

## Περισσότερα σχετικά με συναρτήσεις

# Συναρτήσεις στην Python – dive in

Έχουμε ορίσει και χρησιμοποιήσουμε δικές μας συναρτήσεις στην Python

Η γενική σύνταξη όπως έχουμε δει είναι:

```
def FunctionName( parameters ) :  
    block of code  
    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 από 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()** κληθούν 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.

## Γραμματοσειρές - Strings



## Δεδομένα τύπου String

Ένα αντικείμενο τύπου string είναι μια σειρά από χαρακτήρες

Για να δημιουργήσουμε ένα αντικείμενο τύπου string θα πρέπει να το βάλουμε τη σειρά των χαρακτήρων σε " "

Ο τελεστής **+** δηλώνει ένωση δύο αντικειμένων τύπου string (**concatenation**).

Όταν το αντικείμενο τύπου string περιέχει νούμερα μέσα σε " " εξακολουθεί να είναι string .

Μπορούμε να μετατρέψουμε νούμερα που περιέχονται σε strings σε νούμερα, χρησιμοποιώντας κατάλληλες συναρτήσεις [ int(), float() ].

```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
>>>
```

## Ανάγνωση και μετατροπή από string σε νούμερα

Προτιμούμε να διαβάζουμε strings και κατόπιν να μετατρέπουμε στην κατάλληλη αριθμητική μορφή (int, float)

Η μεθοδολογία αυτή μας δίνει περισσότερο έλεγχο σε λάθος εισαγωγή δεδομένων

Η εισαγωγή αριθμητικών δεδομένων γίνεται επομένως μέσω strings

```
>>> name = input('Enter:')
Enter:Fotis
>>> print(name)
Fotis
>>> apple = input('Enter:')
Enter:100
>>> x = apple - 10
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
TypeError: unsupported
operand type(s) for -: 'str'
and 'int'
>>> x = int(apple) - 10
>>> print(x)
90
```

## Έλεγχος χαρακτήρων μέσα στο string

Ένα αντικείμενο τύπου string είναι ένα tuple (immutable object) με μήκος ίσο με τον αριθμό των χαρακτήρων που περιέχει

Μπορούμε να έχουμε πρόσβαση στον χαρακτήρα που βρίσκεται σε κάποια συγκεκριμένη θέση ακριβώς με τον ίδιο τρόπο όπου έχουμε πρόσβαση στο στοιχείο ενός tuple ή μιας λίστας.

Θα πρέπει να δώσουμε το όνομα του αντικειμένου string ακολουθούμενο από ένα νούμερο που περικλείεται σε [] (i.e. fruit[2] που αντιστοιχεί στον χαρακτήρα n

Το νούμερο μέσα στην τετραγωνική αγκύλη θα πρέπει να είναι ένας ακέραιος και η αρχική του τιμή θα είναι 0 και η μεγαλύτερη το μήκος της γραμματοσειράς - 1.

Μπορεί να είναι μια αριθμητική έκφραση που δίνει αποτέλεσμα ακέραιο

Η συνάρτηση len() επιστρέφει το μήκος της γραμματοσειράς

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

```
>>> fruit = 'banana'
>>> print(len(fruit))
6
```

## Επαναληπτική διαδικασία με strings

Με τη χρήση της δομής while, μιας μεταβλητής επανάληψης (index) και της συνάρτησης len() μπορούμε να εξετάσουμε όλους τους χαρακτήρες ενός αντικειμένου τύπου string

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

0 b  
1 a  
2 n  
3 a  
4 n  
5 a

Η χρήση ενός for loop είναι πολύ καλύτερη

- Η μεταβλητή επανάληψης περιέχεται μέσα στην δομή του for loop

```
fruit = 'banana'
for letter in fruit:
    print(letter)
```

b  
a  
n  
a  
n  
a

Θα μπορούσαμε ενώ κινούμαστε από τη μια θέση στην επόμενη ενός string να εξετάζουμε πόσες φορές εμφανίζεται κάποιος χαρακτήρας

