

## **Αντικείμενα/Modules/Methods**

## Αντικείμενα και μέθοδοι

Η Python περιέχει διάφορα είδη δεδομένων (**data types**) σαν μέρος της γλώσσας.

Όλα τα δεδομένα αντιπροσωπεύονται από αντικείμενα δεδομένων (**data objects**) και σχέσεις μεταξύ διαφόρων αντικειμένων

Η τιμή ενός object μπορεί να αλλάξει κατά την εκτέλεση ενός προγράμματος.

Objects των οποίων η τιμή μπορεί να αλλάξει λέγονται **mutable** .

Υπάρχουν objects η τιμή των οποίων δεν μπορεί να αλλάξει και τα objects αυτά ονομάζονται **immutable**.

# Modules

Η Python περιέχει μια πληθώρα από συναρτήσεις που μπορούν να χρησιμοποιηθούν με τα διάφορα είδη των objects

Η βιβλιοθήκη της Python περιέχει όλα τα είδη των objects και τις συναρτήσεις που μπορούν να χρησιμοποιηθούν για να έχουμε πρόσβαση στις ιδιότητες των objects.

Ο τρόπος που είναι οργανωμένη η βιβλιοθήκη είναι τέτοιος ώστε να αποτελείται από μια σειρά από **modules** που εμπεριέχουν διάφορες συναρτήσεις

Για να χρησιμοποιήσουμε τα modules, θα πρέπει να κάνουμε **import** το module στο πρόγραμμά μας.

Για παράδειγμα:

```
import numpy as np
```

## Namespace των modules

Η εισαγωγή ενός module, εισάγει και μια σειρά από συναρτήσεις που σχετίζονται με το module που καθορίζουν το λεγόμενο **namespace** του module αυτού

Συναρτήσεις μπορεί να έχουν το ίδιο όνομα αλλά να ανήκουν σε διαφορετικά namespaces γιατί ανήκουν σε διαφορετικά module

Για παράδειγμα τα modules **numpy** και **math** περιέχουν στο namespace τους συναρτήσεις που μπορεί να έχουν το ίδιο όνομα, όπως η συνάρτηση `sqrt()`

Χρειάζεται να δηλώσουμε το namespace (ή το module) στο οποίο ανήκει η συνάρτηση.

Για παράδειγμα:

```
import numpy as np
import math
```

```
A = np.sqrt(10)      # χρήση της συνάρτησης sqrt από το module numpy
B = math.sqrt(10)    # χρήση της συνάρτησης sqrt από το module math
```

Προσοχή ότι η συνάρτηση μπορεί να μην έχει τις ίδιες ιδιότητες ασκούμενη σε κάποιο object που ανήκει στο module

```
A = np.sqrt([4,6,8]) # η sqrt της numpy δρα και στα 3 αντικείμενα της list
B = math.sqrt(10)    # η sqrt του module math δρα μόνο σε 1 αντικείμενο
```

## Αντικείμενα και μέθοδοι – Objects και methods

Μέχρι τώρα είδαμε αναθέσεις τιμών σε μεταβλητές. Αυτό που κάνει η python είναι να συσχετίζει ένα όνομα με ένα αντικείμενο (**objects**). Σε μεταγενέστερη εντολή μπορούμε να αναφερθούμε στο αντικείμενο με το όνομα της μεταβλητής που το συσχετίσαμε.

Το αντικείμενο όμως μπορεί να έχει περισσότερες ιδιότητες από κάποια τιμή.

Ένα object μπορεί να εμπεριέχει δεδομένα (**data**) αλλά και συναρτήσεις (**methods**)

Η τιμή ενός object αποτελεί ένα από τα δεδομένα του object

Η method ενός object είναι μια συνάρτησή του που ενεργεί στα δεδομένα του object.

Η method μπορεί να δέχεται και περισσότερα δεδομένα

Για να χρησιμοποιήσουμε μία μέθοδο ενός αντικειμένου θα πρέπει να:

- Δώσουμε το όνομα του αντικειμένου
- Ακολουθούμενο από μια τελεία .
- Το όνομα της μεθόδου
- Ένα ζευγάρι παρενθέσεις που περιέχουν τα δεδομένα (ορίσματα της μεθόδου)

## Είδος μεταβλητής

Μπορούμε να μάθουμε τι είδους είναι μια μεταβλητή χρησιμοποιώντας την συνάρτηση `type()`

Μπορούμε να δούμε όλες τις συναρτήσεις (methods) που σχετίζονται με τον τύπο ενός αντικειμένου χρησιμοποιώντας την εντολή `dir(object name)` ή `dir(object type)`

Για παράδειγμα, από το διπλανό κώδικα θα μπορούσαμε να έχουμε:

```
>>> dir(eee)
>>> dir(type(eee))
```

