Ελληνικό Μεσογειακό Πανεπιστήμιο
Σχολή Μηχανικών / Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

HMY01K06
Επιστημονικός Προγραμματισμός με Python

Διάλεξη Πέμπτη
Αντικείμενα και Ονόματα
Δομημένοι τύποι δεδομένων
(Λίστες / Πλειάδες)

Φθινόπωρο 2025

Αντικείμενα και Κυριολεκτήματα

Objects and Literals

Διαδικαστικός προγραμματισμός (C++)	Αντικειμενοστρεφής προγραμματισμός (Python)
Έμφαση στην έννοια της συνάρτησης	*Έμφαση στην έννοια του αντικειμένου*

- Αντικείμενα (objects)

  Συλλογές δεδομένων με συγκεκριμένες *ιδιότητες* και *μεθόδους χειρισμού*

  Τα αντικείμενα πολλές φορές ονομάζονται και *τύποι δεδομένων*

  Παραδείγματα συλλογών δεδομένων: ακέραιοι, δεκαδικοί, λογικοί, συμβολοσειρές, λίστες, σύνολα…

- Κυριολεκτήματα (literals): *συγκεκριμένοι εκπρόσωποι* αντικειμένων

  Ποσότητες ή σύμβολα που αποτελούν συγκεκριμένους εκπροσώπους μιας συλλογής δεδομένων *η τιμή τους παραμένει ίδια (σταθερή) παντού μέσα στο πρόγραμμα*

  *Παραδείγματα κυριολεκτημάτων: Ο ακέραιος αριθμός **3**, η συμβολοσειρά "Hello", η λογική τιμή **True***

Εκπρόσωπος της συλλογής αντικειμένων `int`

Εκπρόσωπος της συλλογής αντικειμένων `str`

Εκπρόσωπος της συλλογής αντικειμένων `bool`

Αντικείμενα στην Python (1/2)

Όλα τα αντικείμενα στην Python έχουν

συνήθως η διεύθυνση μνήμης που είναι αποθηκευμένο το αντικείμενο

- μια ***μοναδική* ταυτότητα** (έναν *ακέραιο* που επιστρέφεται με το `id(x)`)
- έναν ***μοναδικό* τύπο** (που επιστρέφεται με το `type(x)`)
- κάποιο **περιεχόμενο**

- *Δεν μπορούμε* να αλλάξουμε την *ταυτότητα* ενός αντικειμένου
- *Δεν μπορούμε* να αλλάξουμε τον *τύπο* ενός αντικειμένου
- Κάποια αντικείμενα *μας επιτρέπουν* να αλλάξουμε το *περιεχόμενό τους*
  (χωρίς να αλλάξουμε την ταυτότητα ή τον τύπο τους εννοείται)
- Κάποια αντικείμενα *δεν μας επιτρέπουν* να αλλάξουμε το *περιεχόμενό τους*
  Τα απλά (όχι δομές) αντικείμενα αυτής της μορφής ονομάζονται κυριολεκτήματα (literals)

Ακέραιο
Κυριολέκτημα 10

10

```
>>> id(10)
1882088768
>>> type(10)
<class 'int'>
```

[]

```
>>> id([])
4813328
>>> type([])
<class 'list'>
```

Αντικείμενο
Κενή Λίστα

Αντικείμενα στην Python (2/2)

Τα αντικείμενα μπορεί επίσης να έχουν:

- μηδέν ή περισσότερες *μεθόδους* χειρισμού *(ανάλογα τον τύπο τους)*
- μηδέν ή περισσότερα *ονόματα (προσδιοριστές)*

- Κάποια αντικείμενα έχουν *μεθόδους* που μας επιτρέπουν τόσο *να αποκτήσουμε πρόσβαση (δηλ. να δούμε)* όσο και *να αλλάξουμε* το περιεχόμενό τους *(δηλ. να το τροποποιήσουμε επιτόπου)*
- Κάποια αντικείμενα έχουν μόνο *μεθόδους* που μας επιτρέπουν να αποκτήσουμε *πρόσβαση* στο περιεχόμενό τους, αλλά *όχι να το αλλάξουμε*
- Κάποια αντικείμενα δεν έχουν *καθόλου μεθόδους χειρισμού*

