ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Η/Υ
σε Python

Διάλεξη 7: Συναρτήσεις

Μάρκος Ζάμπογλου, mzampoglou@aegean.gr

Συνάρτηση (function)

- Ένα *τμήμα προγράμματος*
  - *"υποπρόγραμμα"*
- Έχει **εισόδους**, **εξόδους**, και **κώδικα** ο οποίος υλοποιεί μια **λογική**

- **Κλήση συνάρτησης:**

```
έξοδος1,έξοδος2,... = όνομα(είσοδος1,είσοδος2,...)
```

Παραδείγματα συναρτήσεων
που γνωρίζουμε ήδη (1/2)

`abs()`:

- **είσοδος**: μια αριθμητική τιμή
- **έξοδος**: μια αριθμητική τιμή ίδιου τύπου με την είσοδο
- **κώδικας**: (υποθέτουμε) μια δομή επιλογής που αντιστρέφει το πρόσημο αν είναι αρνητικό

`print()`:

- **είσοδοι**: ένα οποιοδήποτε πλήθος εκφράσεων
- **έξοδος**: `None`
  - αν γράψουμε `a = print("γεια","σου")`, το `a` θα είναι `None`
- **κώδικας**: εντολές που εμφανίζουν στην οθόνη τα αποτελέσματα των εκφράσεων

Παραδείγματα συναρτήσεων
που γνωρίζουμε ήδη (2/2)

`pow()`:

- **είσοδος**: δύο αριθμητικές τιμές
- **έξοδος**: μια αριθμητική τιμή της οποίας ο τύπος προκύπτει από τις εισόδους
  - (`int` αν και οι δυο είναι `int`, `float` αν η μια είναι `int` και η άλλη `float`, κλπ)
- **κώδικας**: (υποθέτουμε) μια επαναληπτική δομή που υπολογίζει το γινόμενο της πρώτης εισόδου τόσες φορές όσο η δεύτερη είσοδος

`input()`:

- **είσοδος**: ένα string
- **έξοδος**: ένα string
- **κώδικας**: εντολές που εμφανίζουν στην οθόνη το `string` της εισόδου, διαβάζουν από το χρήστη ένα `string` και το επιστρέφουν

Δημιουργία δικών μας συναρτήσεων

- Μπορούμε να δημιουργήσουμε δικές μας συναρτήσεις στην Python
- Γιατί;
  - Ευκολότερη οργάνωση/συντήρηση κώδικα
  - Αποφυγή επανάληψης των ίδιων εντολών
  - Μεταφερσιμότητα κώδικα σε πολλαπλές εφαρμογές
  - Αφαιρετικότητα (abstraction): δε χρειάζεται να γνωρίζουν οι πάντες **πώς** δουλεύει μια συνάρτηση, μόνο το τι πετυχαίνει

Ορισμός συνάρτησης

```
def <όνομα_συνάρτησης>(είσοδος1, είσοδος2, …):
        <εντολές συνάρτησης>
        return έξοδος1, έξοδος2, …
```

Η εντολή `return` τερματίζει την εκτέλεση της συνάρτησης. Αν υπάρχουν εντολές που θα εκτελούνταν μετά από αυτήν, οι εντολές αυτές αγνοούνται.

Δήλωση & κλήση συναρτήσεων: παράδειγμα

```
def my_first_function(input1,input2):
  b = input1 + input2
  return b

print(my_first_function(4,5))
```

Συνάρτηση (ορισμός)

Κυρίως πρόγραμμα (κλήση συνάρτησης)

```
9
```

Η εντολή `return`

```
def my_abs(num):
  if num<0:
    return -num
  return num
```

Δε χρειάζεται να βάλω **`else`**. Αν η **`num`** είναι αρνητικός αριθμός, θα εκτελεστεί η πρώτη **`return`** και η συνάρτηση θα σταματήσει πριν εκτελεστεί η δεύτερη.

(Παρόλα αυτά δε θα υπήρχε πρόβλημα αν είχα βάλει **`else`**).

Λειτουργία συνάρτησης

- Κατά την εκτέλεση του κυρίως προγράμματος, όταν εντοπιστεί η δήλωση μιας συνάρτησης, ο κώδικας **δεν εκτελείται**
  - Απλώς *δεσμεύεται* το όνομα, και *κρατιέται ο κώδικας της στη μνήμη*
- Όταν κληθεί μια υπάρχουσα συνάρτηση, η εκτέλεση του κυρίως προγράμματος **διακόπτεται** και ξεκινάει η εκτέλεση της συνάρτησης
  - Όταν αυτή ολοκληρωθεί, οι έξοδοι επιστρέφονται στο κυρίως πρόγραμμα
  - Στη συνέχεια, η εκτέλεση του κυρίως προγράμματος **συνεχίζεται**
- **Αν η συνάρτηση κληθεί πριν δηλωθεί, προκαλείται `NameError` καθώς η Python δεν αναγνωρίζει το όνομα**

Παράδειγμα 7.1

Να γραφεί συνάρτηση σε Python που θα δέχεται τις πλευρές ενός ορθογώνιου τριγώνου και θα επιστρέφει το μήκος της υποτείνουσας. Αν οι πλευρές δεν είναι θετικές, να επιστρέφει **`None`**.

Να κληθεί η συνάρτηση με εισόδους **3** και **4** και να εμφανιστεί το αποτέλεσμα.

Παράδειγμα 7.1

Να γραφεί συνάρτηση σε Python που θα δέχεται τις πλευρές ενός ορθογώνιου τριγώνου και θα επιστρέφει το μήκος της υποτείνουσας. Αν οι πλευρές δεν είναι θετικές, να επιστρέφει **None**.

Να κληθεί η συνάρτηση με εισόδους **3** και **4** και να εμφανιστεί το αποτέλεσμα.

```
def ypot(a,b):
    if a>0 and b>0:
        y=(a**2+b**2)**(1/2)
    else:
        y=None
    return y
print(ypot(3,4))
```

Οι συναρτήσεις ως αντικείμενα 🤓

- Στην Python, οι συναρτήσεις είναι **αντικείμενα**
  - ...που έχουν την επιπλέον ιδιότητα να είναι εκτελέσιμα
- Τι σημαίνει αυτό;
  - Ότι πέρα από την ανάθεση του αποτελέσματος συνάρτησης που είδαμε ήδη (`a=abs(-3)`)...
  - ...μπορούμε να κάνουμε ανάθεση της ίδιας της συνάρτησης: `a=abs`
- Πώς λειτουργεί αυτό;
  - **Βλ κώδικα.**

Παρότι δε θα χρησιμοποιήσουμε αυτή τη δυνατότητα, είναι σημαντικό να ξέρουμε τι συμβαίνει όταν *ξεχνάμε τις παρενθέσεις*.

