Εμβέλεια Μεταβλητών (Scope) & Λίστες

Local vs Global

ΧΡΗΣΤΟΣ ΜΑΒΟΓΛΟΥ

Python 2: Εμβέλεια Μεταβλητών (Scope) & Λίστες

1. Θεωρητικό Υπόβαθρο: Local vs Global

Στην Python, η "εμβέλεια" (scope) καθορίζει πού είναι ορατή μια μεταβλητή.

- **Global (Καθολική):** Μια μεταβλητή που **ορίζεται στο κυρίως σώμα του προγράμματος** (έξω από συναρτήσεις), **είναι ορατή παντού**.
- **Local (Τοπική):** Μια μεταβλητή που ορίζεται *μέσα* σε μια συνάρτηση. Ζει μόνο όσο τρέχει η συνάρτηση και χάνεται μετά.

---

2. Απλές Μεταβλητές (Integer, Float, String, Boolean)

Integer, Float, String, Boolean είναι **Immutable (Αμετάβλητα)**.

Περίπτωση Α: Ανάγνωση (Reading)

Μπορούμε να δούμε την τιμή μιας global μεταβλητής μέσα σε συνάρτηση χωρίς πρόβλημα.

Python

```
x = 10  #x is a Global variable

def show_x():
    print "Μέσα στη συνάρτηση:", x  # Βλέπει την Global

show_x()

#Έξοδος:
Μέσα στη συνάρτηση: 10
```

Περίπτωση Β: Τροποποίηση (Writing / Shadowing)

Αν προσπαθήσουμε να αλλάξουμε την τιμή (x = ...) μέσα στη συνάρτηση, η Python δημιουργεί αυτόματα μια **νέα τοπική** μεταβλητή με το ίδιο όνομα. Η Global x μένει ανέπαφη.

Python

```
x = 10  # Global

def change_x():
    x = 5  # Δημιουργείται ΝΕΑ τοπική μεταβλητή 'x'
    print "Local x:", x

change_x()
print "Global x:", x

# Έξοδος:
Local x: 5
Global x: 10  <-- Η αρχική δεν άλλαξε!
```

Περίπτωση Γ: Η εντολή global μέσα στη συνάρτηση

Αν θέλουμε οπωσδήποτε να αλλάξουμε την έξω μεταβλητή, πρέπει να το δηλώσουμε.

```
count = 0


def increment():
    global count  # "Πάρε την έξω μεταβλητή, μην φτιάξεις τοπική"
    count = count + 1


increment()
print count
# Έξοδος: 1
```

---

3. Λίστες: Ο Κανόνας της "Αναφοράς"

Οι λίστες συμπεριφέρονται διαφορετικά γιατί είναι **Mutable** (μεταβαλλόμενες). Όταν περνάμε μια λίστα σε συνάρτηση, περνάμε την "αναφορά" (reference) στη μνήμη.

Κανόνας:

1. Αν αλλάξουμε τα **περιεχόμενα** της λίστας (append, pop, list[0]=...), η αλλαγή φαίνεται παντού.
2. Αν κάνουμε **νέα ανάθεση** στη λίστα (list = ...), σπάμε τον δεσμό και φτιάχνουμε τοπική λίστα.

Παράδειγμα 1: Αλλαγή Περιεχομένου (Επηρεάζει την Global)

```
numbers = [1, 2, 3]


def add_number(lista):
    lista.append(4)
    lista[0] = 99


add_number(numbers)
print numbers


# Έξοδος: [99, 2, 3, 4]  <-- Η λίστα άλλαξε μόνιμα!
```

Σημείωση: Εδώ ΔΕΝ χρειάζεται η εντολή global.

Παράδειγμα 2: Νέα Ανάθεση (ΔΕΝ επηρεάζει την Global)

```
numbers = [1, 2, 3]

def reset_list(lista):
    lista = []    # Το '=' φτιάχνει νέα τοπική μεταβλητή
    lista.append(5)  # Προσθέτει στη νέα τοπική λίστα
    print "Μέσα:", lista

reset_list(numbers)
print "Έξω:", numbers
```

Έξοδος:

Μέσα: [5]

Έξω: [1, 2, 3] <-- Η αρχική λίστα έμεινε ανέπαφη

---

4. Μεθοδολογία Σκέψης

Όταν βλέπετε κώδικα, κάντε τις εξής ερωτήσεις:

1. **Είναι απλή μεταβλητή (integer, float, boolean, string);**
    - Υπάρχει global x στην αρχή της συνάρτησης;
        - **ΝΑΙ:** Αλλάζει η έξω μεταβλητή.
        - **ΟΧΙ:** Αν υπάρχει x = ..., είναι νέα τοπική μεταβλητή.
2. **Είναι Λίστα;**
    - Χρησιμοποιούμε μεθόδους (.append(), .pop()) ή L[0]=...);
        - **ΝΑΙ:** Η αλλαγή είναι μόνιμη στην Global λίστα.
    - Χρησιμοποιούμε το ίσον (L = [...] ή L = []);
        - **ΝΑΙ:** Δημιουργείται νέα τοπική λίστα, η Global δεν αλλάζει.

