουσιαστικά έχει *μία*, αφού είτε η μία θα εκτελεστεί, είτε η άλλη, ποτέ και οι δύο, αφού ανήκουν σε μπλοκ if/else.

Σημείωση: Μια συνάρτηση μπορεί ουσιαστικά να επιστρέφει διαφορετικού τύπου τιμές σε κάθε κλήση της (π.χ., αριθμητική τιμή ή συμβολοσειρά), εάν π.χ. η μεταβλητή που επιστρέφεται παίρνει τιμή μέσα σε ένα μπλοκ if/else. Δεν περιορίζεται δηλαδή στο να επιστρέφει τιμή κάποιου συγκεκριμένου τύπου. Π.χ., η ακόλουθη συνάρτηση επιστρέφει είτε πραγματικό (`float`) είτε συμβολοσειρά (`str`):

```
def my_sqrt(x):
    if x>=0:
        s = math.sqrt(x)
    else:
        s = 'Σφάλμα! Δόθηκε αρνητικός αριθμός.'
    return s
```

Παραδείγματα εξόδου:

```
>>> print(my_sqrt(9))
3.0
>>> print(my_sqrt(-3))
Σφάλμα! Δόθηκε αρνητικός αριθμός.
```

Με την ίδια λογική, από τη στιγμή που το εάν επιστρέφει κάτι ή όχι μια συνάρτηση *δεν* καθορίζεται στον ορισμό της, μια συνάρτηση που περιέχει μπλοκ if/else μπορεί να επιστρέφει τιμή *μόνο* κατά

περίπτωση. Π.χ.,

```
def foo(a,b):
    if a>b:
        return a
    else:
        print('Μεγαλύτερο είναι το', b)
```

Η συνάρτηση αυτή είναι σωστή, παρόλο που άλλοτε επιστρέφει κάτι και άλλοτε όχι. Παραδείγματα εξόδου:

```
>>> foo(5,2)
5
>>> foo(2,5)
Μεγαλύτερο είναι το 5
>>> print(foo(5,2))
5
>>> print(foo(2,5))
Μεγαλύτερο είναι το 5
None
>>> max = foo(2,5)
Μεγαλύτερο είναι το 5
>>> print(max)
None
```

Γίνεται σαφές ότι στην περίπτωση που μια συνάρτηση δεν επιστέφει κάτι αλλά, παρόλα αυτά, η κλήση της εκχωρηθεί σε μια μεταβλητή (ή χρησιμοποιηθεί ως όρισμα στη συνάρτηση `print`), η συγκεκριμένη εντολή δεν είναι λάθος, και η μεταβλητή στην οποία εκχωρήθηκε δημιουργείται με την τιμή `None`. Η τιμή αυτή δηλώνει ότι η συγκεκριμένη μεταβλητή έχει δημιουργηθεί αλλά δεν της έχει δοθεί ακόμα κάποια τιμή (προφανώς ο τύπος της δεν έχει καθοριστεί ακόμα, οπότε προσωρινά θεωρείται ότι είναι μεταβλητή του ειδικού για αυτές τις περιπτώσεις τύπου `NoneType`).

➔ *Επιστροφή πολλαπλών τιμών:*

Μια συνάρτηση στην Python μπορεί να επιστρέφει *πολλαπλές* τιμές. Αυτό γίνεται με πολλαπλές εκφράσεις στην εντολή `return`, χωρισμένες με κόμμα:

```
def όνομα_συνάρτησης():
    εντολή
    εντολή
    κτλ.
    return έκφραση1, έκφραση2, κτλ.
```

Η κλήση μιας τέτοιας συνάρτησης γίνεται με εκχώρηση τιμής σε ισάριθμο αριθμό μεταβλητών με τον αριθμό των επιστρεφόμενων τιμών. Π.χ., αν μία συνάρτηση επιστρέφει 2 τιμές, η κλήση της θα γινόταν ως εξής:

```
μεταβλητή1, μεταβλητή2 = όνομα_συνάρτησης()
```

Παράδειγμα:

```
def get_name():
    # Είσοδος ονόματος και επώνυμου του χρήστη.
    first = input('Δώσε το μικρό σου όνομα: ')
    last = input('Δώσε το επώνυμό σου: ')
    # Επιστροφή και των δύο ονομάτων.
    return first, last
```

Κλήση:

```
>>> first_name, last_name = get_name()
```

Προεπιλεγμένες τιμές ορίσματος

Υπάρχει η δυνατότητα, κάποιες από τις παραμέτρους εισόδου μιας συνάρτησης να γίνουν *προαιρετικές*, που σημαίνει ότι η κλήση της συνάρτησης μπορεί να μην μεταβιβάσει συγκεκριμένες τιμές για αυτές. Για αυτές τις προαιρετικές παραμέτρους εισόδου, πρέπει να οριστούν *προεπιλεγμένες τιμές* κατά τον ορισμό της συνάρτησης, ως εξής:

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

Στο παράδειγμα αυτό, η *παράμετρος_2* είναι προαιρετική.

