

Συναρτήσεις / Functions

Συναρτήσεις χρήστη

Σε αρκετές περιπτώσεις χρειάζεται να επαναλάβουμε τις ίδιες εντολές κώδικα σε διάφορα σημεία του προγράμματός μας. Στις περιπτώσεις αυτές μπορούμε να ορίσουμε μια δομή η οποία εμπεριέχει κάποιο όνομα που μπορεί να κληθεί σε κάποιο σημείο του προγράμματος που απαιτείται για το αποτέλεσμα ενός υπολογισμού.

Έστω για παράδειγμα, θέλουμε να υπολογίσουμε είτε κάποιο άθροισμα ή το παραγοντικό ενός αριθμού.

Θυμηθείτε ότι το παραγοντικό ενός αριθμού ορίζεται ως:

$$n! = \prod_{k=1}^n k$$

Μπορούμε να γράψουμε στην Python το πρόγραμμα ως

```
#!/usr/bin/python3

# Υπολογισμος του παραγοντικου

f=1.0
n = int(input("Give a number to calculate the factorial "))
for k in range(1,n+1):
    f *= k
print("The factorial of %d is %d" % (n,f))
```

Ωστόσο δεν θέλουμε να επαναλαμβάνουμε τον ίδιο κώδικα αν θέλουμε να υπολογίσουμε το παραγοντικό διάφορων αριθμών

Συναρτήσεις χρήστη

Ένας περισσότερο αποδοτικός τρόπος είναι να γράψουμε μια δική μας συνάρτηση για τον υπολογισμό του παραγοντικού ενός οποιοδήποτε αριθμού που περνάμε ως όρισμα

```
#!/usr/bin/python3

# Υπολογισμός του παραγοντικού με συνάρτηση

def factorial(n): ← Ορισμός μιας function
    f=1.0
    for k in range(1,n+1):
        f *= k
    return f

# Κύριο πρόγραμμα

n = int(input("Give a number to calculate the factorial "))
f = factorial(n) ← Call of the function
print("The factorial of %d is %d" % (n,f))
```

Με την κλήση της συνάρτησης το πρόγραμμα πηγαίνει στο τμήμα που ορίζεται από την εντολή `def factorial(n):`

Ακολουθούν οι υπολογισμοί του παραγοντικού και όταν φθάσει στην τελική γραμμή `return f` επιστρέφει στο σημείο του προγράμματος που κάλεσε την συνάρτηση και η τιμή της συνάρτησης είναι αυτή μετά το `return` που είναι η τελική τιμή της μεταβλητής `f`

Προσοχή: οι μεταβλητές ή οι τιμές τους που ορίζονται μέσα στη συνάρτηση, υπάρχουν όσο είμαστε μέσα στο τμήμα της συνάρτησης και καταστρέφονται μετά **local variables**

Συναρτήσεις χρήστη

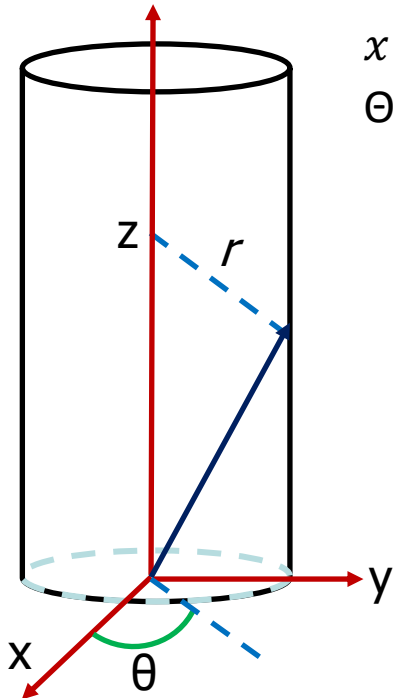
Οι συναρτήσεις μπορούν να έχουν περισσότερα από 1 ορίσματα

Έστω για παράδειγμα θέλουμε να υπολογίσουμε τη απόσταση ενός σημείου από την αρχή του συστήματος συντεταγμένων και το σημείο δίνεται σε κυλινδρικές συντεταγμένες, r , θ , z

Ο απλούστερος τρόπος για να κάνουμε τους υπολογισμούς είναι να μετατρέψουμε τις κυλινδρικές συντεταγμένες σε καρτεσιανές όπως φαίνεται στο διπλανό σχήμα:

$$x = r \cos \theta \quad y = r \sin \theta \quad \text{και} \quad d = \sqrt{x^2 + y^2 + z^2}$$

Θα γράφαμε επομένως μια συνάρτηση ως:



```
#!/usr/bin/python3
```

```
from math import cos, sin, sqrt
```

```
def distance(r,theta,z):
```

```
    x = r*cos(theta)
```

```
    y = r*sin(theta)
```

```
    d = sqrt(x**2 + y**2 + z**2)
```

```
    return d
```

```
# Kyrio programma
```

```
r = float(input("Give the radius of the circle "))
```

```
ang = float(input("Give the azimuthal angle "))
```

```
thez = float(input("Give the z coordinate "))
```

```
dist = distance(r,ang,thez)
```

```
print("The particle is %.5f units for the origin" % dist)
```

Συναρτήσεις χρήστη

Οι συναρτήσεις μπορούν να επιστρέψουν ως αποτέλεσμα οποιοδήποτε τύπο αντικειμένου integer, float, string, complex, list, array.