Θα μπορούσαμε να ζητήσουμε τις methods που περιέχονται στο module `numpy` ή κάποιο άλλο module

```
>>> import numpy as np
>>> dir(np)
>>> import math
>>> dir(math)
```

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last): File
"<stdin>", line 1, in <module>TypeError:
Can't convert 'int' object to str implicitly
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
>>>
```

## Παράδειγμα Objects με methods

Μπορούμε να εξετάσουμε μεθόδους ενός float object `f`, ενός object `s` τύπου string

- `f.is_integer()` Κάθε αντικείμενο τύπου float έχει μια μέθοδο που μας επιτρέπει να εξετάσουμε αν η τιμή του αντικειμένου είναι integer. Η μέθοδος επιστρέφει TRUE αν το δεκαδικό μέρος είναι μηδέν και FALSE διαφορετικά. Δεν απαιτεί κάποια επιπλέον τιμή αλλά πρέπει να υπάρχει το ζευγάρι των παρενθέσεων.
- `s.swapcase()` Κάθε αντικείμενο τύπου string έχει μια μέθοδο που μας επιστρέφει μια νέα string τιμή που η τιμή της είναι μια γραμματοσειρά με τους χαρακτήρες της αρχικής γραμματοσειράς σε αντίθετη κατάσταση (γράμματα από μικρά – κεφαλαία ή κεφαλαία σε μικρά)

```
>>>  
>>> f=3.2  
>>> f.is_integer()  
False  
>>>  
>>>  
>>> s="python"  
>>> s.swapcase()  
'PYTHON'  
>>> □
```

## **Lists/Tuples**



## Αντικείμενα τύπου λίστας: **list type objects**

Αντικείμενα τύπου που ορίζονται με τετραγωνική αγκύλη [ ] και περιέχουν δεδομένα χωρισμένα με , αποτελούν τύπο **list**.

**L = [1, 2, 3]** Λίστα με ακεραίους αφού τα δεδομένα είναι ακέραιοι. Η τιμή του αντικειμένου είναι μια ακολουθία 3 ακεραίων αντικειμένων

Μια λίστα μπορεί να περιέχει 1 μόνο αντικείμενο π.χ. [2.7] το οποίο είναι διαφορετικό Από το 2.7 ως float αντικείμενο.

Μια λίστα μπορεί να είναι άδεια π.χ. []

Ένα object τύπου list μας επιτρέπει να κάνουμε πολλαπλή ανάθεση τιμών σε κάποιες μεταβλητές οι οποίες μοιράζονται ουσιαστικά το ίδιο όνομα

Αυτό κάνει ιδιαίτερα εύχρηστα τα αντικείμενα αυτού του τύπου αφού δεν χρειάζεται να δηλώσουμε πολλές μεταβλητές για να αποθηκεύσουμε τα δεδομένα τους

Τα δεδομένα των objects αυτών μπορεί να έχουν κοινό τύπο αλλά όχι απαραίτητα

## list type objects

Τα list type objects σχετίζονται με μια σειρά από μεθόδους

- **L.reverse()** Είναι μία μέθοδος του αντικειμένου τύπου list που επιστρέφει τη λίστα σε ανάποδη σειρά
- **L.pop(n)** Είναι μία μέθοδος του αντικειμένου τύπου list που επιστρέφει το αντικείμενο που βρίσκεται στην n-θέση της list (**ξεκινώντας το μέτρημα από τη θέση 0**) **και το αφαιρεί από τη λίστα**  
Καλώντας τη μέθοδο pop() χωρίς όρισμα θα αφαιρέσει το πρώτο στοιχείο της list

```
>>> L = [2,5,8,9,11]
>>> L.reverse()
>>> print(L)
>>> L.pop(2)
>>> print(L)
>>> L.pop()
>>> print(L)
```

[11, 9, 8, 5, 2]

8 (το 3<sup>ο</sup> στοιχείο)

[11, 9, 5, 2]

2 (το τελευταίο στοιχείο που είναι η default περίπτωση)

[11, 9, 5]

## List type objects

Τα αντικείμενα τύπου list, είναι οι πλέον εύχρηστοι σύνθετοι τύποι δεδομένων.

Μια **λίστα** περιέχει διάφορες **σταθερές** διαχωρισμένες με **,** και περιέχονται σε **[ ]**.

Μια λίστα μοιάζει με πίνακες δεδομένων με τη διαφορά ότι τα στοιχεία που αποτελούν τη λίστα δεν είναι απαραίτητα του ίδιου τύπου.

Μπορούμε να αποκτήσουμε πρόσβαση στα στοιχεία μιας λίστας χρησιμοποιώντας τον τελεστή **[ ]** και **[:]** και κάποιον δείκτη ή αρίθμηση του οποίου ξεκινά από το 0

Χρησιμοποιώντας αρνητικό δείκτη θέσης, τότε η αρίθμηση ξεκινά από τα δεξιά της λίστας **[-n]**

Ο τελεστής **+** είναι ο τελεστής σύνθεσης δύο ή περισσότερων αντικειμένων τύπου list


Ο τελεστής **\*** είναι ο τελεστής επανάληψης

Η μέθοδος **len(listname)** δίνει το πλήθος των στοιχείων της λίστας

## List type objects