Συναρτήσεις που καλούν συναρτήσεις (1/2)

- Αφού μέσα στον κώδικα μιας συνάρτησης μπορώ να χρησιμοποιήσω ό,τι εντολές θα χρησιμοποιούσα σε ένα πρόγραμμα, μπορώ να χρησιμοποιήσω και συναρτήσεις:

```
def square_of_list(list_in):
    list_out=[pow(n,2) for n in list_in]
    return(list_out)

print(square_of_list([2,4,6,1]))
```

```
[4, 16, 36, 1]
```

Συναρτήσεις που καλούν συναρτήσεις (2/2)

- Αυτό δεν αφορά μόνο έτοιμες συναρτήσεις όπως η **pow()** αλλά και συναρτήσεις που έχω ορίσει εγώ:

```
def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l in
                                  list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                        [2,-2],[11,12,13]]))
[[4, 16, 25, 9], [4, 4], [121, 144, 169]]
```

Παράδειγμα 7.2

Να γραφεί συνάρτηση σε Python που θα δέχεται μια λίστα και θα επιστρέφει τη μέση τιμή των στοιχείων της. Υποθέστε ότι τα στοιχεία της είναι αριθμητικά.

Καλέστε την με είσοδο τη λίστα `[2, 6, 19]` και εμφανίστε το αποτέλεσμα.

Παράδειγμα 7.2

Να γραφεί συνάρτηση σε Python που θα δέχεται μια λίστα και θα επιστρέφει τη μέση τιμή των στοιχείων της. Υποθέστε ότι τα στοιχεία της είναι αριθμητικά.

Καλέστε την με είσοδο τη λίστα `[2, 6, 19]` και εμφανίστε το αποτέλεσμα.

```
def l_mean(l):
  return(sum(l)/len(l))
print(l_mean([2, 6, 19]))
```

Εμβέλεια (scope) μεταβλητών

Η εμβέλεια μιας μεταβλητής είναι το τμήμα του προγράμματος όπου η μεταβλητή αυτή *υπάρχει*

- Οι παράμετροι και οι μεταβλητές που ορίζονται μέσα σε μια συνάρτηση δεν έχουν ισχύ έξω από αυτήν
- Ονομάζονται **τοπικές μεταβλητές** (local variables) και έχουν **τοπική εμβέλεια** (local scope)
    - Διαγράφονται από τη μνήμη μόλις ολοκληρωθεί η συνάρτηση
    - Ξαναδημιουργούνται εκ νέου, τοπικά, αν ξανακληθεί

Τοπικές μεταβλητές: παράδειγμα

```
def function1(input1,input2):
  my_sum = input1+input2
  return my_sum

a = 1
b = 2
c = 3
print(function1(b, c))
print(my_sum)   # ERROR: η my_sum δεν υπάρχει εδώ,
                # όπως και οι input1, input2
```

Καθολικές (global) μεταβλητές

- Από την άλλη, οι μεταβλητές και τα αντικείμενα που ορίζουμε στο κυρίως πρόγραμμα, είναι προσβάσιμα και μέσα στις συναρτήσεις
  - Τις αποκαλούμε **καθολικές μεταβλητές** (global variables)
  - Λέμε ότι έχουν **καθολική εμβέλεια** (global scope)
- Αν όμως μέσα στη συνάρτηση ορίσουμε μια τοπική μεταβλητή με ίδιο όνομα με μια καθολική, τότε υπερισχύει η τοπική

Global variables: παράδειγμα

```
def function2(input1, input2):
  my_var = 5
  print(b)                #18
  print(my_var)           #5, όχι 25
  print(input1)           #4, όχι -43
  return input1+input2

b = 18
my_var = 25
input1 = -43
a = function2(4,5)
print(my_var)             #25! Η τοπική εκδοχή έχει ξεχαστεί
```

Προσοχή:

Αν σε κάποιο σημείο μιας συνάρτησης ορίζουμε μια τοπική μεταβλητή, τότε δε μας επιτρέπεται να τη χρησιμοποιήσουμε πριν την ορίσουμε ακόμα κι αν υπάρχει καθολική μεταβλητή με το ίδιο όνομα.

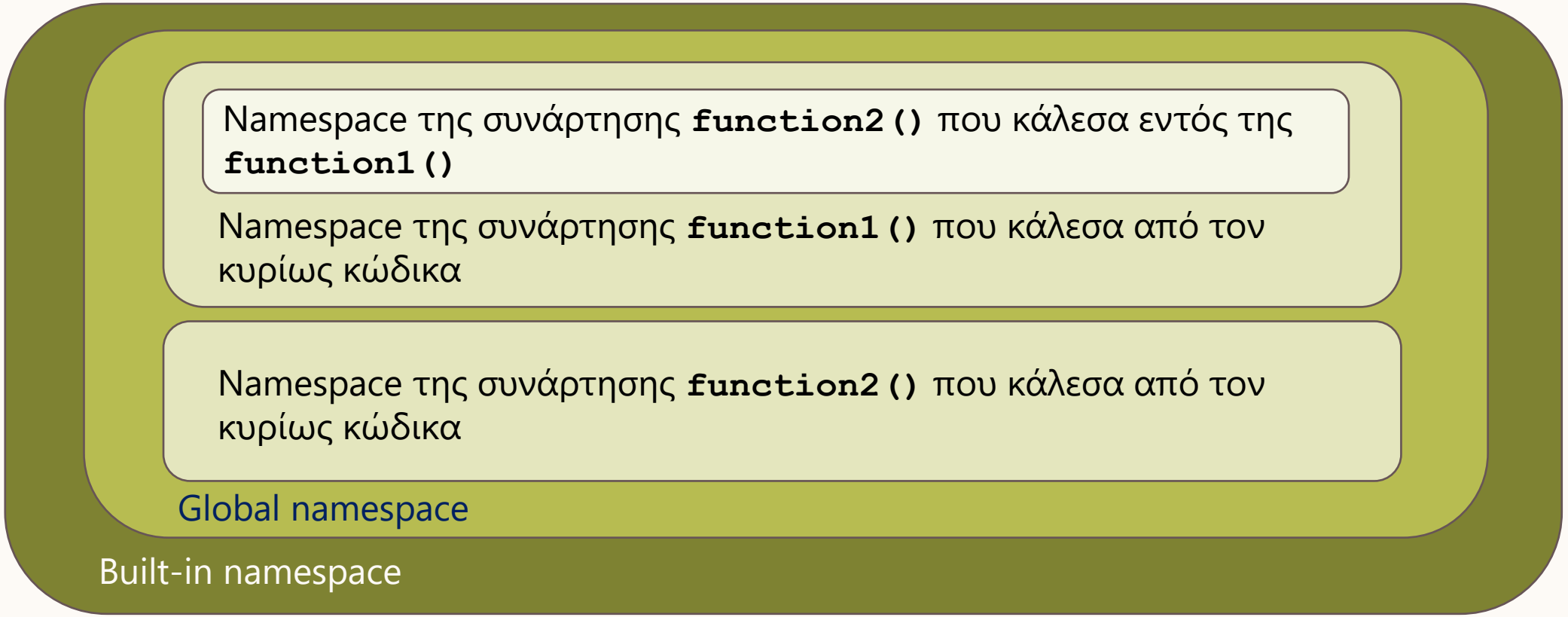
```
def function2(input1, input2):
  print(my_var)                    #Error
  my_var = 5
  return input1+input2
my_var = 25
a = function2(4,5)
```