Προσοχή: Στην περίπτωση που μία συνάρτηση έχει και κανονικές παραμέτρους εισόδου και παραμέτρους εισόδου με προεπιλεγμένες τιμές ορίσματος, πρώτα πρέπει να βρίσκονται στον ορισμό της οι κανονικές παράμετροι και μετά οι προεπιλεγμένες. Δηλαδή, στο παραπάνω παράδειγμα, μετά την παράμετρο_2, εάν υπήρχαν και άλλες παράμετροι εισόδου, θα έπρεπε όλες να είναι με προεπιλεγμένες τιμές ορίσματος.

Π.χ.,
```
def say_something(message, times = 2):
    print(message * times)
```

Παραδείγματα κλήσης:

```
>>> say_something('Hello', 5)
HelloHelloHelloHelloHello
>>> say_something('World')
WorldWorld
```

Άλλο παράδειγμα:

```
def foo(a=1, b, c=3):  # ΛΑΘΟΣ!
    print('a =', a, ', b =', b, ', c =', c)
```

Ο ορισμός της συνάρτησης `foo` είναι λανθασμένος γιατί μετά την παράμετρο εισόδου `a` που είναι με προεπιλεγμένη τιμή ορίσματος, βρίσκεται κανονική παράμετρος εισόδου (`b`). Για να διορθωθεί, θα πρέπει η παράμετρος `b` να είναι *η πρώτη* παράμετρος εισόδου της συνάρτησης.

Ορίσματα με λέξεις-κλειδί (keyword arguments)

Όταν μια συνάρτηση έχει πάνω από μία παραμέτρους εισόδου, υπάρχει η δυνατότητα μεταβίβασης ορισμάτων όχι κατά θέση (όπως γίνεται με τα κανονικά ορίσματα) αλλά χρησιμοποιώντας τα ονόματα των παραμέτρων εισόδου. Τα ορίσματα αυτά, σε αντιδιαστολή με τα "ορίσματα κατά θέση", λέγονται "ορίσματα με λέξεις-κλειδί" (keyword arguments). Για να μεταβιβαστεί ένα όρισμα ως *όρισμα με λέξη-κλειδί*, η κλήση της συνάρτησης συντάσσεται ως εξής:

```
όνομα_συνάρτησης(όνομα_παραμέτρου = τιμή)
```

Σημείωση: Η δυνατότητα μεταβίβασης ορισμάτων με λέξεις-κλειδί σε μια συνάρτηση, δε σχετίζεται με τον ορισμό της συνάρτησης (όπως συμβαίνει με τις προεπιλεγμένες τιμές ορίσματος). Είναι κάτι που παρέχεται για όλες τις συναρτήσεις της Python και προσδιορίζεται μόνο κατά την κλήση της συνάρτησης.

Τα πλεονεκτήματα αυτής της μεθόδου μεταβίβασης ορισμάτων είναι ότι κάποιος δε χρειάζεται να γνωρίζει τη διάταξη των παραμέτρων εισόδου της συνάρτησης που καλεί (πρέπει όμως να γνωρίζει τα ονόματά τους), και ότι έχει τη δυνατότητα να μεταβιβάζει ορίσματα μόνο σε συγκεκριμένες παραμέτρους εισόδου (αρκεί να υπάρχουν προεπιλεγμένες τιμές ορίσματος για τις υπόλοιπες).

Σημείωση: Επειδή ένα όρισμα με λέξη-κλειδί καθορίζει επακριβώς σε ποια παράμετρο εισόδου θα πρέπει να μεταβιβαστεί, η θέση του στην κλήση της συνάρτησης δεν έχει κάποια σημασία.

Προσοχή: Στην περίπτωση που μια συνάρτηση καλείται και με κανονικά ορίσματα (κατά θέση) και με ορίσματα με λέξεις-κλειδί, πρώτα πρέπει να μπαίνουν τα κανονικά ορίσματα στην κλήση της συνάρτησης και μετά τα ορίσματα με λέξεις-κλειδί.