```
>>> mylist = ['abc', 234, 1.42, 'mary', 45.]
>>> alist = [123, 'john']
>>> print(mylist)           #Τυπώνει τη λίστα ['abc', 234, 1.42, 'mary', 45.0]
>>> print(mylist[0])        #Τυπώνει το 1ο στοιχείο της λίστας abc
>>> print(mylist[1:3])      #Τυπώνει από το 2ο έως και το 3ο στοιχείο της λίστας [234, 1.42]
>>> print(mylist[2:])       #Τυπώνει ξεκινώντας από το 3ο στοιχείο της λίστας [1.42, 'mary', 45.0]
>>> print(alist*2)          #Τυπώνει 2 φορές τη λίστα alist [123, 'john', 123, 'john']
>>> newlist=mylist+alist    #Σύνθεση νέας λίστας από τις δύο λίστες
>>> print(newlist)          #Εκτύπωση της νέας λίστας ['abc', 234, 1.42, 'mary', 45.0, 123, 'john']
```

```
>>> mylist = ['physics', 'chemistry', 1921, 2011]
>>> print(mylist[-1])       #Πρόσβαση σε στοιχείο της λίστας από δεξιά προς τα αριστερά
>>> print(len(mylist))      #Πλήθος στοιχείων λίστας
```




2021

4

## List type objects – Αλλαγές στη λίστα

Μπορούμε να αλλάξουμε τα στοιχεία μιας λίστας είτε κάνοντας ανάθεση σε κάποια θέση της λίστας ή χρησιμοποιώντας την εντολή **append()**

```
>>> mylist = ['physics', 'chemistry', 1921, 2021]
>>> print ("Η timi tis listas sti thesi 2 einai: ", list[1])
>>> mylist[1] = 1821    #Ανάθεση νέας τιμής σε στοιχείο της λίστας
>>> print("Η nea timi tis listas sti thesi 2 einai: ", list[1])
```

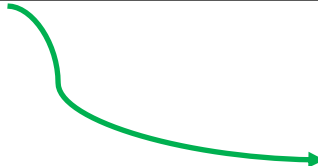


```
Η timi tis listas sti thesi 2 einai: chemistry
Η nea timi tis listas sti thesi 2 einai: 1821
```

## List type objects – Διαγραφή στοιχείου λίστας

Μπορούμε να διαγράψουμε στοιχεία μιας λίστας είτε χρησιμοποιώντας την εντολή **del** αν γνωρίζουμε ποιο στοιχείο θέλουμε να διαγράψουμε ή την μέθοδο **remove(value)** όταν δεν ξέρουμε το στοιχείο (διαγράφει το πρώτο στοιχείο της list που έχει την τιμή value)

```
>>> mylist = [ 12, 34, 23, 12]
>>> print("Η timi tis listas sti thesi 2 einai: ", mylist[1])
>>> del mylist[1]    #Διαγραφή στοιχείου της λίστας
>>> print("Η lista twra einai:", mylist)
>>> remove(12)      #Διαγραφή του πρώτου στοιχείου της λίστας με τιμή 12
>>> print("Η lista twra einai:", mylist)
```



