Περισσότερα σχετικά με συναρτήσεις

Συναρτήσεις χρήστη

Μπορείτε να βάλετε τους ορισμούς διαφόρων συναρτήσεων σε ένα file, p.x. myfuncs.py
Όταν θέλετε να χρησιμοποιήσετε κάποια συνάρτηση μπορείτε να δώσετε την εντολή:
from myfuncs import afunction όπου afunction είναι το όνομα μιας συνάρτησης

powers.py — file που περιέχει τον ορισμό της συνάρτησης

```
def powerN(num,N):
     result = num**N
     return result
```

ορισμός της συνάρτησης

mytest.py — file ενός προγράμματός μας

```
from powers import powerN
number = int(input("Give the number "))
ekthetis = int(import("Give the exponent"))
result = powerN(number,ekthetis)
print("The result of %d**%d is %d"%(number,ekthetis,result))
```

εισαγωγή της συνάρτησης για κλήση της

Τεκμηρίωση της συνάρτησης χρήση

Η τεκμηρίωση της συνάρτησης αποτελεί ένα σημαντικό στάδιο της θεμελίωσης της συνάρτησης

Οι υπόλοιποι χρήστες αλλά και εμείς μπορούμε να ανατρέξουμε για να βρούμε τον τρόπο με τον οποίο χρησιμοποιείται (ουσιαστικά είναι το help της συνάρτησης)

powers.py

```
def powerN(num,N):
    '''
    This function takes as
    input an integer number num
    and an exponent N and
    returns an integer which
    is the num**N '''
    result = num**N
    return result
```

άνοιγμα σχολίου περισσότερο από 1 γραμμή προσοχή στη στοίχιση που ακολουθείται και στα σχόλια

Τεκμηρίωση

κλείσιμο σχολίου

Σε περιβάλλον Python θα μπορούσαμε να γράψουμε:

```
>>> from powers.py import powerN
>>> help(powerN)
```

```
Help on function powerN in module powers:

powerN(num, N)
    This function takes as
    input an integer num
    and an exponent N and
    returns an integer
    which is the num**N
(END)
```

Η συνάρτηση print ()

Η συνάρτηση **print()** χρησιμοποιείται για να τυπώσουμε κάποιο αντικείμενο είτε στην οθόνη ή σε κάποιο εξωτερική διάταξη.

Η πλήρης σύνταξη της συνάρτησης είναι:

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

- **object(s) –** τα αντικείμενα που θα εκτυπωθούν
- **sep –** ο τρόπος διαχωρισμού των αντικειμένων – by default είναι ” ”
- **end –** προσδιορίζει ποιο αντικείμενο θα πρέπει τουλάχιστον να εκτυπωθεί
- **filename –** όνομα αρχείου για εκτύπωση που είναι διαθέσιμο για εγγραφή
- **flush –** καθαρισμός του output stream. Αυτό είναι False by default.

Συνήθως στο τέλος κάθε εκτέλεσης της εντολής print υπάρχει end=‘\n’, αλλαγή γραμμής.

Συναντήσαμε αρκετές περιπτώσεις στις οποίες θα θέλαμε να διατηρηθεί η ίδια γραμμή μετά την εκτύπωση ενός αντικειμένου.

Για να διατηρήσουμε εκτύπωση στην ίδια γραμμή, χρησιμοποιούμε το print με την επιλογή end=“ “

```
print(‘b=’, a, end=‘ ‘)
```

Η συνάρτηση print()

Έστω ότι θέλουμε να τυπώσουμε μια λίστα με 6 αριθμούς σε μορφή 2 γραμμών με 3 στήλες

Α' Τρόπος

```
A = [ 1, 2, 3, 4, 5, 6]
for ii in range(0,6,3):
    print(A[ii+0],A[ii+1],A[ii+2])
```

Β' Τρόπος

```
A = [ 1, 2, 3, 4, 5, 6]
for ii in range(0,6,3):
    for jj in range(3):
       print(A[ii+jj],end=' ')  # Diatiroume tin idia grammi sto jj loop
    print()                     # Allagi grammis meta to telos tou jj loop
```

Περισσότερα σχετικά με συναρτήσεις: local/global μεταβλητές

Συναρτήσεις στην Python – dive in

Έχουμε ορίσει και χρησιμοποιήσει δικές μας συναρτήσεις στην Python

Η γενική σύνταξη όπως έχουμε δει είναι:

```
def  FunctionName( parameters ) :
    block entolwn
    return
```

Η ερώτηση είναι «τι ακριβώς περνούμε στον κώδικα της συνάρτησης μέσω των παραμέτρων?»