Χώροι ονομάτων (Namespaces)

Ένας **χώρος ονομάτων (namespace)** είναι ένας χώρος της μνήμης όπου αποθηκεύονται λειτουργικά στοιχεία της Python

- Όταν ο Python Interpreter εκκινεί, φορτώνει αυτομάτως το *ενσωματωμένο (built-in) namespace*
  - Περιέχει αντικείμενα όπως οι συναρτήσεις **`print()`**, **`str()`**, **`len()`**, **`int()`** κλπ
- Όταν φορτώνουμε ένα αρχείο .py ή .ipynb, δημιουργεί το δικό του *καθολικό (global ή module) namespace*
- Όταν ορίζεται μια συνάρτηση μέσα στο αρχείο, δημιουργεί το δικό της *τοπικό (local) namespace*
- Κάθε namespace "βλέπει" όλα τα στοιχεία των "γονεϊκών" namespaces
  - Μια συνάρτηση που γράφω εγώ, έχει πρόσβαση και στη **`str()`** του module και τις καθολικές μεταβλητές.

Ιεραρχία namespaces



Ο κανόνας LEGB

Όταν καλούμε το όνομα ενός αντικειμένου (είτε αφορά συνάρτηση είτε σε άλλο αντικείμενο), η Python το αναζητά με βάση τον κανόνα LEGB

1. **L**ocal: αν βρισκόμαστε μέσα σε συνάρτηση, πρώτα το αναζητά εντός του τοπικού namespace
2. **E**nclosing: στη συνέχεια το αναζητά εντός των τοπικών namespaces των συναρτήσεων που "περικλείουν" τη συνάρτηση (δηλαδή που την έχουν καλέσει)
3. **G**lobal: στη συνέχεια το αναζητά στο καθολικό namespace του αρχείου
4. **B**uilt-in: τέλος το αναζητά στο ενσωματωμένο namespace

Αν δημιουργήσω μια μεταβλητή **`sum`** στο global namespace, τότε με βάση τον κανόνα **LEGB** όταν γράφω **`sum`** η Python εντοπίζει τη μεταβλητή μου πριν συναντήσει την ενσωματωμένη συνάρτηση **`sum()`**. Αν τρέξω την εντολή **`del sum`**, διαγράφεται η μεταβλητή μου –για τον ίδιο ακριβώς λόγο. Τότε, η ενσωματωμένη συνάρτηση **`sum()`** ξαναγίνεται διαθέσιμη.

Τι θα εμφανίσουν;

```
def sum_of_squares(x,y):
  z = x**2 + y**2
  return z
z = 4
print(sum_of_squares(1,3))
print(z)
```

```
4
```

```
def sum_of_string(s_in):
  s = s_in.split(",")
  return (int(s[0])+int(s[1]))
s_in="14"
print(sum_of_string("3,6"))
print(s_in)
```

```
9
14
```

Εξαίρεση 1: Mutables

Όταν σε ένα εσωτερικό namespace τροποποιούμε τμήμα ενός mutable αντικειμένου χωρίς να του αναθέτουμε συνολικά νέα τιμή, τότε η τροποποίηση παραμένει και αφού σβηστεί το namespace

```
def fun1(a):
   a[0]=5
L=[3,1,7]
fun1(L)
print(L)
```

```
[5, 1, 7]
```

Εξαίρεση 2: Η εντολή `global`

Η εντολή **`global`** μας επιτρέπει να μετατρέψουμε μια τοπική μεταβλητή σε καθολική, ώστε οι αναθέσεις που της κάνουμε εντός ενός εσωτερικού namespace να μεταφέρονται στο καθολικό.

```
def fun1(a,b):
  global c
  c = 12
  return(a+b)

c = 5
d=fun1(2,3))
print(c)
12
```

🤓 Εξαίρεση 3: Η εντολή `nonlocal`

Η εντολή **`nonlocal`** εκτελείται πάνω σε μια τοπική μεταβλητή *μιας συνάρτησης που έχει οριστεί μέσα σε μια άλλη συνάρτηση*. Της επιτρέπει να επηρεάζει την τιμή της συνώνυμης μεταβλητής στο εξωτερικό της namespace –αλλά όχι στο global namespace.

```
def fun1(a,b):
  def fun2(c,d):
    nonlocal e
    e=5
  e = 12
  f=fun2(4,5)
  print(e)
  return(a+b)

g = fun1(2,3)
```

Παράδειγμα 7.3

Να γραφεί συνάρτηση σε Python με όνομα **`remove_smaller()`** που θα δέχεται σαν είσοδο μια λίστα και θα της αφαιρεί το μικρότερο στοιχείο της.

Δηλαδή, αν της δώσω την **`[4,7,1,4]`** να παίρνω την **`[4,7,4]`**

Παράδειγμα 7.3

Να γραφεί συνάρτηση σε Python με όνομα **`remove_smaller()`** που θα δέχεται σαν είσοδο μια λίστα και θα της αφαιρεί το μικρότερο στοιχείο της.