```
Η timi tis listas sti thesi 2 einai: 34
Η lista einai twra [12, 23, 12]
Η lista einai twra [23, 12]
```

## List type objects – Συναρτήσεις λίστας

Υπάρχουν μέθοδοι που μπορούμε να χρησιμοποιήσουμε με λίστες όπως:

- `max(listname):` Επιστρέφει το μέγιστο από τα στοιχεία μιας λίστας
- `min(listname):` Επιστρέφει το ελάχιστο από τα στοιχεία μιας λίστας
- `list(tuple):` Μετατρέπει ένα αντικείμενο tuple σε λίστα
- `cmp(list1,list2):` Συγκρίνει τα στοιχεία δύο λιστών
- `len(list1,list2):` Επιστρέφει το μέγεθος της λίστας
- `listname.append(obj):` Προσθέτει στο τέλος της λίστας listname το obj
- `listname.count(obj):` Επιστρέφει τη συχνότητα εμφάνισης του obj
- `listname.extend(tuple):` Προσθέτει στο τέλος της λίστας το tuple
- `listname.index(obj):` Επιστρέφει τον μικρότερο δείκτη της θέσης του obj
- `listname.insert(index,obj):` Εισάγει το obj στη θέση index
- `listname.pop(obj):` Διαγράφει και τυπώνει το obj από τη λίστα
- `listname.remove(obj):` Διαγράφει το obj από τη λίστα
- `listname.reverse():` Αντιστρέφει τη σειρά των δεδομένων στη λίστα
- `listname.sort([func]):` Ταξινομεί τα δεδομένα στη λίστα σύμφωνα με τη func αν δίνεται

## Tuple type objects

Τα αντικείμενα τύπου **tuple**, είναι ένα άλλο είδος λίστας δεδομένων που μοιάζει με τα αντικείμενα τύπου list.

Ένα αντικείμενο τύπου **tuple** περιέχει διάφορες **σταθερές** χωρισμένες με **,**  
**Σε αντίθεση με τις lists, οι τιμές είναι εγκλεισμένες σε ( ).**

Ενώ τα στοιχεία μιας λίστας μπορεί να αλλάξουν τιμή ή η list να μεγαλώσει τα **tuples δεν μπορούν να τροποποιηθούν (immutable objects).**

Επομένως μπορούν να χρησιμοποιηθούν μόνο για ανάγνωση των στοιχείων τους και δεν μπορούμε να τα αλλάξουμε

Μπορούμε να αποκτήσουμε πρόσβαση στα στοιχεία ενός tuple χρησιμοποιώντας τον τελεστή **[ ]** και **[:]** και κάποιον δείκτη η αρίθμηση του οποίου ξεκινά από το 0

Ο τελεστής **+** είναι ο τελεστής σύνθεσης δύο ή περισσότερων αντικειμένων τύπου tuple

Ο τελεστής **\*** είναι ο τελεστής επανάληψης

Ισχύουν οι ίδιες συναρτήσεις και μέθοδοι όπως και στις λίστες.

Δεν μπορούμε να εφαρμόσουμε την διαγραφή ενός στοιχείου (`del tuple[1]`) αλλά μόνο όλου του tuple: `del mytuple`

Υπάρχει μέθοδος μετατροπής λίστας σε tuple: **`tuple(listname)`**



## Tuple type objects

```
>>> mytuple = ('abc', 234, 1.42, 'mary', 45.)
>>> atuple = (123, 'john')
>>> print(mytuple)           #Τυπώνει το tuple ['abc', 234, 1.42, 'mary', 45.0]
>>> print(mytuple[0])        #Τυπώνει το 1° στοιχείο του tuple abc
>>> print(mytuple[1:3])       #Τυπώνει από το 2° έως και το 3° στοιχείο του tuple [234, 1.42]
>>> print(mytuple[2:])        #Τυπώνει ξεκινώντας από το 3° στοιχείο του tuple [1.42, 'mary', 45.0]
>>> print(atuple*2)           #Τυπώνει 2 φορές του tuple atuple [123, 'john', 123, 'john']
>>> newtuple = mytuple + atuple #Σύνθεση νέου tuple από τα δύο tuples
>>> print(newtuple)           #Εκτύπωση του νέου tuple ['abc', 234, 1.42, 'mary', 45.0, 123, 'john']
```

```
Mytuple = (12, 23, 45, 7)
```

```
Mylist = [45, 6, 45]
```

```
Mytuple[2] = 1000 #Λανθασμένη χρήση – Δεν μπορούμε να αλλάξουμε στοιχείο tuple
```

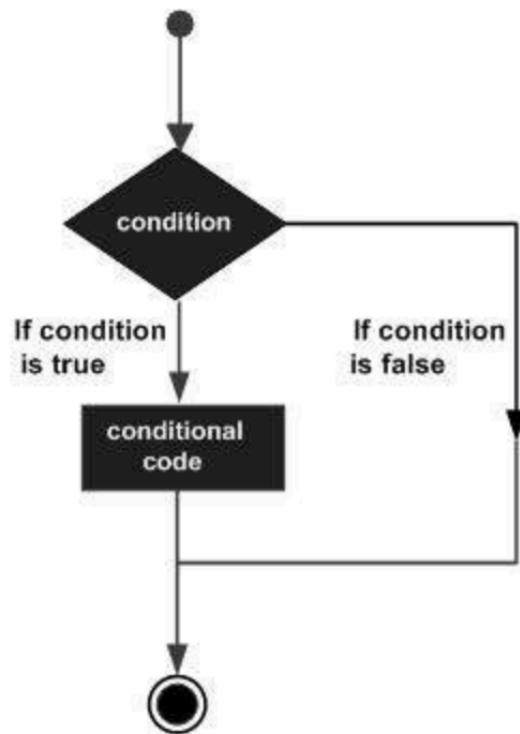
```
Mylist[2] = 100    #Σωστή χρήση τύπου list
```

## Δομές Συνθήκης

# Εντολές συνθήκης

Στα περισσότερα προγράμματα χρειάζεται να εξετάσουμε αν ισχύουν συγκεκριμένες συνθήκες για να αλλάξουμε τη ροή του προγράμματος.

Αν η συνθήκη που ελέγχεται είναι TRUE τότε εκτελούνται μια σειρά από εντολές ενώ αν είναι FALSE εκτελούνται κάποιες άλλες εντολές



**Εντολή if:** Αποτελείται από μια λογική έκφραση και ακολουθείται από μια σειρά εντολών

**Εντολή if else ...:**

Μια λογική έκφραση ακολουθείται από μια σειρά εντολών και αν είναι FALSE μια σειρά άλλων εντολών ακολουθείται

**Εντολή πολλαπλών if else if else if ...else ...:**

Εξέταση διαφόρων περιπτώσεων

## Εντολές συνθήκης