```
word = 'banana'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print(count)
```

# Εξέταση της επαναληπτικής διαδικασίας for loop

Η μεταβλητή επανάληψης (**letter**) εκτελεί μια επαναληπτική διαδικασία σύμφωνα με την ορισμένη και ταξινομημένη ακολουθία (**banana**)

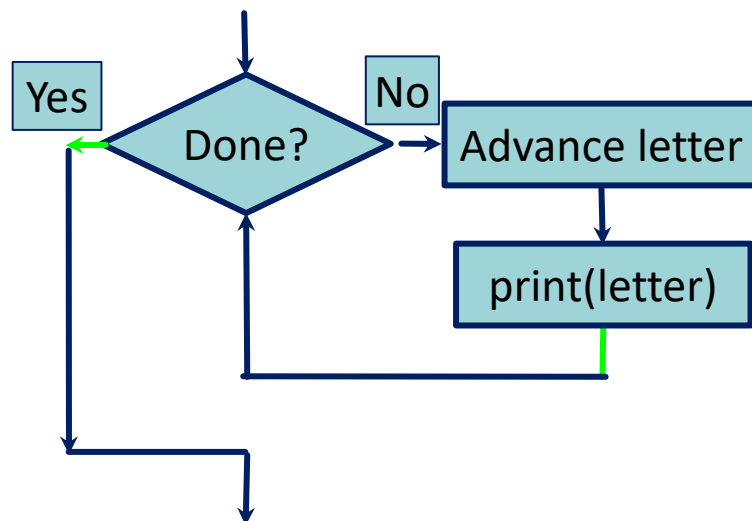
Το κύριο σώμα του loop (οι εντολές του loop) εκτελούνται μια φορά για κάθε τιμή της ακολουθίας

Η μεταβλητή επανάληψης παίρνει όλες τις τιμές που ορίζονται μέσα (**in**) στην ακολουθία

Μεταβλητή  
επανάληψης

string 6-χαρακτήρων

```
for letter in 'banana':  
    print(letter)
```



## Διαχείριση αντικειμένων τύπου strings

## Χρήση τμήματος string - Slicing

Μπορούμε να εξετάσουμε ένα οποιοδήποτε συνεχές διάστημα ενός αντικειμένου τύπου string χρησιμοποιώντας τον τελεστή :

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

Δίνουμε το όνομα του αντικειμένου ακολουθούμενο από [ n1 : n2 ]

Το 1<sup>ο</sup> νούμερο, n1, δίνει τη θέση από την οποία αρχίζουμε και το 2<sup>ο</sup> νούμερο, n2, δηλώνει τη θέση στην οποία σταματούμε (χωρίς να προσμετράται η θέση αυτή). Είναι επομένως διάστημα τιμών θέσεων κλειστό στο κάτω όριο και ανοικτό στο πάνω

Αν το 2<sup>ο</sup> νούμερο είναι μεγαλύτερο από το συνολικό μήκος του string σταματά στο τέλος του string

```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```

## Slicing strings

Αν παραλείψουμε την τιμή του είτε του κάτω ορίου ή την τιμή Του πάνω ορίου τότε θεωρείται αυτόματα η τιμή της αρχικής θέσης ή της τελικής θέσης του string αντίστοιχα

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```



# Strings και η διαχείρισή τους

Δεν μπορούμε να έχουμε πρόσθεση αριθμού σε αντικείμενο τύπου string.

