

Δομές Επαναληπτικής διαδικασίας

Επαναληπτικές Διαδικασίες – Βρόχοι - Loops

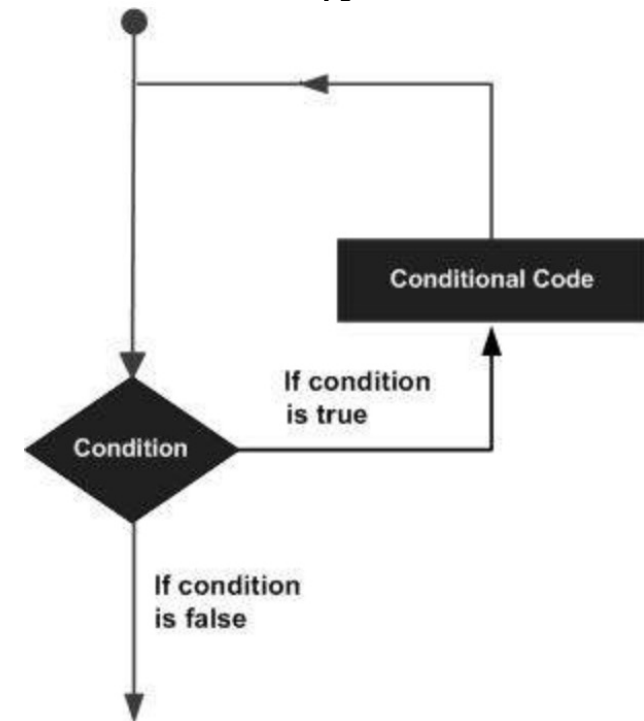
Στους περισσότερους υπολογισμούς θέλουμε να εκτελέσουμε μια σειρά από εντολές αρκετές φορές ώστε να καταλήξουμε στο αποτέλεσμα που θέλουμε.

Φανταστείτε ότι κάποιος μας ζητούσε να υπολογίσουμε τη θέση ενός σώματος συναρτήσει του χρόνου για ένα χρονικό διάστημα 10sec. Θα έπρεπε να χωρίσουμε το χρονικό διάστημα σε μικρότερα υπο-διαστήματα, Δt , και να υπολογίσουμε τη θέση σε κάθε υποδιάστημα. Και την διαδικασία αυτή να την επαναλάβουμε για κάθε Δt

Οι γλώσσες προγραμματισμού προσφέρουν διάφορες δομές που επιτρέπουν για περισσότερο πολύπλοκες εκτελέσεις εντολών και ακολουθιών εκτέλεσης εντολών.

Μια δομή βρόχου ή loop μας επιτρέπει την εκτέλεση μιας εντολής ή ομάδας εντολών πολλές φορές, σύμφωνα με το ακόλουθο διάγραμμα

Για την PYTHON υπάρχουν οι ακόλουθοι τύποι επαναληπτικών διαδικασιών



WHILE Loops

Η διαδικασία αυτή επαναλαμβάνει μια ομάδα εντολών καθόλη τη διάρκεια μια συνθήκη παραμένει αληθής.

Το πρόγραμμα εξετάζει πρώτα αν ικανοποιείται η συνθήκη και αν είναι αληθής εκτελεί την ομάδα των εντολών.

Η σύνταξη της δομής αυτής είναι:

while expression :
statements

Η ομάδα των εντολών (statements) μπορεί να είναι **μια και μόνο** εντολή **ή μια σειρά** από εντολές.