Παράδειγμα 1:

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

Παραδείγματα κλήσης της `foo`:

```
>>> foo(5, 1, 10)
a = 5 , b = 1 , c = 10
>>> foo(c=10, a=5, b=1)
a = 5 , b = 1 , c = 10
>>> foo(1, 2, c=3)
a = 1 , b = 2 , c = 3
>>> foo(1, c=3, b=2)
a = 1 , b = 2 , c = 3
>>> foo(a=10, b=5, 1)
SyntaxError: non-keyword arg after keyword arg
>>> foo(1, 2, b=5)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    foo(1, 2, b=5)
TypeError: foo() got multiple values for argument 'b'
```

- *1η κλήση:* Κλήση με κανονικά ορίσματα (κατά θέση).
- *2η κλήση:* Κλήση με ορίσματα με λέξη-κλειδί για όλες τις μεταβλητές.
- *3η κλήση:* Κλήση με κανονικά ορίσματα για τις `a` και `b`, και με όρισμα με λέξη-κλειδί για τη `c`.
- *4η κλήση:* Κλήση με κανονικό όρισμα για την `a`, και με ορίσματα με λέξεις-κλειδί για τις `b` και `c`.

- *5η κλήση:* Λάθος, γιατί το κανονικό όρισμα (το `1`) θα έπρεπε να προηγείται των ορισμάτων με λέξεις-κλειδί.
- *6η κλήση:* Λάθος, γιατί το όρισμα με λέξη-κλειδί (το `b=5`) αναφέρεται σε παράμετρο στην οποία έχει ήδη μεταβιβαστεί όρισμα κατά θέση (το `2`).

Παράδειγμα 2:

```
def bar(a, b=5, c=10):
    print('a =', a, ', b =', b, ', c =', c)
```

Παραδείγματα κλήσης της `bar`:

```
>>> bar(100,2)
a = 100 , b = 2 , c = 10
>>> bar(100, c=3)
a = 100 , b = 5 , c = 3
>>> bar(50, c=60, b=70)
a = 50 , b = 70 , c = 60
>>> bar(c=100, 200)
SyntaxError: non-keyword arg after keyword arg
```

- *1η κλήση:* Κλήση με κανονικά ορίσματα (κατά θέση) και χρήση της προεπιλεγμένης τιμής της `c`.
- *2η κλήση:* Κλήση με κανονικό όρισμα για την `a`, με όρισμα με λέξη-κλειδί για την `c`, και με προεπιλεγμένη τιμή για την `b`.
- *3η κλήση:* Κλήση με κανονικό όρισμα για την `a`, και με ορίσματα με λέξεις-κλειδί για τις `b` και `c`.
- *4η κλήση:* Λάθος, γιατί το κανονικό όρισμα (το `200`) θα έπρεπε να προηγείται οποιουδήποτε ορίσματος με λέξη-κλειδί.

Ενσωματωμένες συναρτήσεις – Εντολή `import`

Η Python έχει μια μεγάλη ποικιλία έτοιμων συναρτήσεων που μπορούν να χρησιμοποιηθούν από τον προγραμματιστή. Οι συναρτήσεις αυτές, που ονομάζονται *ενσωματωμένες συναρτήσεις*, ανήκουν στη λεγόμενη πρότυπη βιβλιοθήκη (standard library). Κάποιες από αυτές μπορούν να χρησιμοποιηθούν άμεσα, χωρίς κάποια επιπλέον ενέργεια, όπως π.χ., η `print`, η `input`, η `range` κ.α. Οι περισσότερες όμως ενσωματωμένες συναρτήσεις ανήκουν σε συγκεκριμένα αρχεία, γνωστά ως module. Η τεχνική αυτή βοηθάει στη σωστή οργάνωση των συναρτήσεων, αφού το κάθε module αφορά συγκεκριμένη κατηγορία συναρτήσεων. Π.χ, ένα module είναι το `math`, το οποίο περιέχει όλες τις μαθηματικές συναρτήσεις, ενώ ένα άλλο είναι το `random`, το οποίο περιέχει όλες τις συναρτήσεις που σχετίζονται με ψευδοτυχαίους αριθμούς. Για τη χρήση συναρτήσεων που ανήκει σε κάποιο module, απαιτείται η εντολή `import`, η οποία φορτώνει στη μνήμη τις συναρτήσεις που περιέχει το συγκεκριμένο αρχείο module, έτσι ώστε να μπορούν να χρησιμοποιηθούν από το πρόγραμμα. Η σύνταξή της είναι η εξής:

```
import όνομα_module
```

και στη συνέχεια, οποιαδήποτε συνάρτηση του module αυτού μπορεί να κληθεί ως:

```
όνομα_module.όνομα_συνάρτησης()
```

- **Το module `math`**

Το module `math` φορτώνεται στη μνήμη συμπεριλαμβάνοντας στο πρόγραμμα την εντολή:

```
import math
```

οπότε στη συνέχεια, οποιαδήποτε συνάρτηση του module, καλείται μέσα στο πρόγραμμα με τη μορφή:

`math.`*`όνομα_συνάρτησης`*`()`

Κάποιες από τις συναρτήσεις του module `math`:

`sqrt(x)` → τετραγωνική ρίζα του x
`log(x)` → φυσικός λογάριθμος του x
`log10(x)` → δεκαδικός λογάριθμος του x
`exp(x)` → e^x
`factorial(x)` → x!
`floor(x)` → ο μεγαλύτερος ακέραιος που είναι μικρότερος ή ίσος του x
`ceil(x)` → ο μικρότερος ακέραιος που είναι μεγαλύτερος ή ίσος του x
`sin(x)` → ημίτονο του x
`cos(x)` → συνημίτονο του x
`tan(x)` → εφαπτομένη του x

Σταθερές:

`math.pi` → π / `math.e` → e

Σημείωση: Οι ενσωματωμένες μαθηματικές συναρτήσεις:

`abs(x)` → απόλυτη τιμή του x
`pow(x,y)` → x^y
`round(x)` → στρογγυλοποίηση του x στην πλησιέστερη ακέραια τιμή

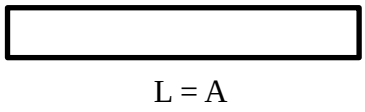
δεν ανήκουν στο module `math`, αλλά στην πρότυπη βιβλιοθήκη, όπως π.χ. οι `print`, `input` κτλ, άρα καλούνται άμεσα, χωρίς το `math.` μπροστά από το όνομά τους.

Προγραμματιστική εφαρμογή:

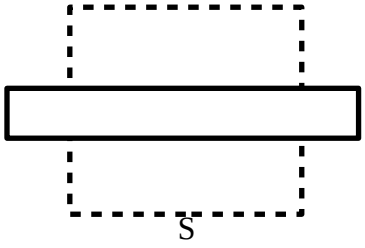
Για δεδομένο αριθμό Α, να βρεθεί η $\sqrt{A}$.

Γεωμετρική αναδιατύπωση: Για θετικό αριθμό Α, να βρεθεί τετράγωνο με εμβαδόν Α.

- Μια αρχική ιδέα:

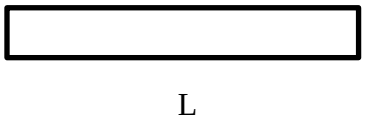


- Παρατήρηση:



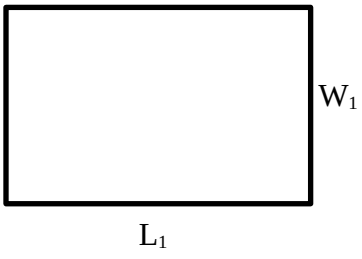
Η απάντηση είναι μεταξύ του L και του W:
W < S < L

- Βασική ιδέα: Αρχικά:



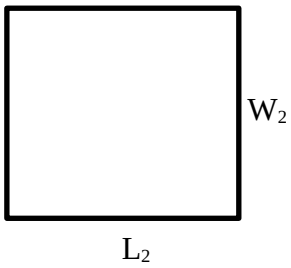
Επόμενο βήμα:

$$L_1 = \frac{L+W}{2}$$
$$W_1 = \frac{A}{L_1}$$



Επόμενο βήμα:

$$L_2 = \frac{L_1+W_1}{2}$$
$$W_2 = \frac{A}{L_2}$$



...

Μετά από κάποιες επαναλήψεις, L ≈ W ≈ $\sqrt{A}$.
Μια αρχική προσέγγιση της συνάρτησης που υλοποιεί τον παραπάνω αλγόριθμο:

```
def my_sqrt(A, steps):
    L = A
    W = A/L
    for i in range(steps):
        L = (L+W)/2
        W = A/L
    return L
```

Μια βελτιωμένη έκδοση της συνάρτησης, η οποία αντί για καθορισμένο αριθμό επαναλήψεων, δέχεται την επιθυμητή ακρίβεια στην προσέγγιση της τετραγωνικής ρίζας είναι η εξής:

```
def my_sqrt2(A, epsilon):
    error = epsilon+1
    L = A
    W = A/L
    while error>epsilon:
        L = (L+W)/2
        W = A/L
        error = (L-W)/L
    return L
```

Στη συνάρτηση αυτή, ως σφάλμα θεωρείται η σχετική διαφορά των τιμών L και W, οι οποίες προσεγγίζουν την τιμή της τετραγωνικής ρίζας από “αντίθετες κατευθύνσεις”. Το ακόλουθο πρόγραμμα χρησιμοποιεί τη συνάρτηση αυτή, καθώς και την ενσωματωμένη συνάρτηση `sqrt` του module `math`, για να υπολογίσει μια προσέγγιση της τετραγωνικής ρίζας ενός αριθμού καθώς και την πραγματική ακρίβεια της προσέγγισης:

```
def main():
    import math
    A = float(input('Δώσε θετικό αριθμό: '))
    e = float(input('Δώσε την επιθυμητή ακρίβεια: '))
    s = my_sqrt2(A, e)
    error = abs(s - math.sqrt(A))
    print('Η προσέγγιση της τετραγωνικής ρίζας του', A, 'ισούται με', s)
    print('Το σφάλμα της προσέγγισης ισούται με', format(error,'.5e'))
```