Ανεξάρτητα από το αν ένα αντικείμενο έχει μεθόδους χειρισμού ή όχι , δεν μπορούμε ***ποτέ*** να αλλάξουμε *ούτε τον τύπο*, *ούτε την ταυτότητά* του

Ονόματα και ονοματοχώροι

Με τα ονόματα τα πράγματα είναι λίγο διαφορετικά

- Τα ονόματα *(οι προσδιοριστές)* ***δεν είναι*** *ιδιότητες* αντικειμένων
- Ένα αντικείμενο μπορεί να έχει *οποιοδήποτε* όνομα ή και *κανένα* όνομα
- Ένα αντικείμενο *δεν ξέρει τι όνομα έχει (και ούτε νοιάζεται γι' αυτό)*

Τα ονόματα "ζουν" σε *ονοματοχώρους (namespaces)* όπως ο ονοματοχώρος ενός *προγράμματος (module namespace)* ή ένας τοπικός ονοματοχώρος μιας *συνάρτησης* ή μιας *μεθόδου (function/method local namespace)*

Οι ονοματοχώροι είναι *συλλογές ζευγών (όνομα, αναφορά στο αντικείμενο)* και εσωτερικά στην Python, υλοποιούνται με χρήση *λεξικών*

Εκχωρήσεις Ονομάτων (όνομα = αντικείμενο)

Οι *εντολές εκχώρησης* τροποποιούν τους *ονοματοχώρους*, ***όχι τα αντικείμενα***.

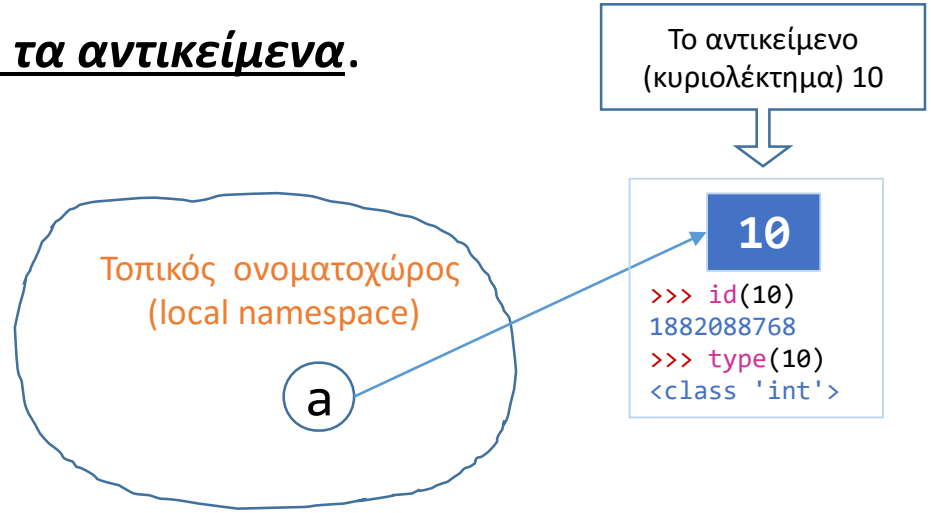
Γράφοντας σε ένα πρόγραμμα

```
a = 10
```

δηλαδή ένας προσδιοριστής

τότε προστίθεται αρχικά ένα *νέο όνομα* a στον *τοπικό ονοματοχώρο* το οποίο αναφέρεται *(συνδέεται)* σε ένα *αντικείμενο,* εν προκειμένω στο ακέραιο κυριολέκτημα `10`