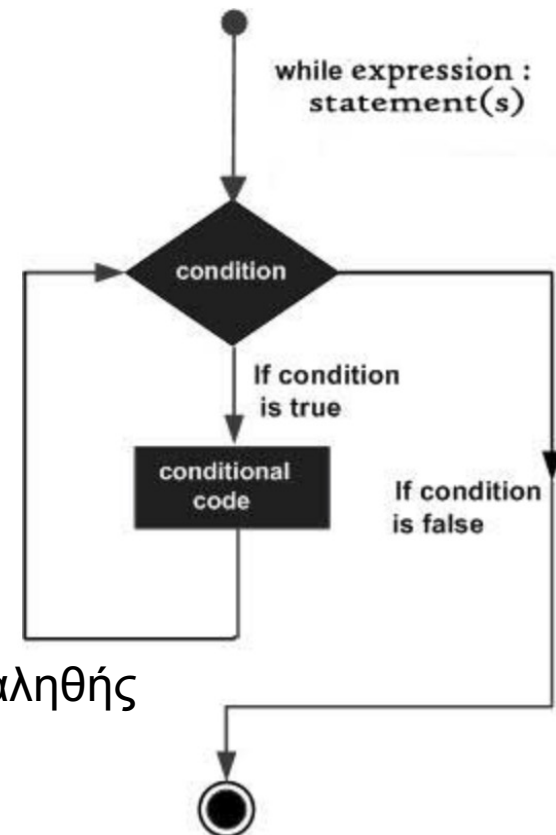
Θα πρέπει όλες να ξεκινούν κάτω από την ίδια στήλη

Η expression μπορεί να είναι οποιαδήποτε λογική έκφραση

Η διαδικασία επαναλαμβάνεται όσο η λογική έκφραση είναι αληθής

Τη στιγμή που έκφραση γίνεται ψευδής η ροή του προγράμματος περνά στην εντολή ακριβώς μετά το loop

Προσοχή: Το loop μπορεί να μην εκτελεστεί ποτέ αν η συνθήκη είναι ψευδής.



WHILE Loops - Παράδειγμα

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

```
count = 0
```

```
while (count < 9):  
    print("The count is:". count)  
    count = count + 1
```



Το block των 2 εντολών print και count εκτελείται έως ότου το count γίνει 8

```
print "Good bye!"
```

```
The count is: 0  
The count is: 1  
The count is: 2  
The count is: 3  
The count is: 4  
The count is: 5  
The count is: 6  
The count is: 7  
The count is: 8  
Good bye!
```

Ατέρμονο loop – Infinite loop

Υπάρχουν περιπτώσεις που η συνθήκη του while loop δεν γίνεται ποτέ ψευδής.

Τότε οι εντολές του loop επαναλαμβάνονται συνεχώς χωρίς να τερματίζεται

Θέλει ιδιαίτερη προσοχή ώστε να αποφεύγονται αυτές οι εσφαλμένες δομές while loop

```
#!/usr/bin/python3

val = 1

while val == 1:      # Auto tha dimiourgisei ena atermono loop
    num = input("Enter a number: ")
    print("You entered:", num)

print("Good bye!")
```

Το αποτέλεσμα του προγράμματος είναι:

```
Enter a number :20
You entered: 20
Enter a number :29
You entered: 29
Enter a number :3
You entered: 3
Enter a number between :Traceback (most recent call last):
  File "test.py", line 5, in <module>
    num = raw_input("Enter a number :")
KeyboardInterrupt
```

Συνεχίζει επ' άπειρο
έως ότου δώσουμε ctr-c:

While Loop – Χρήση εντολής else

Θα μπορούσαμε να συνδυάσουμε την εντολή while με μια εντολή else. Στην περίπτωση αυτή, όταν η συνθήκη του while loop γίνεται ψευδής εκτελούνται οι εντολές που βρίσκονται στο block του else

```
#!/usr/bin/python3

count = 0

while count < 5 :
    print(count, " is less than 5")
    count = count + 1
else :
    print(count, " is not less than 5")
```

Το αποτέλεσμα του κώδικα είναι:

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
```

While Loop – Block μιας εντολής

Θα μπορούσαμε να έχουμε μια και μόνο εντολή που να εκτελείται μετά την συνθήκη της δομής του loop.

Είναι πολύ επικίνδυνη γιατί οδηγεί σε ατέρμονα loop εφόσον δεν υπάρχει εντολή που να αλλάζει την συνθήκη του loop. **Πρέπει να αποφεύγεται**