`S= "This is a string" + 2` δεν επιτρέπεται

Ο τελεστής `+` δηλώνει ένωση (**concatenation**) δύο αντικειμένων τύπου string

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print(b)
HelloThere
>>> c = a + ' ' + 'There'
>>> print(c)
Hello There
>>>
```

Χρήση των συναρτήσεων `int()` και `float()` μπορεί να μετατρέπει strings σε integers ή float μεταβλητές

`S= "11235"`

`S= "1.1235"`

`a= int(S)`

`b= float(S)`

To a είναι 11235

To b είναι 1.1235

```
s = '123'
pie = '3.141592653589'
x = int(s) + 1
y = float(pie) + 1
z_bad = int(s) + int(pie)
z_good = int(s) + int(float(pie))
```

Η Python δεν ξέρει πως να διαχειριστεί την μετατροπή ενός αντικείμενου string που περιέχει υποδιαστολή σε αντικείμενο τύπου int.

## Χρήση της χαρακτηριστικής λέξης **in** ως λογικός τελεστής


Η χαρακτηριστική λέξη **in** που εμφανίζεται στα **for loops** μπορεί να χρησιμοποιηθεί για να ελέγξουμε αν ένα αντικείμενο τύπου **string** περιέχεται σε κάποιο άλλο αντικείμενο τύπου **string**

Η έκφραση που περιέχει το **in**, αποτελεί μια λογική έκφραση και επομένως το αποτέλεσμα θα είναι είτε **True** ή **False** και μπορεί να χρησιμοποιηθεί σε ένα **if** δομή

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Found it!')
...
Found it!
>>>
```

## Strings και η διαχείρισή τους

Η Python μπορεί να προσθέσει δύο strings με +

```
>>>  
>>> s="2" + "3"  
>>> s  
'23'  
>>> 
```

---

Αρκετές συναρτήσεις χρειάζονται ορίσματα που είναι αντικείμενα τύπου string

Αρκετές φορές strings θέλουμε να περιέχουν κάποια νούμερα, τα οποία όμως θα πρέπει να τα μετατρέψουμε σε strings και να προσθέσουμε στο υπόλοιπο string

π.χ. S = "The average number is " + **str**(5)

Η συνάρτηση **str(N)** μετατρέπει το όρισμα N σε string

Μπορούμε να καθορίσουμε τον αριθμό των ψηφίων που θα εμφανιστούν σε μια μεταβλητή string χρησιμοποιώντας το **%** σύμβολο μορφοποίησης ή την μέθοδο **format**

## Η μέθοδος **format**

```
"The value of pi is approximately " + str(np.pi)
"The value of {} is approximately {:.5f}".format('pi', np.pi)
s = "{1:d} plus {0:d} is {2:d}"
s.format(2, 4, 2 + 4)
"Every {2} has its {3}.".format('dog', 'day', 'rose', 'thorn')
"The third element of the list is {0[2]:g}.".format(np.arange(10))
```

```
>>>
>>> "The value of pi is approximately " + str(np.pi)
'The value of pi is approximately 3.141592653589793'
>>>
>>> "The value of {} is approximately {:.5f}".format('pi', np.pi)
'The value of pi is approximately 3.14159'
>>>
>>> s="{1:d} plus {0:d} is {2:d}"
>>> s.format(2,4,2+4)
'4 plus 2 is 6'
>>>
>>> "every {2} has its {3}.".format("dog","day","rose","thorn")
'every rose has its thorn.'
>>> "The third element of the list is {0[2]:g}.".format(np.arange(10))
'The third element of the list is 2.'
>>> □
```

Όταν η Python εξετάζει το `format` ενός αντικειμένου τύπου `string`, ερμηνεύει ένα ζευγάρι `{}` ως τη θέση για να εισαγάγει κάποιους αριθμούς.