```
#!/usr/bin/python
```

```
var = 100
```

```
if ( var == 100 ) : print "Value of expression is 100"
```

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

Value of expression is 100

Good bye!

# Εντολές συνθήκης if .... Else

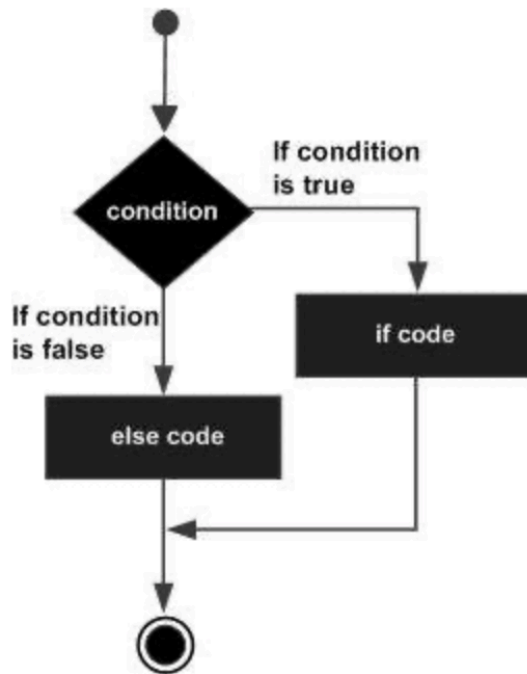
Η σύνταξη είναι:

If λογική εκφραση :

Εντολές

else :

Εντολές



```
#!/usr/bin/python
```

```
var1 = 100
```

```
if var1:
```

```
    print "1 - Got a true expression value"
```

```
    print var1
```

```
else:
```

```
    print "1 - Got a false expression value"
```

```
    print var1
```

```
var2 = 0
```

```
if var2:
```

```
    print "2 - Got a true expression value"
```

```
    print var2
```

```
else:
```

```
    print "2 - Got a false expression value"
```

```
    print var2
```

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

```
1 - Got a true expression value
```

```
100
```

```
2 - Got a false expression value
```

```
0
```

```
Good bye!
```

## Εντολές συνθήκης **if .... elif ... elif ... else**

Η σύνταξη είναι:

If λογική έκφραση :

Εντολές

elif λογική έκφραση1:

Εντολές

elif λογική έκφραση2:

Εντολές

else:

Εντολές

```
#!/usr/bin/python
```

```
var = 100
if var == 200:
    print "1 - Got a true expression value"
    print var
elif var == 150:
    print "2 - Got a true expression value"
    print var
elif var == 100:
    print "3 - Got a true expression value"
    print var
else:
    print "4 - Got a false expression value"
    print var

print "Good bye!"
```

```
3 - Got a true expression value
100
Good bye!
```

# Λογικοί Τελεστές

Οι λογικοί τελεστές που χρησιμοποιούνται στην PYTHON είναι:

**AND** Αν και οι δύο συνθήκες είναι TRUE τότε το αποτέλεσμα (**a and b**) είναι TRUE

**OR** Αν τουλάχιστον η μία από τις δύο συνθήκες είναι TRUE τότε το αποτέλεσμα (**a or b**) είναι TRUE

**NOT** Χρησιμοποιείται για να αντιστρέψει το λογικό αποτέλεσμα του αντικειμένου στο οποίο ενεργεί: **not (a and b)** είναι FALSE

```
a = 5
b = 6
if a > 0 and b < 10:
    print("Passes")
else:
    print("Fails")
```

```
a = 5
b = 6
if a > 6 or b < 10:
    print("Passes")
else:
    print("Fails")
```

```
a = 5
b = 6
if not (a > 0 and b < 10):
    print("Passes")
else:
    print("Fails")
```

## Η δομή `try/except`

Η δομή με `try/except` χρησιμοποιείται για να περικλείουμε επικίνδυνα τμήματα του κώδικα που δεν ξέρουμε τι ακριβώς γίνεται

Αν το τμήμα του κώδικα μέσα στο `try` δουλεύει τότε το τμήμα που περικλείεται στο `except` δεν εκτελείται

Αν το τμήμα του κώδικα μέσα στο `try` δεν δουλεύει τότε εκτελείται το τμήμα που περικλείεται στο `except`

```
$ python3 notry.py
```

```
Traceback (most recent call last): File "notry.py", line 2,  
in <module>   istr = int(astr)ValueError: invalid literal for  
int() with base 10: 'Hello Bob'
```