Δηλαδή, αν της δώσω την **`[4,7,1,4]`** να παίρνω την **`[4,7,4]`**

```
def remove_smaller(l):
  del(l[l.index(min(l))])

l=[4,7,1,4]
remove_smaller(l)
print(l)
```

```
[4,7,4]
```

Ορισμός συνάρτησης μέσα σε συνάρτηση 🤓

```
def function2(input1, input2):
  #Συνάρτηση που ορίζεται μέσα σε άλλη συνάρτηση
  #και υπάρχει μόνο μέσα σε αυτήν
  def inside_function2(input3, input4):
    print(my_local_var, input1)                    #5, 4
    return 4
  my_local_var = 5
  inside_function2(2,4)
  return input1+input2

a = function2(4,5)
b = inside_function2(2,4) #Error
```

Προσοχή! Μιλάμε για την περίπτωση που **ορίζουμε** μια συνάρτηση μέσα σε μια άλλη συνάρτηση, όχι για την **κλήση** μιας συνάρτησης μέσα από μια άλλη.

Με τον ίδιο τρόπο μπορούμε να προσθέσουμε μια *βιβλιοθήκη* μόνο στο τοπικό namespace μιας συνάρτησης

Κλήσεις μέσα σε συναρτήσεις

- Είπαμε ότι ο κώδικας μιας συνάρτησης μπορεί να περιέχει την κλήση μιας άλλης συνάρτησης
- Στην περίπτωση αυτή, η εκτέλεση της εξωτερικής συνάρτησης διακόπτεται μέχρι να ολοκληρωθεί η εσωτερική συνάρτηση
  - Δημιουργείται μια **στοίβα κλήσεων (call stack)**

Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Η εκτέλεση ξεκινάει

```
__main__
```

Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

Οι συναρτήσεις αποθηκεύονται στη μνήμη

__main__

Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

Οι συναρτήσεις αποθηκεύονται στη μνήμη

__main__

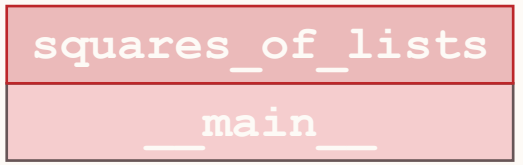
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Η _main_ διακόπτεται και καλείται η squares_of_lists()



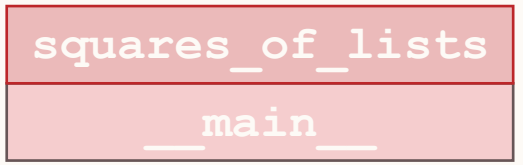
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

Ανατίθεται τιμή στη `list_of_lists`



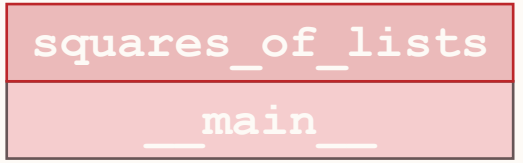
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Ανατίθεται η τιμή `[]` **στη** `list_out`



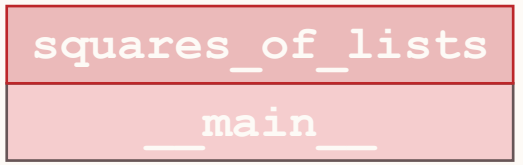
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Ξεκινάει η for



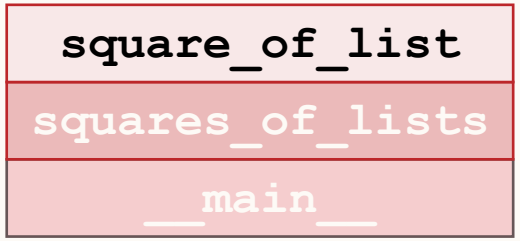
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Η εκτέλεση της `squares_of_lists` **διακόπτεται και ξεκινάει η** `square_of_list`



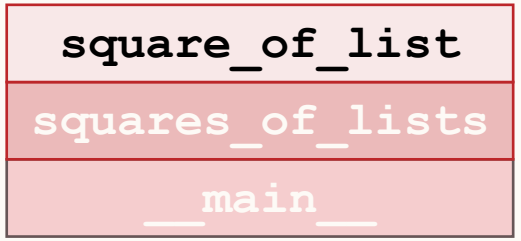
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

Ανατίθεται τιμή στη `list_in`



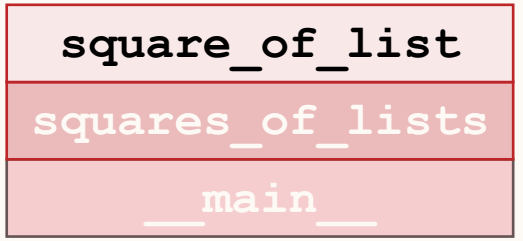
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Ανατίθεται η τιμή `[]` **στη** `list_out`



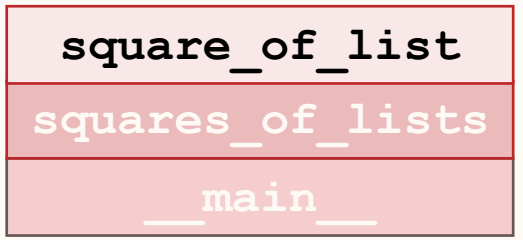
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

Ξεκινάει η for



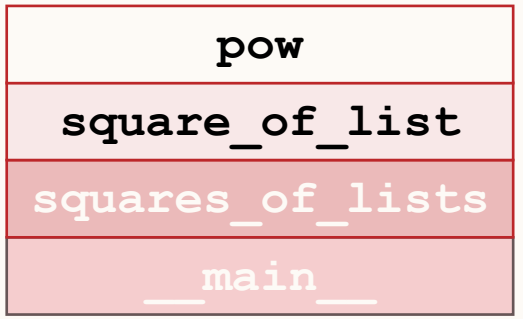
Στοίβα κλήσεων

```
def square_of_list(list_in):
    list_out=[pow(n,2) for n in list_in]
    return(list_out)

def squares_of_lists(list_of_lists):
    list_out = [square_of_list(l) for l
                        in list_of_lists]
    return(list_out)

print(squares_of_lists([[2,4,5,3],
                        [2,-2],[11,12,13]]))
```

Η εκτέλεση της `square_of_list` διακόπτεται και ξεκινάει η πρώτη `pow`



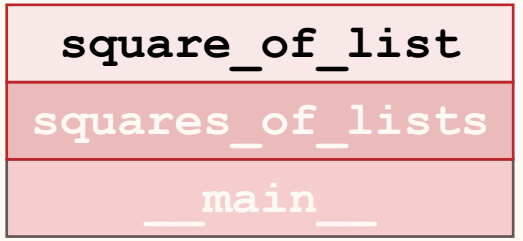
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