Θα μπορούσαμε να γράψουμε μια συνάρτηση που μετατρέπει μεταξύ πολικών και καρτεσιανών συντεταγμένων ως εξής:

```
#!/usr/bin/python3

#Metatropi apo polikes se kartesianes suntetagmenes
from math import cos, sin, sqrt,pi

def cartesian(r,theta):
    x = r*cos(theta)
    y = r*sin(theta)
    position = [x,y]
    return position

# Kyrio programma

r = float(input("Give the radius of the circle "))
ang = float(input("Give the azimuthal angle "))
ang = ang*pi/180.
cart = cartesian(r,ang)
print("The particle is @ x = %.5f and y = %.5f" % (cart[0],cart[1]))
□
```

Θα μπορούσαμε αντί να επιστρέψουμε την list απευθείας ως return [x, y]

Θα μπορούσαμε να επιστρέψουμε επίσης **return x,y** και να την καλέσουμε ως **a,b = cartesian(r,theta)**

Συναρτήσεις χρήστη

Οι συναρτήσεις μπορεί να μην επιστρέφουν κάποια τιμή.

Επιτρέπεται οι συναρτήσεις να μην περιέχουν `return` εντολή

Γιατί να θέλει να γράψει κάποιος μια τέτοια συνάρτηση;

Έστω θέλετε να τυπώσετε τις συνιστώσες ενός διανύσματος με μια εντολή της μορφής:

```
print("(" + r[0] + ", " + r[1] + ", " + r[2] + ")")
```

Επειδή είναι αρκετά μπερδεμένη έκφραση, θα μπορούσατε να γράψετε μια συνάρτηση που να κάνει αυτό όταν της περνάτε ένα διάνυσμα:

```
#!/usr/bin/python3

#Metatropi apo polikes se kartesianes suntetagmenes
from math import cos, sin, sqrt, pi

def print_vector(r):
    print("(" + r[0] + ", " + r[1] + ", " + r[2] + ")")

# Kyrio programma

rp = [3,2,1]
print_vector(rp)
```

Συναρτήσεις χρήστη

Οι συναρτήσεις μπορεί να οριστούν οπουδήποτε στο πρόγραμμα αλλά πάντοτε η εντολή **def function_name()** : πρέπει να προηγείται της πρώτης κλήσης της συνάρτησης

Καλή πρακτική να βρίσκονται στην αρχή του προγράμματος που γράφεται

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 λίστα με όνομα newlist με τα αποτελέσματα

Συναρτήσεις χρήστη

Μπορείτε να βάλετε τους ορισμούς διαφόρων συναρτήσεων σε ένα file, π.χ. 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(input("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 ()

Όπως έχουμε χρησιμοποιήσει μέχρι τώρα, 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**.

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

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

output

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 στη θέση 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**)

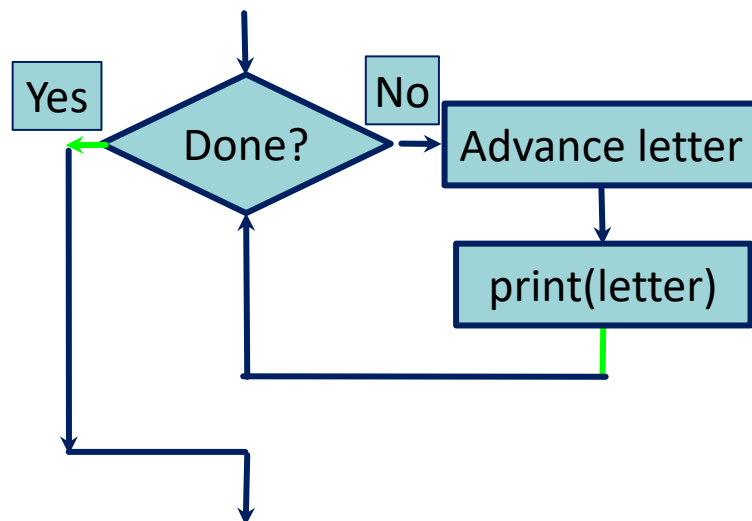
Το κύριο σώμα του loop (οι εντολές του loop) εκτελούνται μια φορά για κάθε τιμή της ακολουθίας

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^ο νούμερο, n1, δίνει τη θέση από την οποία αρχίζουμε και το 2^ο 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^ο 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 αντικειμένου 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 μια **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**