Αν το αντικείμενο δεν υπάρχει, τότε δημιουργείται εκείνη τη στιγμή

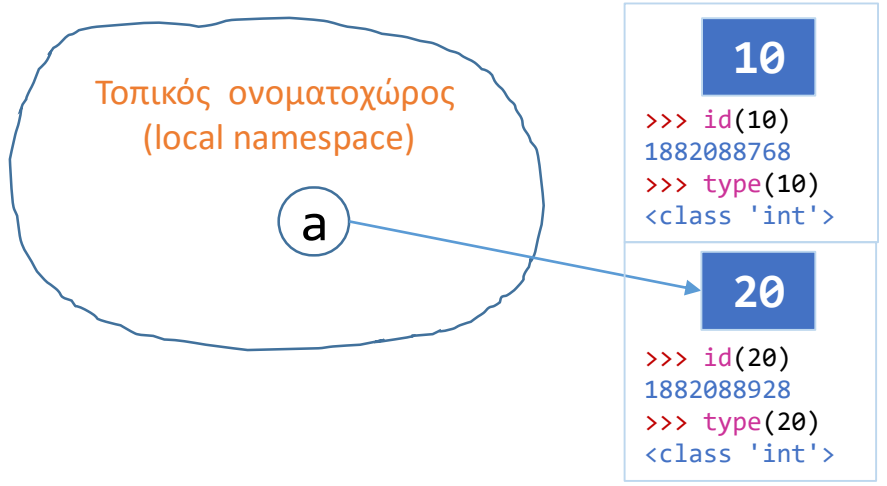


Αν πιο κάτω στο πρόγραμμα γράψουμε

```
a = 20
```

τότε *τροποποιούμε το όνομα* a κάνοντάς το να αναφέρεται σε ένα *άλλο αντικείμενο*, το ακέραιο κυριολέκτημα `20`

Αν το αντικείμενο δεν υπάρχει, τότε δημιουργείται εκείνη τη στιγμή



Το αρχικό αντικείμενο `10` δεν επηρεάζεται από αυτή την πράξη ούτε και νοιάζεται αν έπαψε να εκπροσωπείται στο πρόγραμμα

Εκχωρήσεις Ονομάτων (όνομα = αντικείμενο)