Ολοκληρώνεται η πρώτη pow



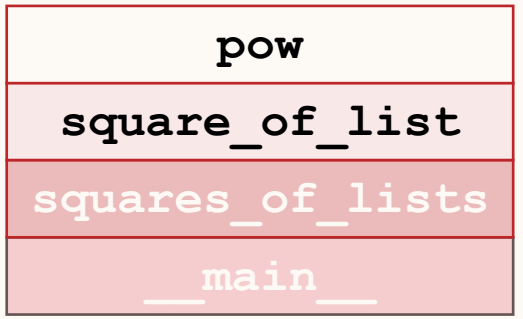
Στοίβα κλήσεων

```
def square_of_list(list_in):
    list_out=[pow(n,2) for n in list_in]
    return(list_out)

def squares_of_lists(list_of_lists):
    list_out = [square_of_list(l) for l
                        in list_of_lists]
    return(list_out)

print(squares_of_lists([[2,4,5,3],
                        [2,-2],[11,12,13]]))
```

Η εκτέλεση της `square_of_list` διακόπτεται και ξεκινάει η δεύτερη `pow`



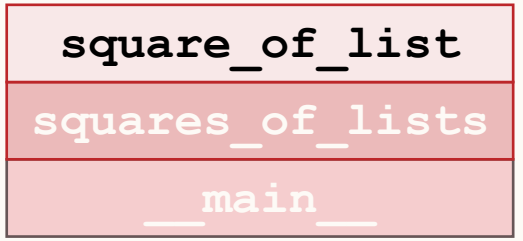
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Ολοκληρώνεται η δεύτερη pow



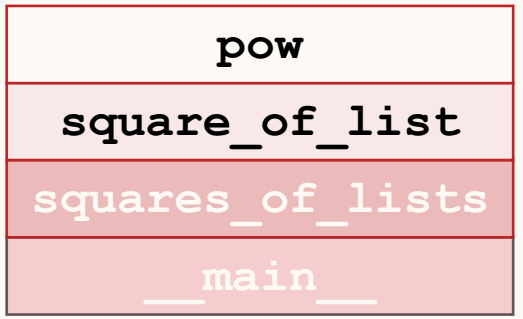
Στοίβα κλήσεων

```
def square_of_list(list_in):
    list_out=[pow(n,2) for n in list_in]
    return(list_out)

def squares_of_lists(list_of_lists):
    list_out = [square_of_list(l) for l
                        in list_of_lists]
    return(list_out)

print(squares_of_lists([[2,4,5,3],
                        [2,-2],[11,12,13]]))
```

Η εκτέλεση της `square_of_list` **διακόπτεται και ξεκινάει η τρίτη** `pow`



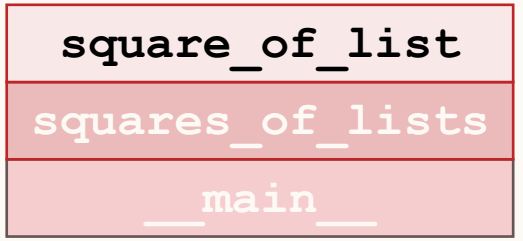
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Ολοκληρώνεται η τρίτη pow



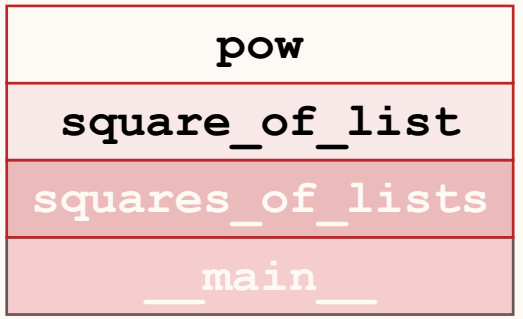
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Η εκτέλεση της `square_of_list` **διακόπτεται και ξεκινάει η τέταρτη** `pow`



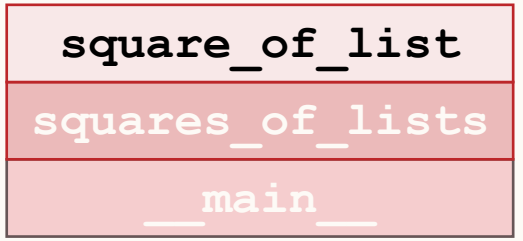
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

Ολοκληρώνεται η τέταρτη pow



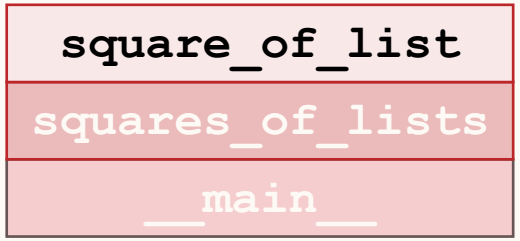
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

Επιστρέφεται η `list_out` **και κλείνει η** `square_of_list`



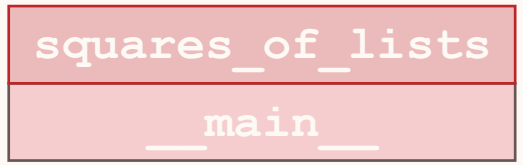
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Η τιμή μπαίνει στη λίστα `list_out`



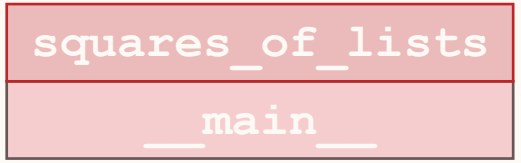
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Επόμενη επανάληψη της `for`



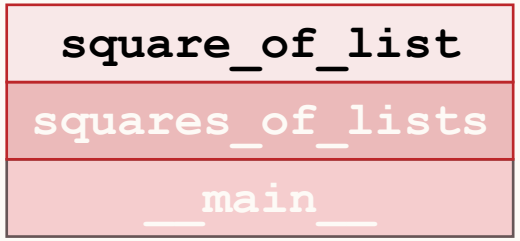
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

Καλείται ξανά η square_of_list
Μια νέα εκτέλεση προστίθεται στο stack



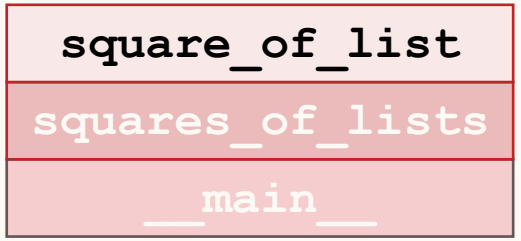
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                    in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                    [2,-2],[11,12,13]]))
```

Όταν και αυτή η `square_of_list` ολοκληρωθεί, θα επιστρέψουμε στη `squares_of_lists`



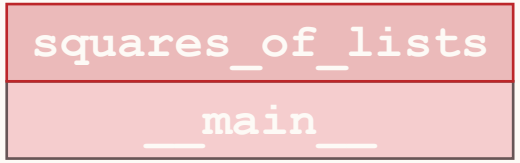
Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

Όταν ολοκληρωθούν όλες οι επαναλήψεις, θα επιστρέψουμε στη `main`...



Στοίβα κλήσεων