```
#!/usr/bin/python3  
  
flag = 1  
  
while (flag): print("Given flag is really true!")  
print("Good bye!")
```

Απαιτείται CTRL-C για να σταματήσει η εκτέλεση αυτού του προγράμματος.

FOR Loop – Δεύτερη δομή επανάληψης

Η δομή αυτή του loop εκτελεί την ακολουθία των εντολών πολλές φορές και διαχειρίζεται τον κώδικα που ελέγχει τον αριθμό των επαναλήψεων του loop.

Μπορεί να επαναλάβει κάποιες διαδικασίες χρησιμοποιώντας διάφορες ακολουθίες συμπεριλαμβανομένων αντικειμένων σε lists ή γραμματοσειρές (strings)

Η σύνταξη είναι η ακόλουθη:

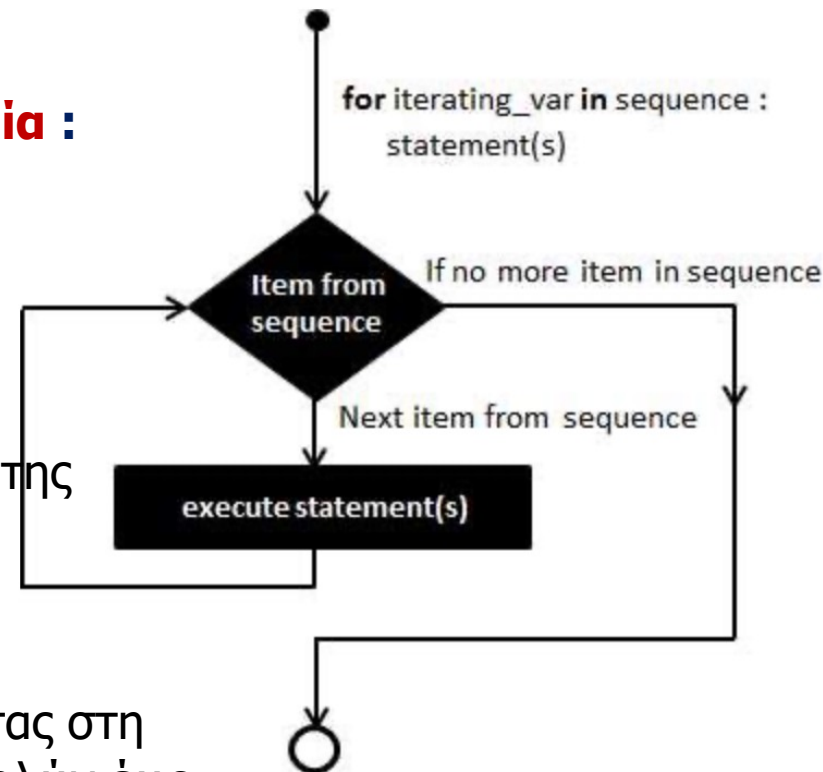
for μεταβλητή_επανάληψης **in** ακολουθία :
statements

Αν η ακολουθία περιέχει μια λίστα από λογικές εκφράσεις ελέγχεται πρώτη

Ακολουθεί η ανάθεση του πρώτου αντικειμένου της λίστας στην μεταβλητή επανάληψης

Εκτελείται μετά το block των εντολών

Ακολουθεί η ανάθεση των αντικειμένων της λίστας στη μεταβλητή επανάληψης και η εκτέλεση των εντολών έως ότου εξαντληθούν τα αντικείμενα της λίστας.



FOR Loop – Παράδειγμα

```
#!/usr/bin/python3

for letter in "Python" :                # 1st example
    print("Current letter is : ",letter)

fruits = ["banana", "apple", "mango"]  # 2nd example
for fruit in fruits:
    print("Current fruit in the list is :",fruit)

print("Good bye!")
```

Εκτέλεση του διπλανού κώδικα δίνει:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

FOR Loop – Χρήση του δείκτη θέσης της ακολουθίας

Θα μπορούσαμε να επαναλάβουμε μια διαδικασία εντολών χρησιμοποιώντας τον δείκτη της θέσης μέσα στην ακολουθία

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