Αν στη συνέχεια γράψουμε

```
b = a
```

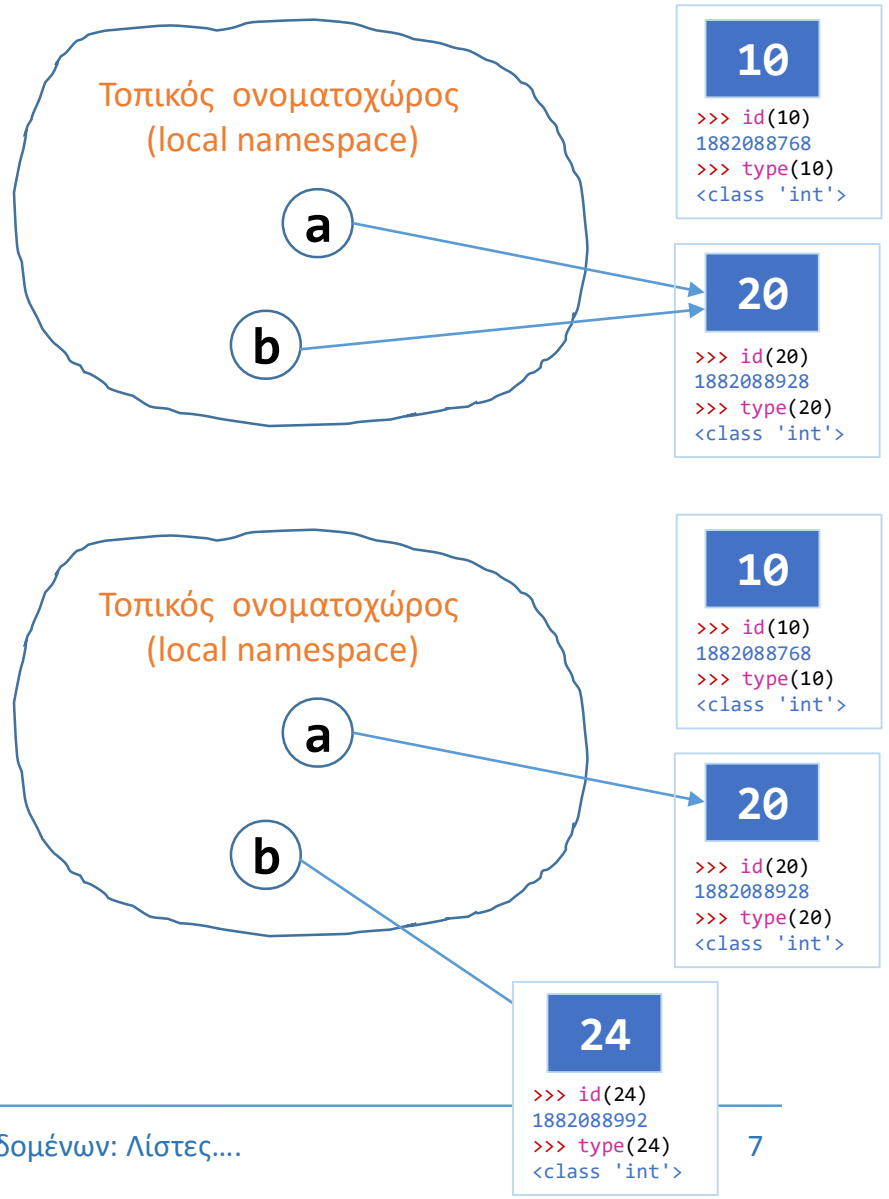
τότε *προστίθεται ένα ακόμη όνομα* b στον τοπικό ονοματοχώρο το οποίο *αναφέρεται στο ίδιο αντικείμενο* που αναφέρεται και το a , δηλαδή στο ακέραιο κυριολέκτημα 20
(τα `id(a)` και `id(b)` επιστρέφουν τον ίδιο αριθμό `1882088928`)

Το ακέραιο κυριολέκτημα 20 *δεν επηρεάζεται* από αυτή τη πράξη *ούτε νοιάζεται αν ένα ακόμα όνομα αναφέρεται πάνω του*

Τέλος, αν γράψουμε

```
b = b + 4
```

i. αρχικά γίνεται η πράξη b + 4 η οποία έχει σαν αποτέλεσμα τη δημιουργία ενός νέου αντικειμένου (ακέραιο κυριολέκτημα 24)
ii. στη συνέχεια τροποποιείται το όνομα b ώστε να αναφέρεται σε αυτό το νέο αντικείμενο
(το `id(b)` επιστρέφει τώρα τον αριθμό `1882088992`)



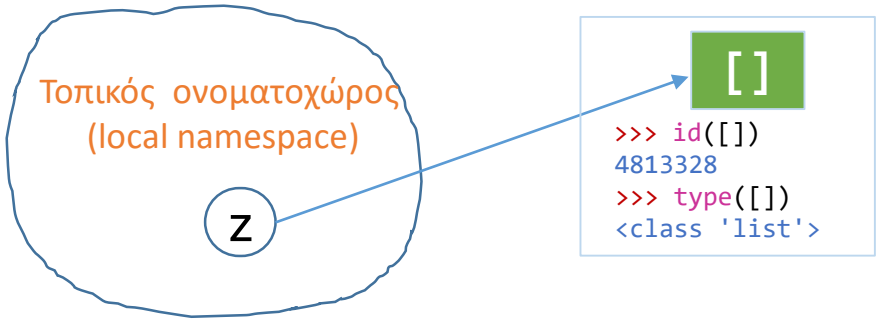
Εκχωρήσεις Ονομάτων (όνομα = αντικείμενο)