```
def square_of_list(list_in):
  list_out=[pow(n,2) for n in list_in]
  return(list_out)

def squares_of_lists(list_of_lists):
  list_out = [square_of_list(l) for l
                  in list_of_lists]
  return(list_out)

print(squares_of_lists([[2,4,5,3],
                  [2,-2],[11,12,13]]))
```

...και θα εμφανίσουμε το αποτέλεσμα που επέστρεψε η `squares_of_lists`

`__main__`

Έξοδοι

- Οι έξοδοι μιας συνάρτησης είναι ό,τι τύπου και ό,τι πλήθους επιθυμούμε

```
def geometric_series(base,factor,n):
  series = []
  sum_series = 0
  term = base
  for i in range(n):
    series.append(term)
    sum_series += term
    term *= factor
  return series, sum_series

geom_series, geom_sum = geometric_series(3,2,6)
print(geom_series, geom_sum)
[3, 6, 12, 24, 48, 96] 189
```

Αν έγραφα
g = geometric_series(3,2,6)
τι τύπου αντικείμενο θα ήταν το **g**;

Παράμετροι / ορίσματα

- Οι είσοδοι μιας συνάρτησης λέγονται και **παράμετροι (parameters)** ή **ορίσματα (arguments)**
  - Το πλήθος τους μπορεί να είναι 0
- Στην πιο απλή περίπτωση, έχουμε παραμέτρους θέσης (**positional arguments**)
  - Κατά την κλήση, η πρώτη τιμή πάει στην πρώτη παράμετρο, η δεύτερη στη δεύτερη κλπ

```
def polynomial2(x, a, b, c):
  return(a*x**2 + b*x + c)

print(polynomial2(5,2,-6,1))
21
```

Προσοχή στα namespaces!

- Η συνωνυμία στις παραμέτρους μεταξύ κλήσης και εκτέλεσης **δε μας αφορά** (αφού μιλάμε για *διαφορετικά namespaces*).
  - Μας αφορά μόνο η θέση των παραμέτρων.

```
def polynomial2(x, a, b, c):
  return(a*x**2 + b*x + c)
a = 5
b = 2
c = -6
d = 1
print(polynomial2(a,b,c,d))
21
```

global		local
a	⇒	x
b	⇒	a
c	⇒	b
d	⇒	c

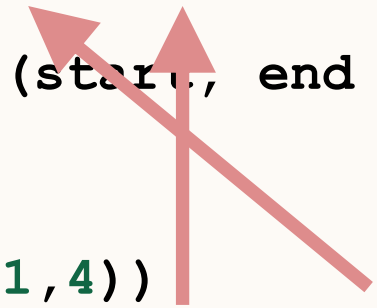
Η **a** του *global namespace* ανατίθεται στη **x** του *local namespace*, η **b** του *global* στην **a** του *local*, κλπ

Ονοματισμένες παράμετροι

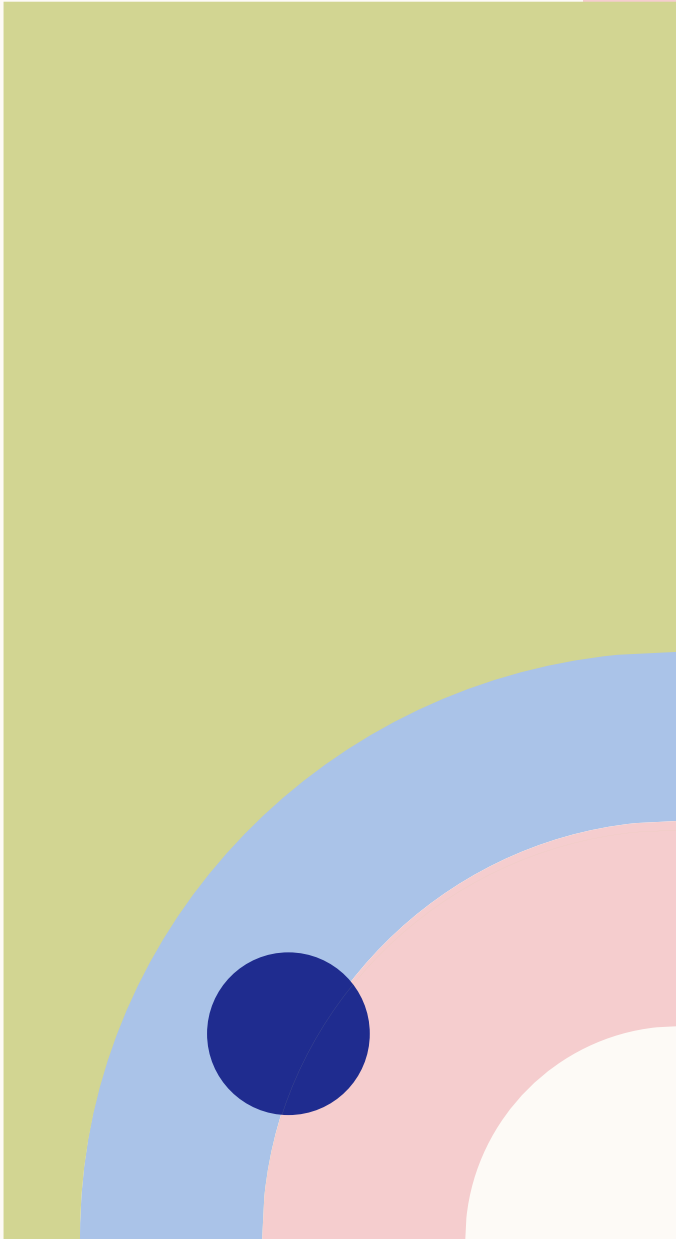
Μια άλλη επιλογή είναι να αναθέσουμε τιμές στις παραμέτρους με το όνομά τους
Στην περίπτωση αυτή, η θέση τους **αγνοείται**.

