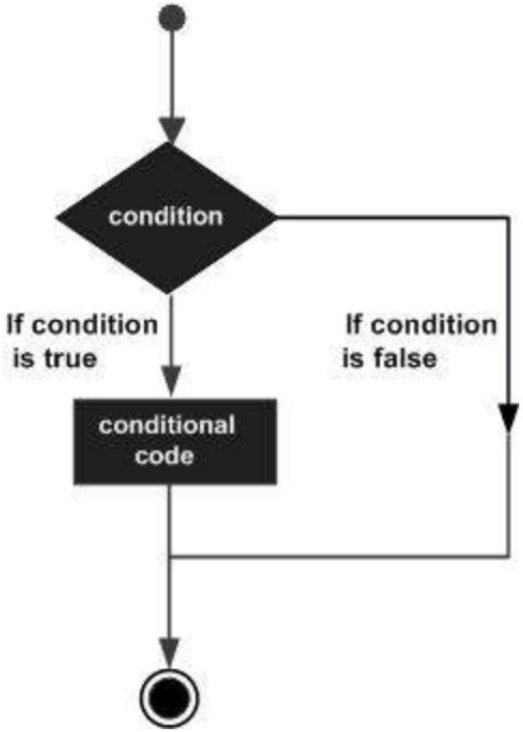
Δομές Συνθήκης

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 που ακολουθείται από μια σειρά εντολών ενώ αν είναι FALSE ακολουθείται από μια σειρά άλλων εντολών

Εντολή πολλάπλών if else if else if …else …:

Εξέταση διαφόρων περιπτώσεων

Εντολές συνθήκης

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

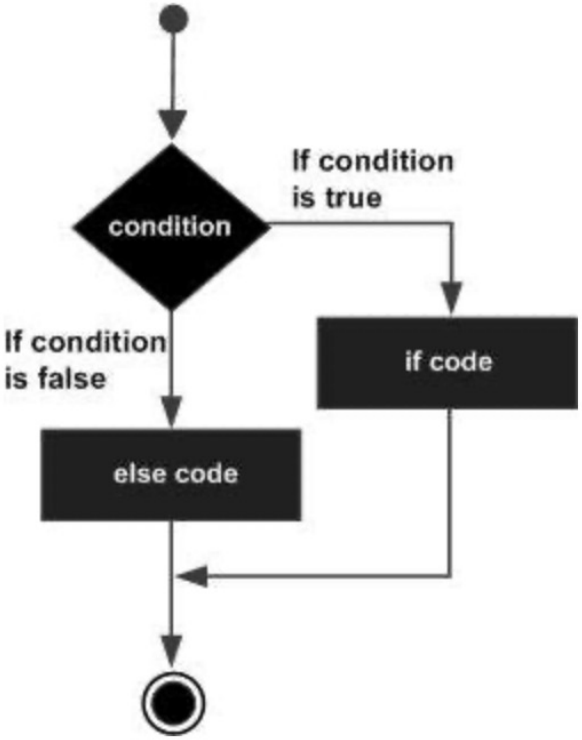
```
Value of expression is 100
Good bye!
```

Εντολές συνθήκης if …. Else

Η σύνταξη είναι:

If λογική έκφραση :
  Εντολές

else :
  Εντολές




```
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 εκφραση :
  Εντολές
elif λογική έκφραση1:
  Εντολές
elif λογική έκφραση2:
  Εντολές
else:
  Εντολές

```
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!
```

Τελεστές σύγκρισης

Οι τελεστές σύγκρισης που χρησιμοποιούνται στην PYTHON είναι:

== Αν οι τιμές δύο αντικειμένων είναι ίσες
τότε το αποτέλεσμα (a == b) είναι TRUE

!= Αν οι τιμές δύο αντικειμένων **δεν** είναι ίσες
τότε το αποτέλεσμα (a != b) είναι TRUE

<> Αν οι τιμές δύο αντικειμένων **δεν** είναι ίσες
τότε το αποτέλεσμα (a <> b) είναι TRUE

> Αν η τιμή του αριστερού αντικειμένου είναι μεγαλύτερη του δεξιού
τότε το αποτέλεσμα (a > b) είναι TRUE

< Αν η τιμή του αριστερού αντικειμένου είναι μικρότερη του δεξιού
τότε το αποτέλεσμα (a < b) είναι TRUE

>= Αν η τιμή του αριστερού αντικειμένου είναι μεγαλύτερη ή ίση του δεξιού
τότε το αποτέλεσμα (a >= b) είναι TRUE

<= Αν η τιμή του αριστερού αντικειμένου είναι μικρότερη ή ίση του δεξιού
τότε το αποτέλεσμα (a <= b) είναι TRUE

Παράδειγμα