Αντικείμενα που μας επιτρέπουν να αλλάξουμε το περιεχόμενό τους *(mutables)*

Δίνοντας την εντολή

```
z = []    # δημιουργία κενής λίστας
```

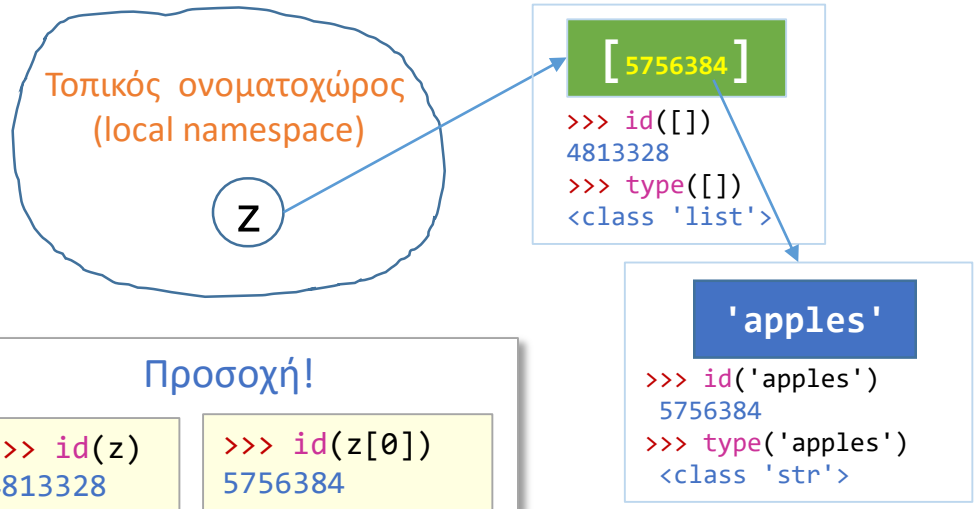
προσθέτουμε ένα *νέο όνομα* z στον *τοπικό ονοματοχώρο* το οποίο αναφέρεται σε ένα νέο *αντικείμενο τύπου list* που δημιουργείται αρχικά κενό περιεχομένου
(το `id(z)` επιστρέφει την τιμή `4813328`)



Αν στη συνέχεια δώσουμε την εντολή

```
z.append('apples')  # προσθήκη στο τέλος
```

τότε *εφαρμόζουμε μια μέθοδο πάνω στο αντικείμενο* στο οποίο αναφέρεται το z *(δηλαδή στην κενή λίστα)* που την αναγκάζει να τροποποιήσει τον περιεχόμενό της



Αυτό αλλάζει *το περιεχόμενο του αντικειμένου*, αλλά *δεν επηρεάζει τον τοπικό ονοματοχώρο* (το z εξακολουθεί να δείχνει στο ίδιο αντικείμενο, το `id(z)` εξακολουθεί να επιστρέφει την τιμή `4813328`)

Προσοχή!

```
>>> id(z)
4813328
```

```
>>> id(z[0])
5756384
```

Εκχωρήσεις Ονομάτων (όνομα = αντικείμενο)