```
astr = 'Hello Bob'  
istr = int(astr)  
print('First', istr)  
astr = '123'  
istr = int(astr)  
print('Second', istr)
```

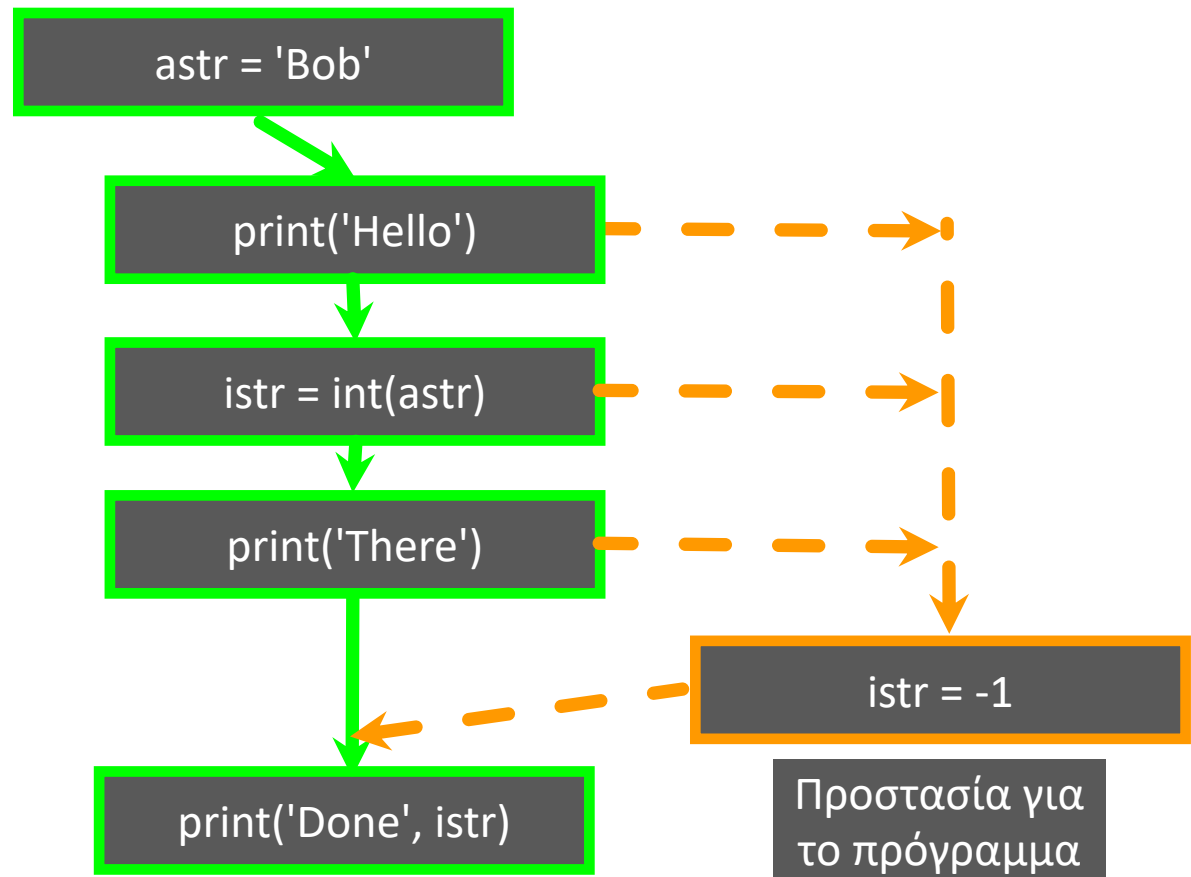
πρόβλημα

error



# try / except

```
astr = 'Bob'  
try:  
    print('Hello')  
    istr = int(astr)  
    print('There')  
except:  
    istr = -1  
  
print('Done', istr)
```



## try / except

```
rawstr = input('Enter a number:')
try:
    ival = int(rawstr)
except:
    ival = -1