Ασκήσεις

Άσκηση 1: Η "παγίδα" της τοπικής μεταβλητής

```
x = 50

def alage_timi():
    x = 100
    print "Inside:", x

alage_timi()
print "Outside:", x
```

Ερώτηση: Τι θα τυπωθεί;

Άσκηση 2: Η εντολή global

```
score = 0

def kerdises():
    global score
    score = 10

kerdises()
print score
```

Ερώτηση: Τι θα τυπώσει το πρόγραμμα;

Άσκηση 3: Λίστα και .append()

```
noumera = [1, 2]

def prosthiki(lista):
    lista.append(3)

prosthiki(noumera)
print noumera
```

Ερώτηση: Η λίστα noumera άλλαξε ή έμεινε ίδια;

Άσκηση 4: Λίστα και το σύμβολο =

```
colors = ["red", "blue"]

def reset_colors(lista):
    lista = ["green", "yellow"]
    lista.append("black")

reset_colors(colors)
print colors
```

Ερώτηση: Τι περιέχει τελικά η λίστα colors;

Άσκηση 5: Ανάγνωση vs Εγγραφή

```
a = 5

def praxis():
    y = a + 5
    print y

praxis()
print a
```

Ερώτηση: Τι θα τυπώσει η εντολή print a στο τέλος;

Μεταβλητή ή λίστα δίνεται ως παράμετρος.

Στην Python, όταν περνάς μια μεταβλητή ως παράμετρο σε συνάρτηση, δεν περνάς την ίδια τη μεταβλητή (το κουτί), αλλά **μια αναφορά στο αντικείμενο** (ένα "καρτελάκι" που δείχνει την τιμή στη μνήμη).

1. Αμετάβλητοι Τύποι – Immutable (Integer, String, Float, Boolean)

Όταν περνάς έναν έναν αμετάβλητο τύπο ως παράμετρο, η συνάρτηση φτιάχνει ένα **τοπικό όνομα** που δείχνει στην ίδια τιμή.

Αν προσπαθήσεις να αλλάξεις την τιμή μέσα στη συνάρτηση, η Python αναγκαστικά φτιάχνει ένα **νέο αντικείμενο** και βάζει το τοπικό όνομα να δείχνει εκεί.

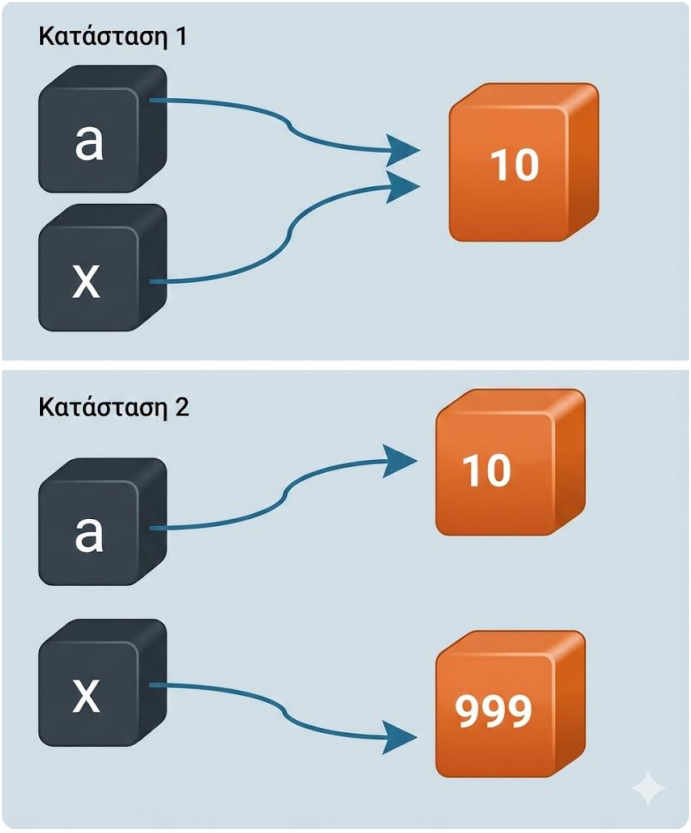
Λειτουργεί σαν **"Φωτοτυπία"**. Η συνάρτηση παίρνει την τιμή, αλλά ό,τι αλλαγή και να κάνει στην παράμετρο, η αρχική μεταβλητή έξω δεν επηρεάζεται.