```
a = 21
b = 10
c = 0

if ( a == b ):
   print "Line 1 - a is equal to b"
else:
   print "Line 1 - a is not equal to b"

if ( a != b ):
   print "Line 2 - a is not equal to b"
else:
   print "Line 2 - a is equal to b"

if ( a <> b ):
   print "Line 3 - a is not equal to b"
else:
   print "Line 3 - a is equal to b"

if ( a < b ):
   print "Line 4 - a is less than b"
else:
   print "Line 4 - a is not less than b"

if ( a > b ):
   print "Line 5 - a is greater than b"
else:
   print "Line 5 - a is not greater than b"

a = 5;
b = 20;
if ( a <= b ):
   print "Line 6 - a is either less than or equal to  b"
else:
   print "Line 6 - a is neither less than nor equal to  b"

if ( b >= a ):
   print "Line 7 - b is either greater than  or equal to b"
else:
   print "Line 7 - b is neither greater than  nor equal to b
```

Το αποτέλεσμα του διπλανού κώδικα είναι:

```
Line 1 - a is not equal to b
Line 2 - a is not equal to b
Line 3 - a is not equal to b
Line 4 - a is not less than b
Line 5 - a is greater than b
Line 6 - a is either less than or equal to b
Line 7 - b is either greater than or equal to b
```

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

Η δομή try/except

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

```
$ 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'
```

error

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

πρόβλημα

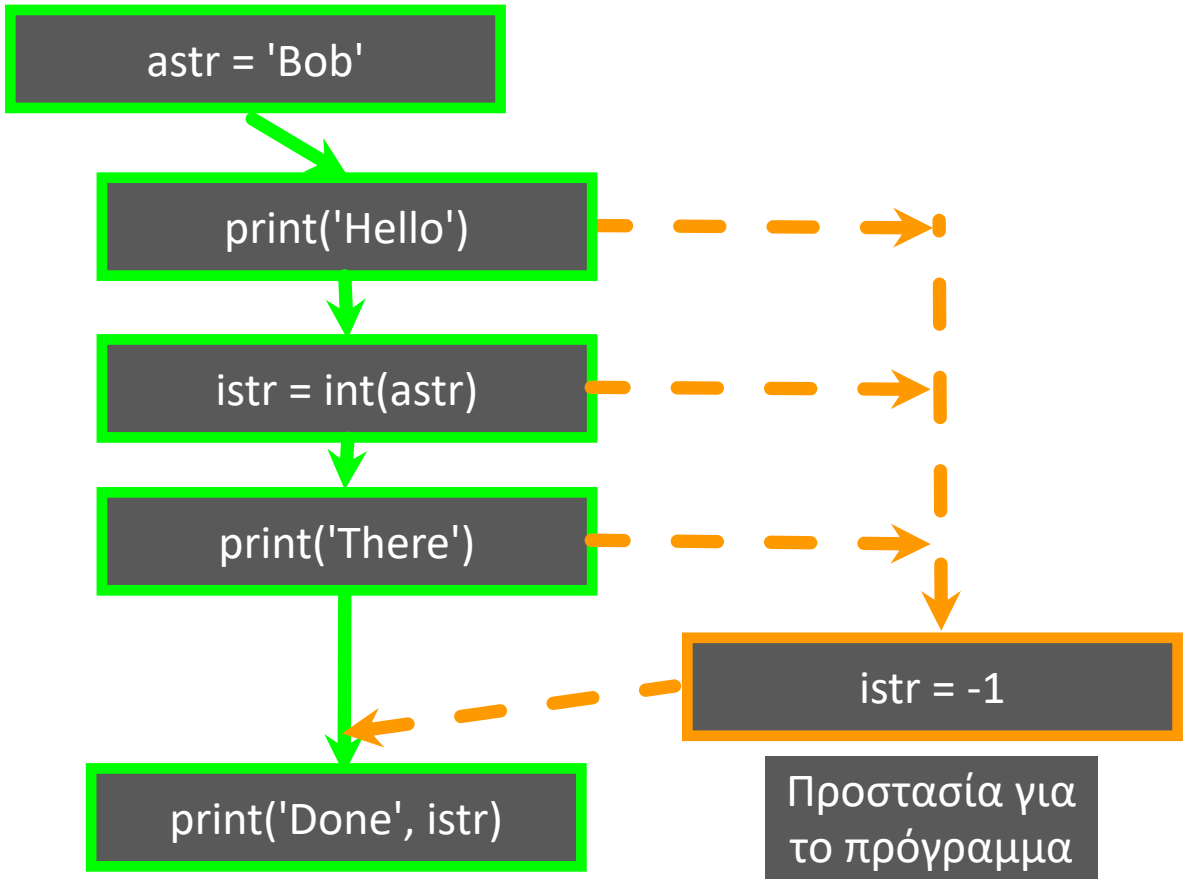
try / except

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

```
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
$
```

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)

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 εξέτασουμε 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'
>>>
```