Όλες οι παράμετροι (ορίσματα) που χρησιμοποιούνται στην Python περνούν με αναφορά (**by reference**) στη διεύθυνση της μνήμης του υπολογιστή.

Αυτό σημαίνει ότι όταν αλλάξουμε την τιμή που είναι αποθηκευμένη στην διεύθυνση της μνήμης που αναφέρεται η παράμετρος τότε η αλλαγή αυτή θα εμφανιστεί και στο τμήμα του κώδικα που καλεί την συνάρτηση.

```
#!/usr/bin/python3
def changeme( mylist ):
   mylist.append([1,2,3,4])
   print("Values inside the function: ", mylist)
   return
mylist = [10,20,30]
changeme( mylist )
print("Values outside the function: ", mylist)
```

output

→ [10,20,30,1,2,3,4]

→ [10,20,30,1,2,3,4]

Συναρτήσεις στην Python – dive in

Η αναφορά στη διεύθυνση της μνήμης μπορεί να αλλάξει μέσα στο κύριο σώμα της συνάρτησης και να απενεξαρτοποιηθεί από τη διεύθυνση που καθόρισε το τμήμα του κώδικα που κάλεσε τη συνάρτηση:

```
def changeme( mylist ):
   mylist = [1,2,3,4]
   print("Values inside the function: ", mylist)
   return
mylist = [10,20,30]
changeme( mylist )
print("Values outside the function: ", mylist)
```

Ορισμός αντικειμένου mylist δημιουργεί νέα διεύθυνση

output

[1,2,3,4]

[10,20,30]

Στο παραπάνω παράδειγμα, η mylist είναι **τοπική παράμετρος** (local variable list) για τη συνάρτηση και οι τιμές που εισαγάγουμε αλλάζουν την τοπική list αλλά δεν αλλάζουν τη λίστα που εισάγεται από το τμήμα που καλεί τη συνάρτηση.

```
def swap(a,b):
   temp = b
   b = a
   a = temp
   return
x = 2
y = 3
swap(x,y)
print("x, y", x,y)
```

output

2, 3

Τοπικές (local) και Γενικευμένες (global) μεταβλητές

Οι μεταβλητές σε ένα πρόγραμμα δεν είναι προσβάσιμες σε όλα τα σημεία του προγράμματος.

Αυτό εξαρτάται από το που έχουν οριστεί οι μεταβλητές

Ανάλογα με την έκταση εφαρμογής (**scope**) μιας μεταβλητής, στην Python ξεχωρίζουμε δύο είδη:

- **Τοπικές** (**local**) μεταβλητές
- **Γενικευμένες** (**global**) μεταβλητές

Μεταβλητές που ορίζονται στο σώμα μιας συνάρτησης έχουν τοπικό χαρακτήρα, ενώ αυτές που ορίζονται εκτός συναρτήσεων είναι γενικευμένες μεταβλητές.

Όσες μεταβλητές ορίζονται μέσα σε μια συνάρτηση είναι προσβάσιμες μόνο από εντολές της συνάρτησης ενώ μεταβλητές που ορίζονται εκτός της συνάρτησης είναι προσβάσιμες από όλες τις συναρτήσεις του προγράμματος.