Δείγματα εξόδου:

```
Δώσε θετικό αριθμό: 99
Δώσε την επιθυμητή ακρίβεια: 0.1
Η προσέγγιση της τετραγωνικής ρίζας του 99.0 ισούται με 9.98124920731545
Το σφάλμα της προσέγγισης ισούται με 3.13748e-02

Δώσε θετικό αριθμό: 99
Δώσε την επιθυμητή ακρίβεια: 0.00001
Η προσέγγιση της τετραγωνικής ρίζας του 99.0 ισούται με 9.949923682546618
Το σφάλμα της προσέγγισης ισούται με 4.93115e-05
```

- **Το module `random`**

Το module `random` φορτώνεται στη μνήμη συμπεριλαμβάνοντας στο πρόγραμμα την εντολή:

```
import random
```

πότε στη συνέχεια, οποιαδήποτε συνάρτηση του module, καλείται μέσα στο πρόγραμμα με τη μορφή:

```
random.όνομα_συνάρτησης()
```

Κάποιες από τις συναρτήσεις του module `random`:

i. `randint`

`randint(x,y)` → τυχαίος ακέραιος στο διάστημα [x, y]
Π.χ.,

```
>>> number = random.randint(1, 100)
>>> print(number)
75
>>> print(random.randint(5,10))
6
```

ii. `randrange`

`randrange(x)` → τυχαίος ακέραιος στο διάστημα [0, x)
`randrange(x,y)` → τυχαίος ακέραιος στο διάστημα [x, y)
`randrange(x,y,z)` → τυχαίος ακέραιος από τη λίστα ακεραίων [x, x+z, x+2*z, ..., y)

Δηλαδή, η `randrange` δημιουργεί λίστα αριθμών όπως θα δημιουργούσε η `range` και στη συνέχεια επιλέγει τυχαία έναν από αυτούς.

Π.χ, η εντολή: `number = random.randrange(0, 101, 10)`
επιστρέφει έναν από τους αριθμούς: `[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]`

iii. `random`

`random()` → τυχαίος πραγματικός αριθμός στο διάστημα [0, 1)

Π.χ.,

```
>>> random.random()
0.4875829280454119
>>> print(format(random.random(), '.4f'))
0.1298
```

iv. `uniform`

`uniform(x,y)` → τυχαίος πραγματικός αριθμός στο διάστημα [x, y)

Το seed στην παραγωγή ψευδοτυχαίων αριθμών

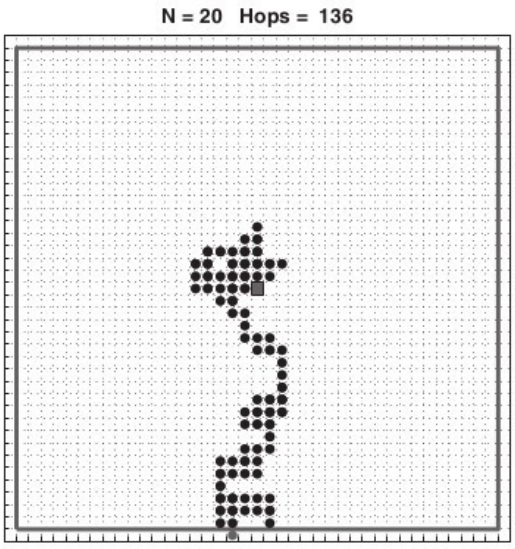
Οι τυχαίοι αριθμοί που παράγονται από τον διερμηνέα της Python (και γενικά από τους υπολογιστές) δεν είναι στην πραγματικότητα τυχαίοι. Παράγονται από κάποια συγκεκριμένη εξίσωση, η οποία όμως αρχικοποιείται με κάποια "τυχαία" τιμή, το λεγόμενο seed. Η τιμή του seed χρησιμοποιείται στον υπολογισμό που επιστρέφει τον επόμενο τυχαίο αριθμό μέσω της κλήσης κάποιας αντίστοιχης συνάρτησης (όπως αυτές που αναφέρθηκαν παραπάνω), και προκύπτει από την τιμή της ώρας του συστήματος τη στιγμή που ένα πρόγραμμα συμπεριλαμβάνει το module `random` με την εντολή `import`. Η ώρα του συστήματος είναι ένας ακέραιος αριθμός που αντιπροσωπεύει την τρέχουσα ημερομηνία και ώρα με ακρίβεια εκατοστού του δευτερολέπτου. Έτσι, μπορεί να θεωρηθεί ότι ο αριθμός αυτός είναι τυχαίος, αφού κάθε φορά που γίνεται `import` το module `random`, το seed θα είναι διαφορετικό. Αν χρησιμοποιούνταν το ίδιο seed, η αλληλουχία των ψευδοτυχαίων αριθμών που θα επέστρεφαν οι συναρτήσεις τυχαίων αριθμών, θα ήταν πάντα ίδιες. Σε περίπτωση που αυτό είναι επιθυμητό, μπορεί να επιτευχθεί με την κλήση της συνάρτησης seed του module `random`, με την οποία μπορεί να καθοριστεί από τον προγραμματιστή η τιμή του seed. Π.χ., η παρακάτω εντολή θέτει το seed ίσο με 10:

```
random.seed(10)
```