```
def factorial(start,end):
  prod = 1
  for i in range(start, end + 1):
    prod *= i
  return prod

print(factorial(1,4))
print(factorial(end = 4, start = 1))
```



Ανάποδα!



Namespaces και πάλι

- Με τις ονοματισμένες παραμέτρους μπορεί να γίνει ακόμα πιο καθαρό το ότι η συνωνυμία σε διαφορετικά namespaces δε μας ενδιαφέρει

```
def polynomial2(x, a, b, c):
  return(a*x**2 + b*x + c)
a = 5
b = 2
c = -6
d = 1
print(polynomial2(x=a,a=b,b=c,c=d))
21
```

global		local
a	⇒	x
b	⇒	a
c	⇒	b
d	⇒	c

Η **a** του *global namespace* ανατίθεται στη **x** του *local namespace*, η **b** του *global* στην **a** του *local*, κλπ

Προσοχή!

Αν κατά την κλήση της συνάρτησης χρησιμοποιώ και παραμέτρους θέσης και ονοματισμένες, οι παράμετροι θέσης πρέπει να προηγούνται:

```
def polynomial2(x, a, b, c):
  return(a*x**2 + b*x + c)
a = 5
b = 2
c = -6
d = 1
print(polynomial2(a, b, c, c=d))
```

Πρώτα οι θεσιακές, έπειτα οι ονοματισμένες.

Αλλιώς χτυπάει **SyntaxError**!

Προεπιλεγμένες τιμές παραμέτρων

- Η **`list.sort()`** μπορεί να κληθεί με μια παράμετρο **`reverse=True`**.
  - Μπορεί όμως και να κληθεί χωρίς αυτήν
- Στη δεύτερη περίπτωση, η **`reverse`** θα έχει τιμή **`False`** και η λίστα θα ταξινομηθεί με αύξουσα σειρά (*όχι φθίνουσα*)
- Λέμε ότι η παράμετρος **`reverse`** της συνάρτησης **`list.sort()`** έχει προεπιλεγμένη (*default*) τιμή **`False`**.
- Μπορούμε κι εμείς να φτιάξουμε συναρτήσεις με προεπιλεγμένες τιμές στις παραμέτρους

Προεπιλεγμένες τιμές παραμέτρων

```
def factorial(end, start=1):
    prod = 1
    for i in range(start, end + 1):
        prod *= i
    return prod

print(factorial(4))
```

```
24
```

Οι παράμετροι με προεπιλεγμένες τιμές μπαίνουν στο τέλος

το 4 πάει στην πρώτη θέση (end), στη start δεν πάει τίποτα, οπότε η start παίρνει την προεπιλεγμένη τιμή 1

Συναρτήσεις με ανοιχτό πλήθος ορισμάτων (packing)

- Κάποιες συναρτήσεις, όπως η **`print()`** και η **`sum()`** μπορούν να δεχθούν όσες παραμέτρους θέλουμε
- Μπορούμε κι εμείς να γράψουμε τέτοιες συναρτήσεις, με το χαρακτήρα **`*`**
  - Οι παράμετροι αποθηκεύονται σε ένα tuple με όσο **`len()`** χρειάζεται

```
def prod(*args):
  prod = 1
  for a in args:
    prod *= a
  return prod

print(prod(2,4,6,7,2))
```

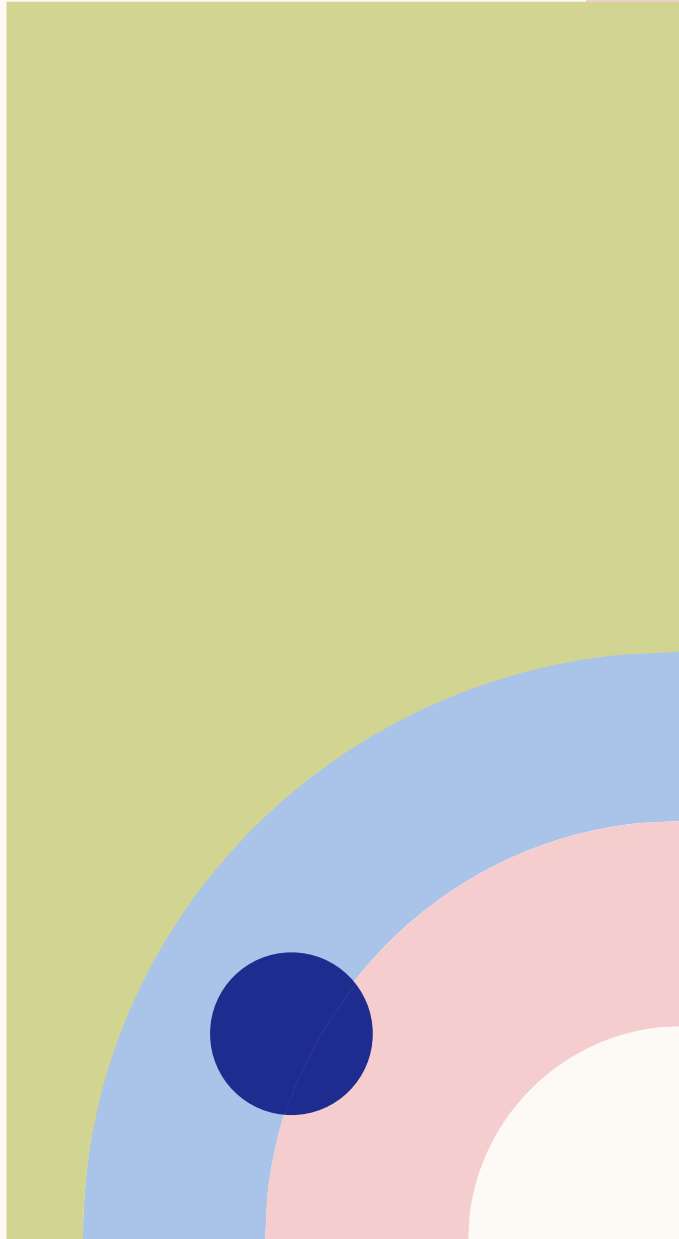
```
672
```

Packing σε λεξικό

- Με παρόμοιο τρόπο, με το σύμβολο ** μια σειρά ονοματισμένων παραμέτρων περνάνε στη συνάρτηση ως κλειδιά-τιμές ενός λεξικού