Το όρισμα της `format` είναι μια σειρά από εκφράσεις οι τιμές των οποίων θα εισαχθούν σε συγκεκριμένες θέσεις. Μπορεί να είναι `strings`, `αριθμοί`, `lists` ή κάτι πιο πολύπλοκο

## Η μέθοδος **format**

- ❑ Μια άδεια {} ερμηνεύεται ως την εισαγωγή του επόμενου στοιχείου μιας λίστας. Το στοιχείο μπορεί να έχει συγκεκριμένο τρόπο γραφής (formatted) εάν στα άγκιστρα περιέχεται ο τελεστής **:** ακολουθούμενος από μια εντολή μορφοποίησης

**Για παράδειγμα:**

```
>>> "The value of pi is approximately " + str(np.pi)
'The value of pi is approximately 3.141592653589793'
```

**{:.5f}** Σημαίνει ότι μορφοποίησε το τρέχον στοιχείο της λίστας ως έναν αριθμό float με 5 ψηφία μετά την υποδιαστολή

- ❑ Μπορούμε να αναφέρουμε στοιχεία της λίστας δίνοντας τη θέση τους στη λίστα που περιέχεται στην εντολή format.

**Για παράδειγμα:**

```
>>> s="{1:d} plus {0:d} is {2:d}"
>>> s.format(2,4,2+4)
'4 plus 2 is 6'
>>>
```

**S** ορίζεται σαν string με 3 θέσεις κρατημένες και οι οποίες θα γεμίσουν αργότερα

Θα πάρει το δεύτερο στοιχείο της λίστας **{1:d}**, το πρώτο **{0:d}** και το τρίτο **{2:d}** και θα τα εισαγάγει με αυτή τη σειρά στην string s.

Το σύμβολο **d** σημαίνει αναπαράσταση του αριθμού σε βάση δεκαδικού συστήματος ως **ακέραιος**.

Η Python μπορεί να αναπαραστήσει ακραίους σε δυαδικό (**binary**) με **{:b}** οκταδικό (**octal**) **{:o}**, δεκαεξαδικό (**hexadecimal**) **{:h}**

## Η μέθοδος **format**

- ❑ Δεν είναι απαραίτητο να χρησιμοποιήσουμε όλα τα στοιχεία της λίστας

```
>>> "every {2} has its {3}.".format("dog", "day", "rose", "thorn")  
'every rose has its thorn.'
```

- ❑ Μπορούμε να αναφερθούμε σε συγκεκριμένο στοιχείο μιας λίστας που εισάγεται στην εντολή **format**

```
>>> "The third element of the list is {0[2]:g}.".format(np.arange(10))  
'The third element of the list is 2.'  
>>> □
```

Το πεδίο αντικατάστασης **0[2]** αναφέρεται στο 3<sup>ο</sup> στοιχείο του πρώτου ορίσματος ενώ ο μορφοποιητικός κωδικός **:g** δίνει την οδηγία στην Python να απεικονίσει τον αριθμό με τον λιγότερο αριθμό ψηφίων (**g**eneral format)

- ❑ Η εντολή **dir(str)** δίνει πληροφορίες για τις διαθέσιμες μεθόδους διαχείρισης strings

## Η μέθοδος %

```
"The value of pi is approximately " + str(np.pi)
"The value of %s is approximately %.5f" % ('pi', np.pi)
s = "%d plus %d is %d"
s % (2, 4, 2 + 4)

>>> "The value of pi is approximately " + str(np.pi)
'The value of pi is approximately 3.141592653589793'
>>>
>>>
>>> "The value of %s is approximately %.5f" % ('pi', np.pi)
'The value of pi is approximately 3.14159'
```

- ❑ Αντικείμενο τύπου string το οποίο ακολουθείται από τον **τελεστή %** αναμένεται να ακολουθείται από έναν τρόπο μορφοποίησης και τιμές που θα εισαχθούν στο string
- ❑ Τα επιθυμητά σημεία εισαγωγής των τιμών σηματοδοτούνται με την ύπαρξη του χαρακτήρα % μέσα στο string. Αυτό ακολουθείτε με την περιγραφή του τρόπου αναπαράστασης της τιμής που θα εισαχθεί.
- ❑ **%s** σημαίνει εισαγωγή στοιχείου και μορφοποίηση ως string
- %.5f** σημαίνει εισαγωγή στοιχείου ως floating αριθμός με 5 δεκαδικά ψηφία
- %d** σημαίνει εισαγωγή στοιχείου ως integer με βάση το 10