```
fruits = ["banana", "apple", "mango"]  
for index in range(len(fruits)) :  
    print("Current fruit in the list is :", fruits[index])  
  
print("Good bye!")
```

Το πρόγραμμα θα δώσει:

```
Current fruit : banana  
Current fruit : apple  
Current fruit : mango  
Good bye!
```

Χρησιμοποιήσαμε την μέθοδο `len()` για να βρούμε το πλήθος των στοιχείων της λίστας και κατόπιν τη μέθοδο `range()` για να έχουμε την πραγματική ακολουθία των αριθμών της επανάληψης.

Η μέθοδος **`range()`** δημιουργεί μία ακολουθία αριθμών ξεκινώντας από το 0 οι οποίοι αυξάνονται κατά 1 και σταματά πριν κάποιο αριθμό N που δίνουμε ως όρισμα.

Το εύρος των αριθμών είναι επομένως `[0,N)`

Η πλήρης σύνταξη της `range()` είναι: **`range (start, stop, step)`**

Start: προαιρετικός ακέραιος για το που ξεκινά. Αν δεν καθοριστεί είναι 0

Stop: υποχρεωτικός ακέραιος για το που σταματά.

Step: προαιρετικός ακέραιος που δηλώνει πως αυξάνεται η ακολουθία. Κανονικά 1.

Αποτέλεσμα: Ακολουθία ακεραίων στο διάστημα **`[start, stop)`**

FOR Loop – Χρησιμοποίηση else

Θα μπορούσαμε να έχουμε την εντολή else με μια δομή loop.

Αν η εντολή else χρησιμοποιηθεί με for loop, τότε η εντολή εκτελείται όταν το loop έχει εξαντλήσει την λίστα των αριθμών επανάληψης της ακολουθίας.

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

```
for num in range(10, 20) :           #Epanalipsi metaksu 10 kai 20
    for i in range(2, num) :         #Epanalipsi ws pros tous pithanoun diairetes
        if num % i == 0:             #tou arithmou num. Tha einai apo 2 ews num
            j = num/i                # % ypologizei to upoloipo tis diairesis 2
            print("%d equals %d * %d"% (num,i,j))
            break                    # To piliko tis diairesis

            #Diakopi tis epanaliptikis diadikasias gia
            #na metaferthoume ston epomeno num. Stamata
            #diladi tin epanalipsi ws pros to i

else:                                # else einai meros tou LOOP
    print(num, "is a prime number")
    break
```

```
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
```

Nested Loop – Loop μέσα σε loop

Θα μπορούσαμε να έχουμε ένα loop μέσα σε άλλο loop.
Η σύνταξη είναι η ακόλουθη:

```
for μεταβλητή_επανάληψης in ακολουθία :  
    for μεταβλητή_επανάληψης in ακολουθία :  
        statements  
    statements
```

Η σύνταξη για nested while loop είναι η ακόλουθη:

```
while expression :  
    while expression :  
        statements  
    statements
```

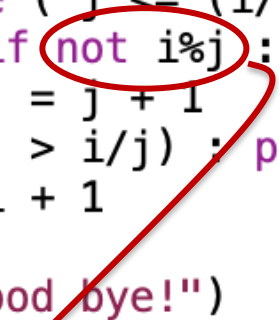
Εν γένει, θα μπορούσε να χρησιμοποιήσει κάποιος οποιοδήποτε είδος loop μέσα σε κάποιο άλλο είδος loop.

Nested Loop – Παράδειγμα

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