Συμπέρασμα: Η έξω απλή μεταβλητή **δεν αλλάζει ποτέ** μέσα στη συνάρτηση (εκτός και αν έχει δηλωθεί global).

```
def change_number(x):
    print "Inside (before):", x

    x = 999
    print "Inside (after):", x

a = 10
change_number(a)
print "Outside:", a
```

Κατάσταση 1:

```
def change_number(x):
    …

a = 10
change_number(a)
…
```



Κατάσταση 2:

```
def change_number(x):
    …
    x = 999
    …

a = 10
change_number(a)
…
```

2. Μεταβλητοί Τύποι - Immutable (Λίστες)

Όταν περνάει μια λίστα, η παράμετρος (τοπικό όνομα) δείχνει **στην ίδια ακριβώς λίστα** στη μνήμη με την εξωτερική μεταβλητή.

Λειτουργεί σαν **"Κλειδί Θυρίδας"**. Δίνουμε στη συνάρτηση πρόσβαση στην ίδια λίστα μνήμης.

1. **Τροποποίηση (**append**):** Η συνάρτηση βάζει πράγματα στη θυρίδα μας. **Η αλλαγή μένει.**
2. **Ανάθεση (=):** Η συνάρτηση πετάει το κλειδί μας και φτιάχνει δική της θυρίδα. **Η αλλαγή ΔΕΝ μένει.**

Έχουμε δύο σενάρια:

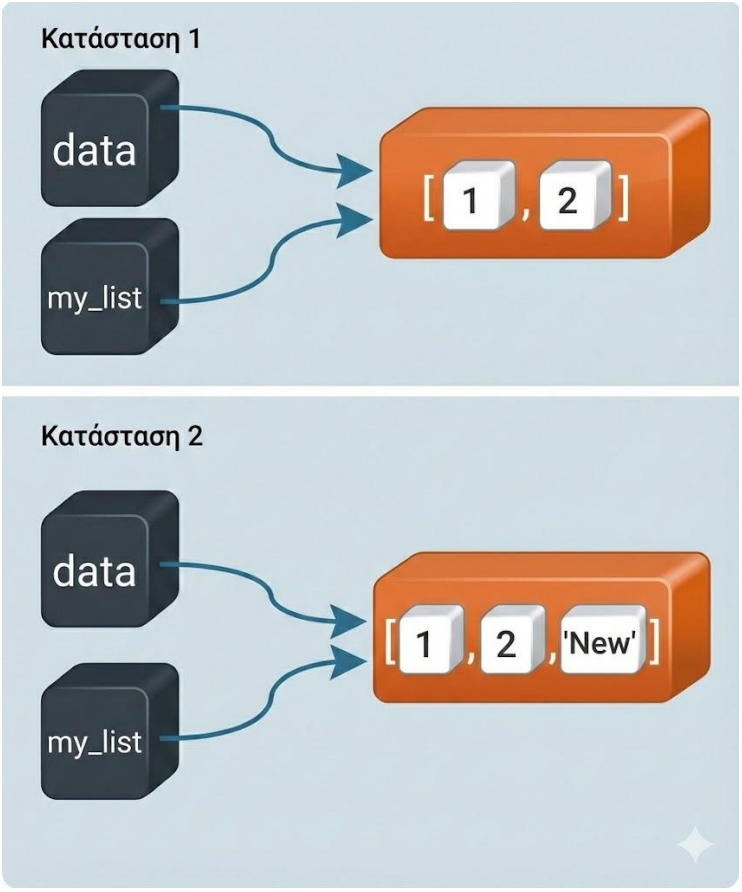
Α. Τροποποίηση (Mutation) -> Αλλάζει και η έξω!
Αν χρησιμοποιήσετε εντολές που αλλάζουν τα περιεχόμενα της λίστας: (.append(), .pop(), L[0]=...), τότε επειδή και τα δύο ονόματα δείχνουν την ίδια λίστα, η αλλαγή φαίνεται και έξω.

```
def add_item(my_list):
    my_list.append("New")
data = [1, 2]
add_item(data)
print data
```

Τυπώνει: [1, 2, 'New'] <-- ΑΛΛΑΞΕ!

Κατάσταση 1:

```
def add_item(my_list):
    …


data = [1, 2]
add_item(data)
…
```



Κατάσταση 2:

```
def add_item(my_list):
    my_list.append("New")


data = [1, 2]
add_item(data)
…
```

Β. Επαν-ανάθεση (Reassignment) -> ΔΕΝ αλλάζει η έξω!