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. []

- 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 τελευταίο στοιχείο της list

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

>>> print(L)
[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 η 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

test1.py

```
#!/usr/bin/python3

mylist = ['abc', 234, 1.42, 'mary', 45.]
alist = [123, 'john']

print(mylist)
print(mylist[0])
print(mylist[1:3])
print(mylist[2:])
print(alist*2)
newlist = mylist + alist
print(newlist)
```

Ορισμός λίστας. Θα μπορούσαμε να έχουμε mylist = [] – η λιστα με μηδενικό περιεχόμενο

Τυπώνει τη λίστα
Τυπώνει το 1ο στοιχείο της λίστας
Τυπώνει από το 2ο έως και το 3ο στοιχείο της λίστας
Τυπώνει ξεκινώντας από το 3ο στοιχείο της λίστας
Τυπώνει 2φορές τη λίστα alist
Σύνθεση νέας λίστας από τις δύο λίστες
Εκτύπωση της νέας λίστας

Για να τρέξετε το πρόγραμμα:

1) στο terminal:

```
python3 test1.py
```

ή 2) σε περιβάλλον python

```
exec(open("test1.py").read())
```

```
['abc', 234, 1.42, 'mary', 45.0]
abc
[234, 1.42]
[1.42, 'mary', 45.0]
[123, 'john', 123, 'john']
['abc', 234, 1.42, 'mary', 45.0, 123, 'john']
```

```
#!/usr/bin/python3

list = ['physics', 'chemistry', 1921, 2021]

print(list[-1])        #Πρόσβαση σε στοιχείο της λίστας από δεξιά προς τα αριστερά

print(len(list))       #Πλήθος στοιχείων λίστας
```

```
python3 test1.py
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()**

test1.py

```
#!/usr/bin/python3

list = ['physics', 'chemistry', 1921, 2021]

print("H timi tis listas sti thesi 2 einai: ")
print(list[1])

list[1] = 1812          #Ανάθεση νέας τιμής σε στοιχείο της λίστας
print("H nea timi tis listas sti thesi 2 einai: ")
print(list[1])


list.append(1812)       #Πρόσθεση στο τέλος της λίστας ενός νέου στοιχείου με τιμή 1812
print(len(list))        #Έλεγχος του πλήθους των στοιχείων της λίστας
print(list[len(list)-1]) #Εκτύπωση του τελευταίου στοιχείου της λίστας
```

Η θέση είναι το μήκος της λίστας – 1 γιατί η αρίθμηση των στοιχείων ξεκινά από το 0.

```
python3 test1.py
H timi tis listas sti thesi 2 einai:
chemistry
H nea timi tis listas sti thesi 2 einai:
1812
5
1812
```

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()** όταν δεν ξέρουμε το στοιχείο

```
list = ['physics', 'chemistry', 1921, 2021]
print("H timi tis listas sti thesi 2 einai:")
print(list[1])
del list[1]  #Διαγραφή στοιχείου της λίστας
print("H lista einai twra")
print(list)
```

```
H timi tis listas sti thesi 2 einai:
chemistry
H lista einai twra
['physics', 1921, 2021]
bash-3.2$
```

List type objects – Συναρτήσεις λίστας

Υπάρχουν συναρτήσεις 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
comp(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 ή η list να μεγαλώσει τα tuples **δεν** μπορούν να τροποποιηθούν.

Επομένως μπορούν να χρησιμοποιηθούν μόνο για ανάγνωση των στοιχείων τους αλλά 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

```
#!/usr/bin/python3

mytuple = ('abcd', 786, 2.32, 'john', 35.)
atuple = (123, 'john')

print(mytuple)                  #Τυπώνει το tuple
print(mytuple[0])               #Τυπώνει το 1ο στοιχείο του tuple
print(mytuple[1:3])             #Τυπώνει από το 2ο έως και το 3ο στοιχείο του tuple
print(mytuple[2:])              #Τυπώνει ξεκινώντας από το 3ο στοιχείο του tuple
print(atuple*2)                 #Τυπώνει 2φορές του tuple atuple
print(mytuple+atuple)           #Σύνθεση νέου typle από τα δύο tuples
newtuple = mytuple+atuple       #Εκτύπωση του νέου tuple
print(newtuple)
```

```
('abcd', 786, 2.32, 'john', 35.0)
abcd
(786, 2.32)
(2.32, 'john', 35.0)
(123, 'john', 123, 'john')
('abcd', 786, 2.32, 'john', 35.0, 123, 'john')
('abcd', 786, 2.32, 'john', 35.0, 123, 'john')
```

```
#!/usr/bin/python3

mytuple = ('abcd', 786, 2.32, 'john', 35.)
list = ['abcd', 786, 2.32, 'john', 35.]

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

list[2] = 1000        #Σωστή χρήση τύπου list
```