Κάθε φορά που η τιμή του seed καθορίζεται στον ίδιο αριθμό (π.χ., εδώ 10), η αλληλουχία των ψευδοτυχαίων αριθμών που παράγουν οι συναρτήσεις του module `random` θα είναι η ίδια.

Προγραμματιστική εφαρμογή: Τυχαίος περίπατος (Random walk):

Έστω μια τετραγωνική περιοχή $(2n+1)$-επί-$(2n+1)$ με κέντρο στην αρχή των αξόνων που έχει καλυφθεί από τετράγωνα 1-επί-1. Ένα ρομπότ τοποθετείται στο κεντρικό τετράγωνο και προχωράει πηγαίνοντας από τετράγωνο σε τετράγωνο σύμφωνα με κάποιους πολύ απλούς κανόνες. Συγκεκριμένα, αν (x_c, y_c) είναι η τρέχουσα θέση του ρομπότ, τότε προχωράει σε κάποια από τις 4 γειτονικές κατευθύνσεις ((x_c, y_c+1), (x_c+1, y_c), (x_c, y_c-1) και (x_c-1, y_c)) με πιθανότητα 0.25 (δεν κινείται διαγώνια). Η πορεία του ρομπότ συνεχίζεται μέχρι να φτάσει σε κάποιο από τα ακραία τετράγωνα. Η προσομοίωση της πορείας αυτής ονομάζεται τυχαίος περίπατος (random walk). Π.χ.:



<u>*Το πρόβλημα:*</u> Για δεδομένο n, ποιος είναι ο μέσος αριθμός βημάτων που απαιτείται για να φτάσει

το ρομπότ στο σύνορο;

Μια συνάρτηση που υλοποιεί έναν τυχαίο περίπατο του ρομπότ είναι η ακόλουθη:

```
def random_walk(n):
    import random
    x = 0
    y = 0
    steps = 0
    # Ο τυχαίος περίπατος:
    while abs(x)<n and abs(y)<n:  # όσο δεν έχει φτάσει στην άκρη
        r = random.random()
        if r < 0.25:
            y += 1   # Πήγαινε επάνω
        elif r < 0.5:
            x += 1   # Πήγαινε δεξιά
        elif r < 0.75:
            y -= 1   # Πήγαινε κάτω
        else:
            x -= 1   # Πήγαινε αριστερά
        steps += 1
    return steps
```

Η συνάρτηση δέχεται το μέγεθος της τετράγωνης περιοχής, και επιστέφει το πλήθος των βημάτων που απαιτήθηκαν μέχρι την ολοκλήρωση του τυχαίου περιπάτου, δηλαδή μέχρι το ρομπότ να φτάσει σε κάποιο ακραίο τετράγωνο της περιοχής. Μία άλλη έκδοση της συνάρτησης που αντί για τη `random` χρησιμοποιεί τη `randint`, θα μπορούσε να ήταν η εξής:

```
def random_walk2(n):
    import random
    x = 0
    y = 0
    steps = 0
    # Ο τυχαίος περίπατος:
    while abs(x)<n and abs(y)<n:  # όσο δεν έχει φτάσει στην άκρη
        r = random.randint(1,4)
        if r == 1:
            y += 1   # Πήγαινε επάνω
        elif r == 2:
            x += 1   # Πήγαινε δεξιά
        elif r == 3:
            y -= 1   # Πήγαινε κάτω
        else:
            x -= 1   # Πήγαινε αριστερά
        steps += 1
    return steps
```