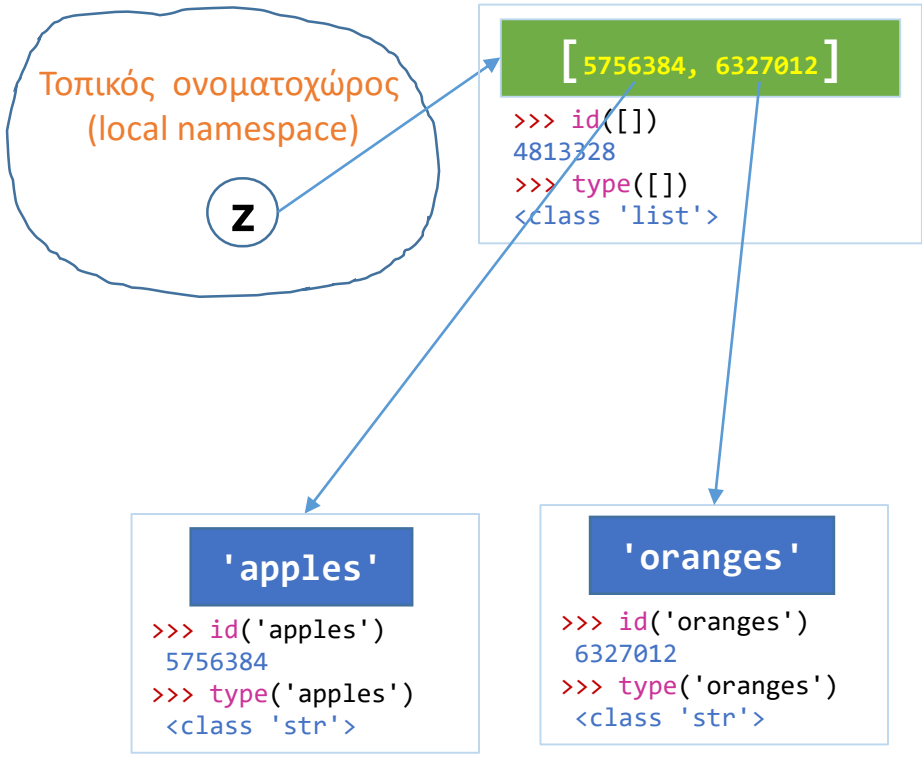
Αντικείμενα που μας επιτρέπουν να αλλάξουμε το περιεχόμενό τους *(mutables)*

Αν στη συνέχεια προσθέσουμε και δεύτερο στοιχείο στη λίστα

```
z.append('oranges')  # προσθήκη στο τέλος
```

τότε *εφαρμόζουμε ξανά μια μέθοδο πάνω στο αντικείμενο* στο οποίο αναφέρεται το z *(δηλαδή στην λίστα που περιέχει ένα στοιχείο)* αναγκάζοντάς την να τροποποιήσει το περιεχόμενό της

Αυτό αλλάζει *το περιεχόμενο του αντικειμένου*, αλλά *δεν επηρεάζει τον τοπικό ονοματοχώρο* (το z εξακολουθεί να δείχνει πάντα στο ίδιο αντικείμενο - το `id(z)` επιστρέφει πάντα 4813328)




```
>>> id(z)          >>> type(z)
4813328            <class 'list'>
>>> id(z[0])       >>> type(z[0])
5756384            <class 'str'>
>>> id(z[1])       >>> type(z[1])
6327012            <class 'str'>
```

Δομημένα Αντικείμενα (τύποι δεδομένων) της Python

Επιπρόσθετα από τους βασικούς τύπους δεδομένων που έχουμε δει μέχρι τώρα *(integer, float, logical, complex και string)*, η Python υποστηρίζει και τους ακόλουθους δομημένους τύπους δεδομένων:

- Λίστες *(lists)*

```
L = [ 1, 2, "καλημέρα", 5.8, [1,3] ]
```

- Πλειάδες *(tuples)*

```
T = ( 3.14, "καλησπέρα", 7, 4 )
```

- Σύνολα *(sets)*

```
S = { 'μήλα','αχλάδια', 3, 'μπανάνες' }
```

- Λεξικά *(dictionaries)*

```
D = { "Μάρκα":"VW", "Μοντέλο":"Golf", "Έτος":2025 }
```

Λίστες

- Η κλάση (τύπος) αντικειμένων *list* της Python μας δίνει τη δυνατότητα να δημιουργούμε μια *ακολουθία ετερογενών -εν γένει- δεδομένων κάτω από το ίδιο όνομα*

- Για να δημιουργήσουμε μια λίστα αντικειμένων στην Python, απλά αναγράφουμε όλα τα αντικείμενα μέσα σε αγκύλες [ ], χωρισμένα με *κόμμα*

```
>>> colors=['red','green','blue']
```