```
def flex_inputs(**kwargs):
    for key in kwargs.keys():
        print(key)
    for value in kwargs.values():
        print(value)

flex_inputs(input1=2,value2=4,another_value=6)
```



Αναδρομή

- Είπαμε ότι μια συνάρτηση μπορεί να καλεί οποιαδήποτε άλλη συνάρτηση μέσα από τον κώδικά της
- Άρα μπορεί να καλεί και τον εαυτό της!
- Πρέπει:
  - Να ορίσουμε το πρόβλημα σε βήματα, κάθε ένα από τα οποία να καλεί την ίδια μέθοδο
  - Σε κάθε βήμα θα πρέπει το πλήθος των υπολογισμών που απομένουν να μειώνεται
  - Θα πρέπει να υπάρχει ένας ειδικός έλεγχος ώστε, όταν μειωθεί αρκετά το πλήθος των υπολογισμών, η συνάρτηση να μην καλεί τον εαυτό της αλλά να επιστρέφει μια τιμή και να τερματίζει.

Παράδειγμα 7.4

Να γραφεί αναδρομική συνάρτηση που θα υπολογίζει το πλήθος των ψηφίων ενός θετικού ακέραιου αριθμού

Ανάλυση: μπορώ να γράψω τον αλγόριθμο ως εξής:

πλήθος_ψηφίων(αριθμός) = { 1, αν αριθμός<10 ; 1+**πλήθος_ψηφίων**(αριθμός//10) αλλιώς

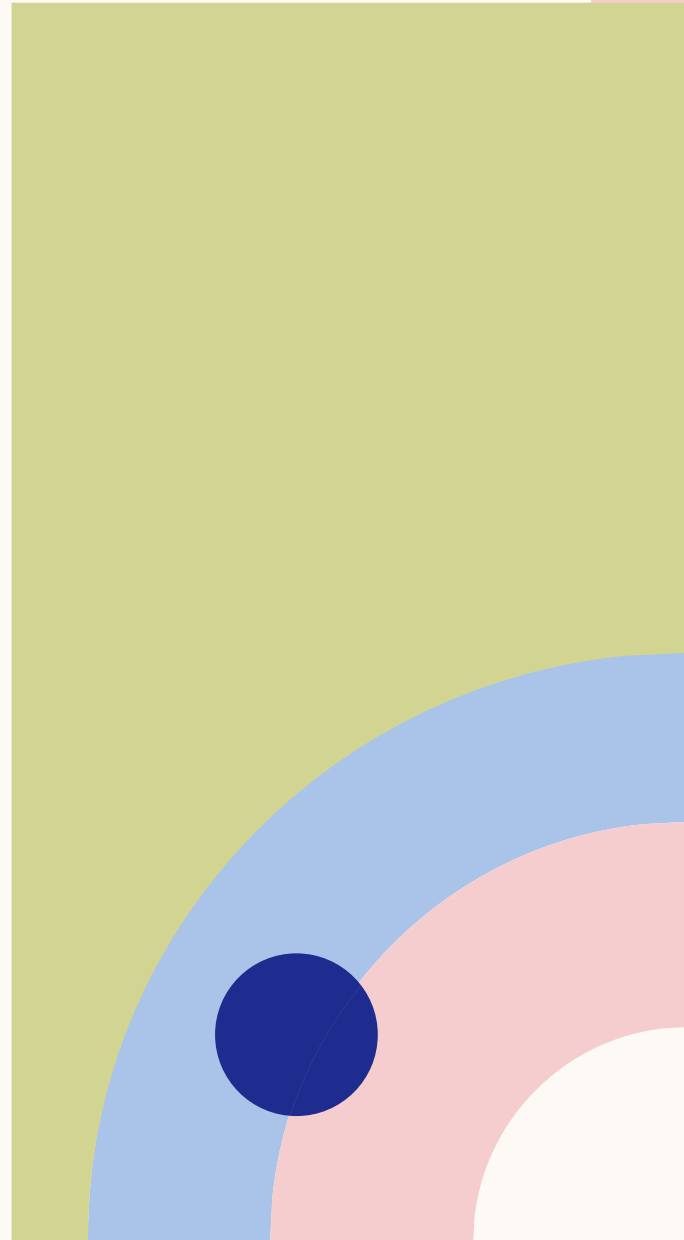
Αναδρομικό πλήθος ψηφίων

πλήθος_ψηφίων(αριθμός) = { 1, αν αριθμός<10 ; 1+**πλήθος_ψηφίων**(αριθμός//10) αλλιώς

```
def count_digits(n):
  if n<10:
    return 1
  else:
    return 1+count_digits(n//10)
```

Παράδειγμα 7.5

Να γραφεί αναδρομική συνάρτηση σε Python που να υπολογίζει το **n!** (**n** παραγοντικό)



Παράδειγμα 7.5

Να γραφεί αναδρομική συνάρτηση σε Python που να υπολογίζει το **n!** (**n** παραγοντικό)

Αναδρομικά, το **n!** μπορεί να γραφεί:

$$n! = \begin{cases} 1, & \text{αν } n=1 \\ n*(n-1)! & \text{αλλιώς} \end{cases}$$

Παράδειγμα 7.5

$$n! = \begin{cases} 1, & \text{αν } n=1 \\ n*(n-1)! & \text{αλλιώς} \end{cases}$$

```
def fact(n):
  if n==1:
    return 1
  else:
    return n*fact(n-1)
```

Παράδειγμα 7.6

Να γραφεί συνάρτηση σε Python που θα δέχεται δυο λίστες και θα επιτελεί πρόσθεση ανά στοιχείο. Αν οι λίστες έχουν διαφορετικό μήκος, να εμφανίζει ένα μήνυμα λάθους και να επιστρέφει `None`

Παράδειγμα 7.6

Να γραφεί συνάρτηση σε Python που θα δέχεται δυο λίστες και θα επιτελεί πρόσθεση ανά στοιχείο. Αν οι λίστες έχουν διαφορετικό μήκος, να εμφανίζει ένα μήνυμα λάθους και να επιστρέφει **None**

```
def add_list(list1, list2):
  if len(list1)!=len(list2):
    print("ERROR")
    return None
  new_list = []
  for i in range(len(list1)):
    new_list.append(list1[i]+list2[i])
  return new_list
```

Παράδειγμα 7.6

Να γραφεί συνάρτηση σε Python που θα δέχεται δυο λίστες και θα επιτελεί πρόσθεση ανά στοιχείο. Αν οι λίστες έχουν διαφορετικό μήκος, να εμφανίζει ένα μήνυμα λάθους και να επιστρέφει **None**

Εναλλακτικά:

```
def add_list2(list1, list2):
  if len(list1)!=len(list2):
    print("ERROR")
    return None
  return [sum(x) for x in zip(a,b)]
```

Ας ξαναεπισκεφθούμε την `print()`

https://www.w3schools.com/python/ref_func_print.asp

```
print(object(s), sep=separator, end=end, file=file,
                                              flush=flush)
```

Τι επίδραση έχουν οι αλλαγές στη **`separator`** και στην **`end`**;