Η συνάρτηση αυτή θα χρησιμοποιηθεί για να υλοποιηθούν διάφοροι τυχαίοι περίπατοι για διάφορα μεγέθη *n*, και να υπολογιστεί ο μέσος όρος των βημάτων που απαιτούνται για κάθε μέγεθος *n* για την ολοκλήρωση των τυχαίων περιπάτων. Παρακάτω παρουσιάζεται το αντίστοιχο πρόγραμμα, για μεγέθη *n* = 5, 10, 15, 20, 25, 30, 35, 40, 45 και 50. Για κάθε μέγεθος, υλοποιούνται 1000 τυχαίοι περίπατοι ώστε να υπολογιστεί ο μέσος όρος των απαιτούμενων βημάτων.

```
def main():
```

```
    nTrials = 1000  # πλήθος δοκιμών για κάθε μέγεθος τετραγώνου
    print('Αποτελέσματα βασισμένα σε', nTrials, 'δοκιμές.')
    print('Μέγεθος \tM.O. βημάτων')
    for n in range(5,51,5):  # για διάφορα μεγέθη τετραγώνων
        steps = 0
        for k in range(nTrials):
            steps += random_walk(n)
        avg = steps/nTrials
        print(n, '\t\t', format(avg, '8.3f'))
```

Δείγμα εξόδου του προγράμματος:

```
Αποτελέσματα βασισμένα σε 1000 δοκιμές.
Μέγεθος    M.O. βημάτων
5             29.544
10           116.404
15           265.860
20           475.829
25           743.069
30          1070.013
35          1432.832
40          1934.235
45          2345.195
50          3006.419
```

Δημιουργία Module

Οι συναρτήσεις που δημιουργούνται από τον προγραμματιστή, μπορούν και αυτές (όπως οι ενσωματωμένες συναρτήσεις της Python) να αποθηκεύονται σε συγκεκριμένα module, τα οποία στη συνέχεια μπορούν να γίνονται import από άλλα προγράμματα. Ουσιαστικά ένα module δεν είναι τίποτε άλλο παρά ένα πρόγραμμα Python το οποίο περιέχει συναρτήσεις. Άρα, ένα module αποθηκεύεται σε ένα αρχείο:

όνομα_module.py

Το αρχείο αυτό του module θα πρέπει να βρίσκεται στον ίδιο υποκατάλογο (φάκελο) στον οποίο θα βρίσκεται και το πρόγραμμα που θα το κάνει `import`. Σε αυτή την περίπτωση, στο πρόγραμμα θα πρέπει να υπάρχει η εντολή:

`import` *όνομα_module*

και οι συναρτήσεις του module θα πρέπει να καλούνται όπως και στην περίπτωση των έτοιμων module της Python, δηλαδή:

όνομα_module.όνομα_συνάρτησης()

Παράδειγμα:

(αρχείο circle.py)

```
# Το module circle περιέχει συναρτήσεις που σχετίζονται με κύκλους.
import math
# Η συνάρτηση area δέχεται ακτίνα κύκλου και επιστρέφει το εμβαδόν του.
def area(radius):
    return math.pi * radius**2
# Η συνάρτηση δέχεται ακτίνα κύκλου και επιστρέφει την περιφέρειά του.
def circumference(radius):
    return 2 * math.pi * radius
```

(αρχείο geometry.py)

```
import circle

radius = float(input('Δώσε την ακτίνα του κύκλου: '))
print('Το εμβαδόν είναι', circle.area(radius) )
print('Η περιφέρεια είναι', circle.circumference(radius) )
```

Τοπικές και καθολικές μεταβλητές

Μια *τοπική (local) μεταβλητή* δημιουργείται στο εσωτερικό μιας συνάρτησης και δεν είναι προσπελάσιμη από εντολές που βρίσκονται έξω από τη συνάρτηση.

Π.χ.,

```
# Ορισμός της συνάρτησης main
def main():
    get_name()
    print('Γεια σου', name)   # Αυτό προκαλεί σφάλμα!

# Ορισμός της συνάρτησης get_name
def get_name():
    name = input('Πώς σε λένε; ')

main()
```

Εμβέλεια (scope) μιας μεταβλητής ονομάζεται το τμήμα του προγράμματος από το οποίο είναι προσπελάσιμη η μεταβλητή αυτή.

Προσοχή: Μια τοπική μεταβλητή δεν είναι προσπελάσιμη από κώδικα που βρίσκεται μέσα στη συνάρτηση σε σημείο *πριν* από τη δημιουργία τής μεταβλητής.

Σε αντιδιαστολή με τις τοπικές μεταβλητές, οι *καθολικές (global) μεταβλητές* είναι αυτές που δημιουργούνται με μία εντολή εκχώρησης που βρίσκεται έξω από όλες τις συναρτήσεις ενός προγράμματος. Οι καθολικές μεταβλητές είναι προσπελάσιμες από οποιαδήποτε εντολή ενός προγράμματος.