Με την κλίση μιας συνάρτησης, οι μεταβλητές που ορίζονται μέσα στην συνάρτηση έρχονται σε ισχύ στο πεδίο εφαρμογής της συνάρτησης

Local και global μεταβλητές

Το παρακάτω παράδειγμα δείχνει τις περιπτώσεις των local και global μεταβλητών.

```
total = 0  # This is global variable.
glb   = 1  # This is global variable.

def sum( arg1, arg2 ):
   # Add both the parameters and return them
   total = arg1 + arg2 #  total is local variable.
   test  = 2*glb       #  test is local variable but glb global
   print("Inside the function local total : ", total, test)
   return total, test

a, b = sum( 10, 20 )
print("Outside the function global total : ", total, glb, a, b)
```

Το αποτέλεσμα του παραπάνω παραδείγματος θα είναι:
30, 2
0, 1, 30, 2

Namespace και scope

Έχουμε αναφέρει νωρίτερα ότι όταν κάνουμε import ένα module αυτόματα ορίζεται ένας χώρος με τα ονόματα όλων των συναρτήσεων/μεθόδων που σχετίζονται με το module

Ο χώρος αυτός ονομάζεται **namespace** του module και αποτελεί στην πραγματικότητα ένα dictionary με τα ονόματα των συναρτήσεων ως keys του dictionary και τα αντίστοιχα αντικείμενα αποτελούν τις τιμές των keys.

Μια εντολή python μπορεί να έχει πρόσβαση στο global namespace ή σε ένα local namespace.

Αν μια local και μια global μεταβλητή έχουν το ίδιο όνομα, τότε η local μεταβλητή επισκιάζει την global

Κάθε συνάρτηση σχετίζεται με το δικό της τοπικό namespace μεταβλητών

Για να αλλάξουμε την τιμή μιας global μεταβλητής μέσα σε μία συνάρτηση θα πρέπει να ορίσουμε την μεταβλητή ως global μέσα στη συνάρτηση, δίνοντας την εντολή:
global VarName

Η εντολή global varname

Η εντολή **global** VarName δηλώνει στην Python ότι VarName είναι global και παύει η Python να προσπαθεί να την βρει στο local namespace

```
Money = 2000
def AddMoney():
  Money = Money + 1
  print(Money)

AddMoney()
print(Money)
```

UnboundLocalError: local variable 'money' referenced before assignment

fix

```
Money = 2000
def AddMoney():
  global Money
  Money = Money + 1
  print(Money)

AddMoney()
print(Money)
```

Οι συναρτήσεις **globals()** και **locals()** επιστρέφουν τα keys του dictionary του global namespace και ενός local namespace dictionary.

Αν οι συναρτήσεις **globals()** και **locals()** κλιθούν μέσα από μια συνάρτηση, επιστρέφουν τα ονόματα των μεταβλητών που είναι global για τη συνάρτηση ή είναι local για τη συνάρτηση.

Οι συναρτήσεις αυτές επιστρέφουν dictionaries ως αντικείμενα με keys τα ονόματα όλων των global και local μεταβλητών και μπορούμε να τα εξαγάγουμε χρησιμοποιώντας τη συνάρτηση keys του dictionary.

Χρήσιμες δομές

List Comprehension

Έστω ότι θέλουμε να εξαγάγουμε κάθε χαρακτήρα από ένα αντικείμενο τύπου string και να τον εισαγάγουμε σε μια λίστα

Ένας τρόπος είναι να χρησιμοποιήσουμε ένα *for loop*

```
LetList = []
word = 'banana'
for letter in word:
   LetList.append(letter)
print(LetList)
```

Output:

['b','a','n','a','n','a']

Η Python προσφέρει έναν πιο γρήγορο τρόπο για να κάνουμε την ίδια διαδικασία.

Η μέθοδος ονομάζεται **list comprehension** και παρέχει τον ορισμό και δημιουργία μιας λίστας στηριζόμενη σε προϋπάρχουσα λίστα

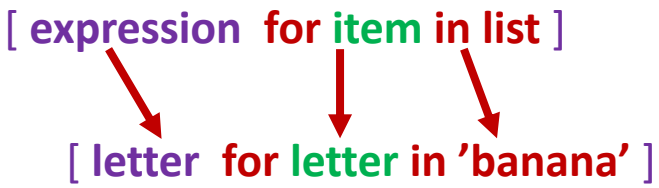
```
word = 'banana'
LetList = [ letter for letter in word]
print(LetList)
```

Στο παραπάνω παράδειγμα, δημιουργείται μια νέα λίστα με όνομα *LetList* και περιέχει τα στοιχεία της *word* που είναι η string μεταβλητή για τη λίστα/ακολουθία 'banana'

List Comprehension

Η σύνταξη της δομής list comprehension είναι η ακόλουθη:

[expression for item in list]



Στο προηγούμενο παράδειγμα είχαμε :

Παρόλο η σταθερά 'banana' είναι τύπου string και όχι list, η δομή ερμηνεύει την ακολουθία ως list. Το ίδιο συμβαίνει και με τα tuples.