if ival > 0 :
    print('Nice work')
else:
    print('Not a number')
```

```
$ python3 trynum.py
Enter a number:42
Nice work
$ python3 trynum.py
Enter a number:forty-two
Not a number
$
```

## 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,  $\Delta t$ , και να υπολογίσουμε τη θέση σε κάθε υποδιάστημα. Και την διαδικασία αυτή να την επαναλάβουμε για κάθε  $\Delta t$

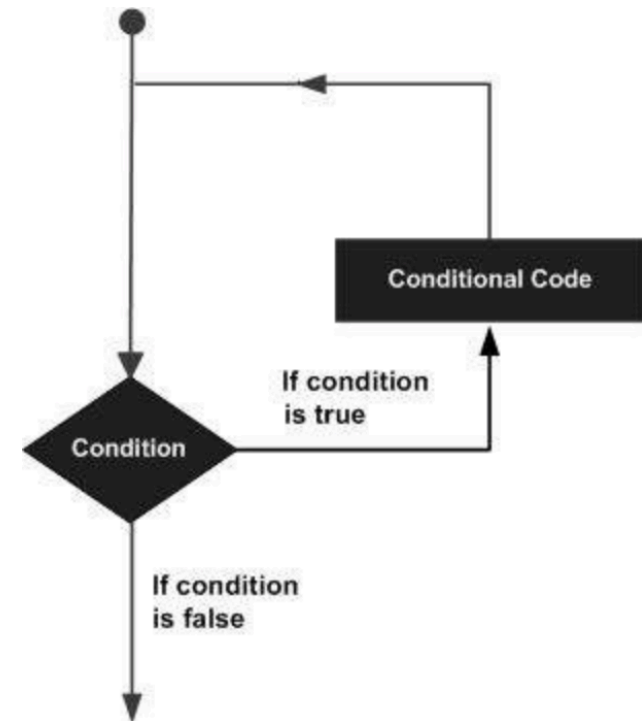
Οι γλώσσες προγραμματισμού προσφέρουν διάφορες δομές που επιτρέπουν για περισσότερο πολύπλοκες εκτελέσεις εντολών και ακολουθιών εκτέλεσης εντολών.

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** (αόριστο loop)

**for loops** (προσδιορισμένο loop)



# WHILE Loops

Η διαδικασία αυτή επαναλαμβάνει μια ομάδα εντολών καθ' όλη τη διάρκεια που 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.

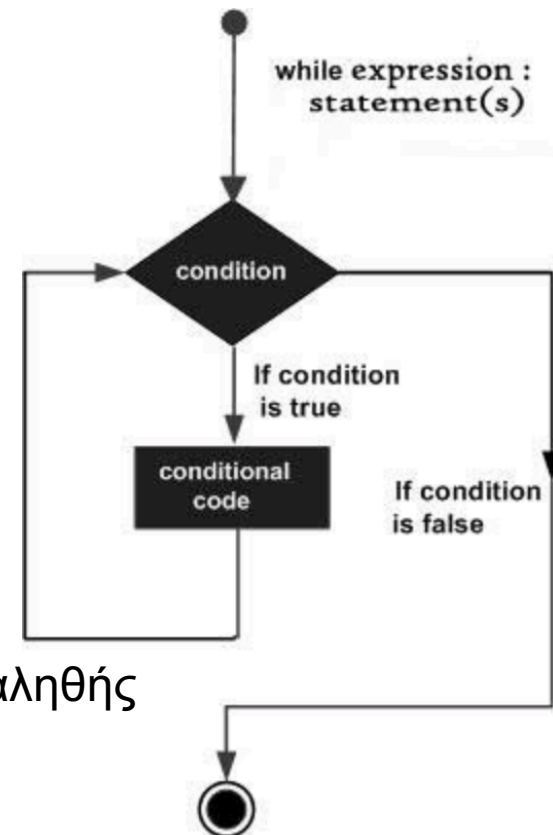
Θα πρέπει όλες να ξεκινούν κάτω από την ίδια στήλη

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/python
```

```
count = 0
while (count < 9):
    print 'The count is:', count
    count = count + 1

print "Good bye!"
```

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

```
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/python

var = 1
while var == 1 : # This constructs an infinite loop
    num = raw_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/python
```

```
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/python

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:

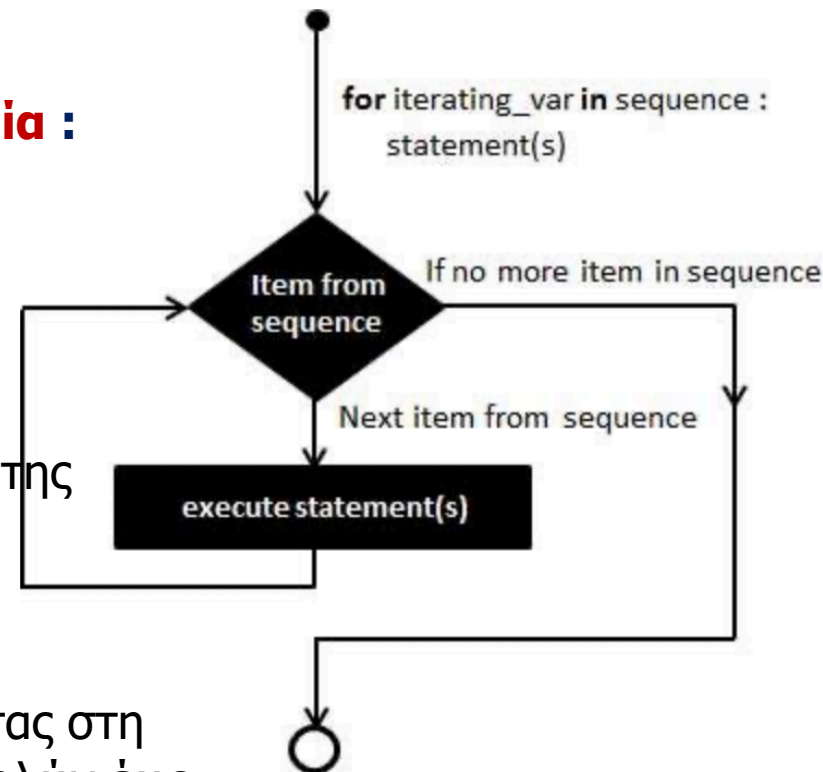
**for** μεταβλητή\_επανάληψης **in** ακολουθία :  
**statements**

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

Εκτέλεση του διπλανού κώδικα δίνει:

```
#!/usr/bin/python

for letter in 'Python':    # First Example
    print 'Current Letter :', letter

fruits = ['banana', 'apple', 'mango']
for fruit in fruits:      # Second Example
    print 'Current fruit :', fruit

print "Good bye!"
```

```
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/python

fruits = ['banana', 'apple', 'mango']
for index in range(len(fruits)):
    print 'Current fruit :', 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/python
```

```
for num in range(10,20):      #to iterate between 10 to 20
    for i in range(2,num):    #to iterate on the factors of the number
        if num%i == 0:       #to determine the first factor
            j=num/i           #to calculate the second factor
            print '%d equals %d * %d' % (num,i,j)
            break #to move to the next number, the #first FOR
    else:                     # else part of the 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/python

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, " is prime"
    i = i + 1

print "Good bye!"
```

Το αποτέλεσμα θα είναι:

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
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

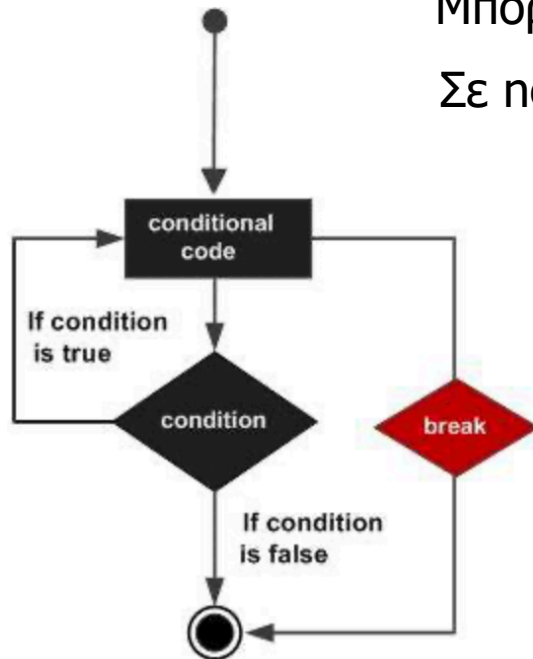
Υπάρχουν 3 εντολές που υποστηρίζει η PYTHON

### **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/python
```

```
for letter in 'Python':      # First Example
    if letter == 'h':
        break
    print 'Current Letter :', letter

var = 10                      # Second Example
while var > 0:
    print 'Current variable value :', var
    var = var -1
    if var == 5:
        break

print "Good bye!"
```

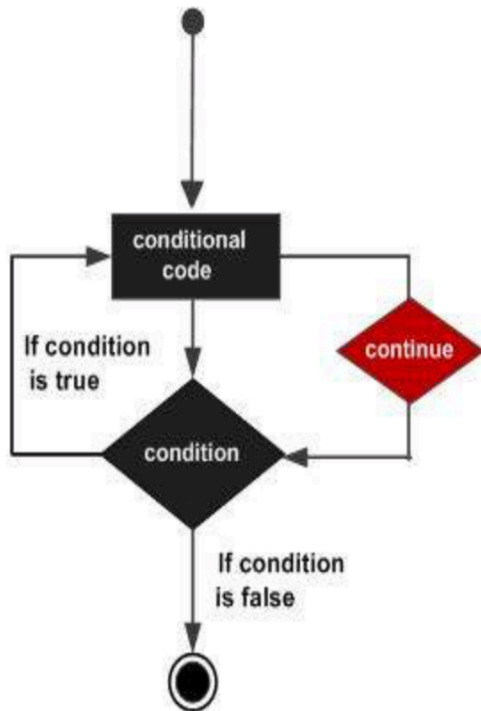
```
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

# 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/python
```

```
for letter in 'Python':    # First Example
    if letter == 'h':
        continue
    print 'Current Letter :', letter
```

```
var = 10                    # Second Example
while var > 0:
    var = var -1
    if var == 5:
        continue
    print 'Current variable value :', 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/python

for letter in 'Python':
    if letter == 'h':
        pass
    print 'This is pass block'
    print 'Current Letter :', 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!
```