```
i = 2
while(i < 100) :
    j = 2
    while ( j <= (i/j) ) :
        if not i%j : break
        j = j + 1
    if (j > i/j) : print(i, " einai prwtos")
    i = i + 1

print("Good bye!")
```



Μια true λογική έκφραση έχει τιμή 1

Μια false λογική έκφραση έχει τιμή 0

Όσο το υπόλοιπο της διαίρεσης του i με το j είναι μεγαλύτερο του 0, η έκφραση $i \% j = \text{true}$ και επομένως η έκφραση: $\text{not } i \% j = \text{false}$

Η επαναληπτική διαδικασία συνεχίζει έως ότου $i \% j = 0 = \text{false}$ (το υπόλοιπο είναι 0) και τότε η έκφραση: $\text{not } i \% j = \text{not false} = \text{true}$

Το αποτέλεσμα θα είναι:

```
>>> exec(open("test1.py").read())
2  einai prwtos
3  einai prwtos
5  einai prwtos
7  einai prwtos
11 einai prwtos
13 einai prwtos
17 einai prwtos
19 einai prwtos
23 einai prwtos
29 einai prwtos
31 einai prwtos
37 einai prwtos
41 einai prwtos
43 einai prwtos
47 einai prwtos
53 einai prwtos
59 einai prwtos
61 einai prwtos
67 einai prwtos
71 einai prwtos
73 einai prwtos
79 einai prwtos
83 einai prwtos
89 einai prwtos
97 einai prwtos
Good bye!
```

Εντολές ελέγχου με τα loops - BREAK

Υπάρχουν εντολές οι οποίες αλλάζουν τη ροή του προγράμματος του loop.
Προσοχή ότι όταν τελειώνει η εκτέλεση όλες οι μεταβλητές που είχαν δημιουργηθεί καταστρέφονται

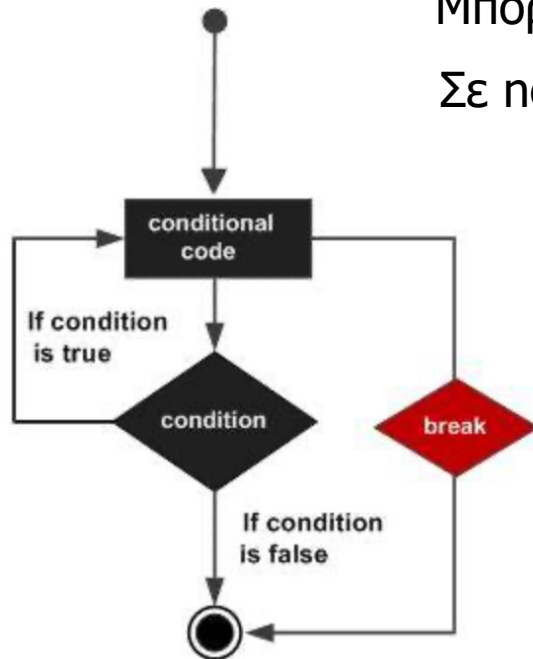
Υπάρχουν 3 εντολές που υποστηρίζει η PYTHON

break εντολή

Σταματά την ακολουθία του loop και μεταφέρει τη ροή στην πρώτη εντολή μετά το loop

Μπορεί να χρησιμοποιηθεί τόσο σε while όσο και σε for loops

Σε nested loops σταματά τη ροή του πιο εσωτερικού loop



BREAK – Παράδειγμα

```
#!/usr/bin/python3

for letter in "Python" :      # 1st example
    if letter == 'h' :
        break
    print("Current letter is :", letter)

var = 10                      # 2nd example
while var > 0:
    print("Current value of variable var = ",var)
    var = var - 1
    if var == 5:
        break

print("Good bye!")
```

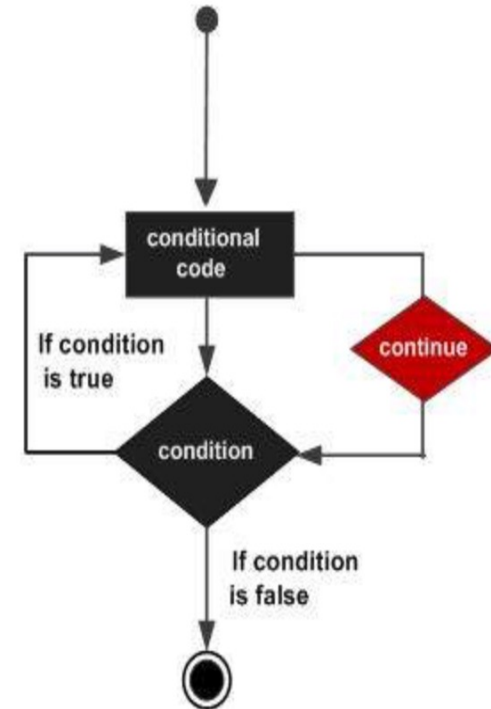
```
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