Η δομή της list comprehension μπορεί να συνδυαστεί με μια άλλη δομή list comprehension, ή μια συνθήκη.

List Comprehension – με συνθήκη

Χρησιμοποιώντας μια συνθήκη στη θέση της expression στη list comprehension, μπορούμε να μεταβάλλουμε μια υπάρχουσα list ή tuple

```
AList = [ x for x in range(20) if x%4 == 0]
print(AList)
```

Output:

[4, 8,12,16]

Η λίστα AList γεμίζει με τα στοιχεία της λιστας [0,19] τα οποία διαιρούνται με το 4.

List Comprehension – με φωλιασμένη συνθήκη

Μπορεί στη θέση της συνθήκης να υπάρχει φωλιασμένη συνθήκη κάνοντας την συνθήκη αρκετά πιο πολύπλοκη για την επιλογή από την προϋπάρχουσα list

```
AList = [x for x in range(100) if x%2 == 0 if x%5 == 0]
print(AList)
```

Output:

[0,
10,
20,
30,
40,
50,
60,
70,
80,
90]

Η list comprehension στην περίπτωση αυτή επιλέγει ως στοιχεία της list AList αυτά που ικανοποιούν τις συνθήκες:

- x διαιρείται με το 2
- x διαιρείται με το 5

❑ Αν ικανοποιούνται και οι 2 συνθήκες, το στοιχείο x φυλάσσεται στην list AList

List Comprehension – με if … else… συνθήκη

Θα μπορούσε η συνθήκη υπόθεσης να είναι περισσότερο πολύπλοκη:

```
AList = [‘Even’ if x%2 == 0 else ‘Odd’ for x in range(10)]
print(AList)
```

Output:

[‘Even’,
‘Odd’,
’Even’,
‘Odd’,
‘Even’,
‘Odd’,
’Even’,
‘Odd’,
‘Even’,
‘Odd’]

Η list comprehension στην περίπτωση αυτή επιλέγει ως στοιχεία της list AList :

- ‘Even’ αν το x διαιρείται με το 2
- ‘Odd’ αν το x δεν διαιρείται με το 2

List Comprehension – με φωλιασμένο loop

Σε ασκήσεις του προηγούμενου εργαστηρίου υπάρχει στις προτεινόμενες λύσεις η χρήση ενός φωλιασμένου loop

```
AList = [[random.randint(0,100) for y in range(3)] for x in range(4)]
print(AList)
```

Στην περίπτωση αυτή δημιουργείται μια list με 3 στήλες και 4 γραμμές

Ο τρόπος που δουλεύει το φωλιασμένο loop στην περίπτωση αυτή, είναι να πάρει το x μια τιμή από το εύρος [0,4) (αυτή θα είναι η γραμμή της AList) και κατόπιν να επιλέξει τιμές του y στο διάστημα [0,3) (οι στήλες της γραμμής).

Η τιμή κάθε στοιχείου επιλέγεται τυχαία στο κλειστό διάστημα [0,100] με τη randint

Μέθοδος map()

Μια συνάρτηση μπορεί να εφαρμοστεί σε όλα τα στοιχεία ενός array ή μιας list χρησιμοποιώντας την μέθοδο **map**

Για παράδειγμα θα μπορούσαμε να έχουμε μια συνάρτηση που πολλαπλασιάζει τα στοιχεία μιας λίστας με 2 και αφαιρεί 1

```
def f(x):
    return 2*x - 1

newlist =list(map(f,oldlist))
```

Εφαρμόζει τη συνάρτηση f σε κάθε στοιχείο της λίστας oldlist και δημιουργεί μια νέα λιστα με τα αποτελέσματα με όνομα newlist