Παράδειγμα 1:

```
# Δημιουργία καθολικής μεταβλητής.
my_value = 10

# Η συνάρτηση show_value εμφανίζει
# την τιμή μιας καθολικής μεταβλητής.
def show_value():
     print(my_value)

# Κλήση της συνάρτησης show_value()
show_value()
```

Παράδειγμα 2:

```
x = 50  # καθολική μεταβλητή
def func(x):
    print('Το x είναι', x)
    x=2
    print('Άλλαξα το τοπικό x σε', x)
func(x)
print('Το x είναι ακόμα', x)
```

Έξοδος:

```
Το x είναι 50
Άλλαξα το τοπικό x σε 2
Το x είναι ακόμα 50
```

Παράδειγμα 3:

Τι λάθος έχει το ακόλουθο πρόγραμμα;

```
x = 50  # καθολική μεταβλητή
def func():
    print('Το x είναι', x)
    x=2
    print('Άλλαξα το τοπικό x σε', x)
func()
print('Το x είναι ακόμα', x)
```

Κατά τη διερμηνεία, προκύπτει το εξής σφάλμα:

```
Traceback (most recent call last):
  File "/home/dinos/Desktop/test.py", line 7, in <module>
    func()
  File "/home/dinos/Desktop/test.py", line 3, in func
    print('Το x είναι', x)
UnboundLocalError: local variable 'x' referenced before assignment
```

Το πρόβλημα παρουσιάζεται επειδή η εντολή `x=2` δημιουργεί μία τοπική μεταβλητή μέσα στη συνάρτηση `func` με το όνομα `x`, το οποίο είναι ίδιο με το όνομα της καθολικής μεταβλητής. Σε αυτή την περίπτωση, η τοπική μεταβλητή έχει προτεραιότητα στο όνομα μέσα στη συνάρτηση. Επομένως, η πρώτη εντολή της συνάρτησης `print('Το x είναι', x)` αναφέρεται πλέον στην τοπική μεταβλητή, η οποία όμως δεν έχει πάρει ακόμα τιμή σε αυτό το σημείο.

Εάν *δεν υπήρχε* η εντολή `x=2`, το πρόγραμμα θα μεταγλωττιζόταν, θεωρώντας ότι η μεταβλητή `x` μέσα στη συνάρτηση είναι η καθολική μεταβλητή `x` (*Παράδειγμα 3α*):

```
x = 50  # καθολική μεταβλητή
def func():
    print('Το x είναι', x)
    # x=2
    print('Άλλαξα το τοπικό x σε', x)
func()
print('Το x είναι ακόμα', x)
```

οπότε θα εκτύπωνε:

```
Το x είναι 50
Άλλαξα το τοπικό x σε 50
Το x είναι ακόμα 50
```

Επίσης, το πρόγραμμα θα λειτουργούσε κανονικά εάν δεν υπήρχε η εντολή `print('Το x είναι', x)` (*Παράδειγμα 3β*):

```
x = 50  # καθολική μεταβλητή
def func():
    # print('Το x είναι', x)
    x=2
    print('Άλλαξα το τοπικό x σε', x)
func()
print('Το x είναι ακόμα', x)
```

οπότε θα εκτύπωνε:

```
Άλλαξα το τοπικό x σε 2
Το x είναι ακόμα 50
```

Τα προβλήματα που παρουσιάζονται στο Παράδειγμα 3 οφείλονται στο γεγονός ότι μία συνάρτηση δε μπορεί με αυτόν τον τρόπο να αλλάξει την τιμή μιας καθολικής μεταβλητής. Γι αυτό και η εντολή `x=2` δημιουργεί τοπική μεταβλητή (με το ίδιο όνομα με την καθολική μεταβλητή). Για να μπορεί μία συνάρτηση να μεταβάλει την τιμή μιας καθολικής μεταβλητής, θα πρέπει η καθολική μεταβλητή να δηλωθεί μέσα στη συνάρτηση, με την εντολή `global`, ως εξής:

```
x = 50  # καθολική μεταβλητή
def func():
    global x
    print('Το x είναι', x)
    x=2
    print('Άλλαξα το τοπικό x σε', x)
func()
print('Το x είναι ακόμα', x)
```

Το πρόγραμμα αυτό εκτυπώνει:

```
Το x είναι 50
Άλλαξα το τοπικό x σε 2
Το x είναι ακόμα 2
```

Σημείωση 1: Η χρήση των καθολικών μεταβλητών καλό είναι να αποφεύγεται και να προτιμάται η χρήση τοπικών μεταβλητών και η μεταβίβασή τους ως ορίσματα σε άλλες συναρτήσεις που τις χρησιμοποιούν.

Σημείωση 2: Η χρήση καθολικών μεταβλητών χωρίς τη δήλωσή τους ως `global` μέσα σε συναρτήσεις, ουσιαστικά δημιουργεί “καθολικές σταθερές” σε ένα πρόγραμμα, κάτι που πολλές φορές είναι χρήσιμο.