Αν μέσα στη συνάρτηση ξαναοριστεί η τιμή της λίστας με τον τελεστή εκχώρησης:

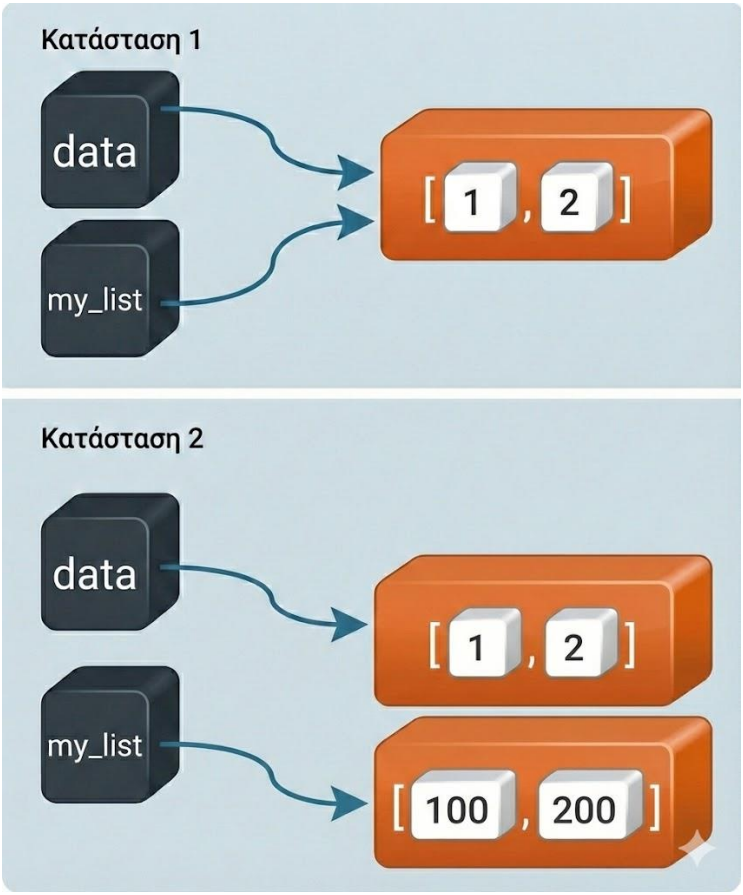
λίστα = άλλη_τιμή

τότε κόβεται ο δεσμός με την αρχική τιμή. Η τοπική παράμετρος σταματάει να δείχνει στην αρχική λίστα και δείχνει στη νέα. Η αρχική μένει ανέπαφη.

```
def break_link(my_list):
    my_list = [100, 200]
    print "Inside:", my_list

data = [1, 2]
break_link(data)
print "Outside:", data
```

Κατάσταση 1:

```
def break_link(my_list):
    …


data = [1, 2]
break_link(data)
…
```



Κατάσταση 2:

```
def break_link(my_list):
    my_list = [100, 200]
    …


data = [1, 2]
break_link(data)
…
```

Συνοπτικός Πίνακας

Τύπος Δεδομένων	Τι κάνουμε μέσα στη συνάρτηση;	Επηρεάζεται η έξω μεταβλητή;
Ακέραιος / String	x = 5 (Ανάθεση)	**ΟΧΙ**
Λίστα	L.append(5) (Τροποποίηση)	**ΝΑΙ**
Λίστα	L[0] = 5 (Τροποποίηση στοιχείου)	**ΝΑΙ**
Λίστα	L = [1, 2] (Νέα Ανάθεση)	**ΟΧΙ**

Αν οι παράμετροι σε μια συνάρτηση είναι οι:

- **Interger, Float, String, Boolean:** Είναι σαν να στέλνεις σε κάποιον μια **φωτοτυπία** του εγγράφου σου. Ό,τι και να γράψει πάνω, το δικό σου πρωτότυπο δεν αλλάζει.
- **Λίστες:** Είναι σαν να δίνεις σε κάποιον το **κλειδί** για τη θυρίδα σου.
    - Αν βάλει κάτι μέσα (append), το βρίσκεις κι εσύ.
    - Αν πετάξει το κλειδί που του έδωσες και αγοράσει μια δική του θυρίδα (=), τότε ό,τι βάλει στη δική του δεν αφορά εσένα.

Ασκήσεις Κατανόησης

Προσπαθήστε να προβλέψετε την έξοδο πριν εκτελέσετε τον κώδικα.

Άσκηση 1

```
k = 10
def f1():
    k = 20
f1()
print k
```

Άσκηση 2

```
lista = [1]
def f2(x):
    x.append(2)
f2(lista)
print lista
```

Άσκηση 3

```
score = 50
def f3():
    global score
    score = 100
f3()
print score
```

Απαντήσεις:

1. **10**
2. **[1, 2]**
3. **100**