Εντολές ελέγχου με τα loops - **CONTINUE**

Επιστρέφει τη ροή στην αρχή του while loop

Απορρίπτει όλες τις υπόλοιπες εντολές της ακολουθίας του loop στην τρέχουσα επανάληψη και μετακινεί τη ροή στην αρχή του loop

continue εντολή



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

```
for letter in "Python" :      # 1st example
    if letter == 'h' :
        continue
    print("Current letter is :", letter)

var = 10                      # 2nd example
while var > 0:
    var = var - 1
    if var == 5:
        continue
    print("Current value of variable var = ",var)

print("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```


Εντολές ελέγχου με τα loops - Pass

Χρησιμοποιείται όταν κάποια εντολή χρειάζεται συντακτικά αλλά δεν θέλετε να χρησιμοποιήσετε συγκεκριμένη εντολή

Είναι μια μηδενική εντολή **pass** εντολή

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

```
for letter in "Python" :  
    if letter == 'h' :  
        pass  
        print(" This is pass block")  
    print("Current letter is :", letter)  
  
print("Good bye!")
```

```
Current Letter : P  
Current Letter : y  
Current Letter : t  
This is pass block  
Current Letter : h  
Current Letter : o  
Current Letter : n  
Good bye!
```

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

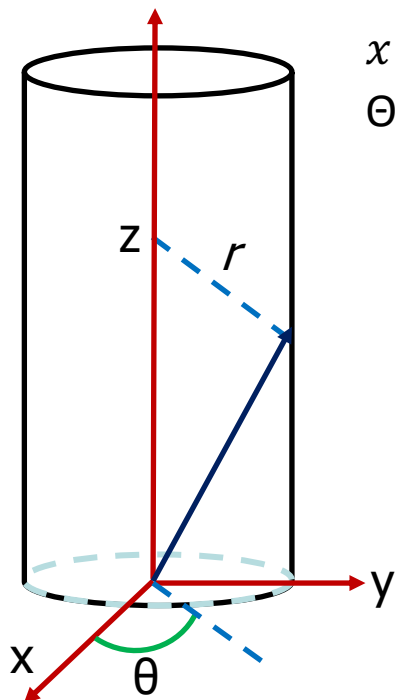
Οι συναρτήσεις μπορούν να έχουν περισσότερα από 1 ορίσματα

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 μετρατρέψουμε 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 τον array απευθείας ως `return array([x,y],float)`

Θα μπορούσαμε να επιστρέψουμε επίσης **return x,y** και να την καλέσουμε ως
α,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 τα αποτελέσματα με όνομα newlist

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.

Έχουμε συναντήσει 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

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.

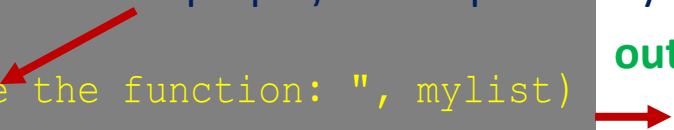
Αυτό σημαίνει ότι όταν αλλάξουμε την τιμή που είναι αποθηκευμένη στην διεύθυνση της μνήμης που αναφέρεται η παράμετρος τότε η αλλαγή αυτή θα εμφανιστεί και στο τμήμα του κώδικα που καλεί την συνάρτηση.

```
#!/usr/bin/python3  
def changeme( mylist ):  
    mylist.append([1,2,3,4])  
    print("Values inside the function: ", mylist) → [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 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()** 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.