- Μια λίστα μπορεί να περιέχει ανάμικτα αντικείμενα οποιουδήποτε τύπου

```
>>> L=['Monday',"Tuesday", 24,'Hello', 5, 7.3, True]
>>> L
['Monday', 'Tuesday', 24, 'Hello', 5, 7.3, True]
```

Σε αντίθεση με τις περισσότερες γλώσσες προγραμματισμού *(πχ C++ ή Java)* η Python στην βασική της μορφή *δεν υποστηρίζει arrays* (πίνακες) καθώς οι λίστες αποτελούν μια γενικότερη μορφή τους *(το περιεχόμενο ενός array πρέπει να είναι ενιαίου τύπου)*

Υπάρχουν εντούτοις επιπρόσθετα πακέτα *(όπως πχ. το πακέτο NumPy)* που προσφέρουν υποστήριξη για arrays, προσφέροντας και σειρά ειδικών μεθόδων για αυτά

Λίστες

- Μια λίστα μπορεί να περιέχει μέσα άλλους δομημένους τύπους δεδομένων *(λίστες, πλειάδες κλπ.)*

```
>>> languages = [('English','French'),'German','Greek','Chinese','Spanish']
>>> languages
[('English', 'French'), 'German', 'Greek', 'Chinese', 'Spanish']
```

<-- Πλειάδα (tuple) δυο στοιχείων -->

```
>>> lang = [['English'],['French'],['German'], 'Greek', 'Chinese','Spanish']
>>> lang
[['English'], ['French'], ['German'], 'Greek', 'Chinese', 'Spanish']
```

<---------- 3 Λίστες με 1 στοιχείο η καθεμιά ---------->

- Για να *προσπελάσουμε* ένα στοιχείο μιας λίστας, βάζουμε τον *δείκτη* του σε *αγκύλη* δίπλα στο όνομα της λίστας. Η αρίθμηση των δεικτών *ξεκινάει από το μηδέν* *(όπως και στις συμβολοσειρές)*

```
>>> lang[4]
'Chinese'
```

Το τέταρτο στοιχείο της λίστας lang ***είναι μια συμβολοσειρά***

```
>>> lang[0]
['English']
```

Το πρώτο στοιχείο της λίστας lang ***είναι μια λίστα ενός στοιχείου*** *που περιέχει μια* ***συμβολοσειρά***

Τμηματική πρόσβαση / μεταβολή στοιχείων λίστας

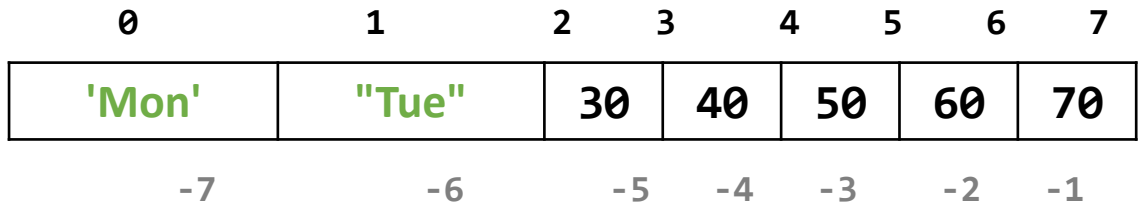
```
>>> L = ['Mon',"Tue",30,40,50,60,70]
>>> L
['Mon','Tue',30,40,50,60,70]

>>> len(L)
7
```

Πρόσβαση ενός στοιχείου της λίστας

```
>>> L[3]
40
```

Μεταβολή ενός στοιχείου λίστας

```
>>> L[2] = "Wed"
>>> L
['Mon','Tue','Wed',40,50,60,70]
```



Πρόσβαση τμήματος λίστας (όπως ακριβώς και στις συμβολοσειρές)

```
>>> L[1:3]
['Tue', 30]
>>> L[-3:-1]
[50, 60]
>>> L[-6:]
['Tue', 30, 40, 50, 60, 70]
```

Μεταβολή τμήματος λίστας

```
>>> L[1:3]= [777, 888, 999]
>>> L
['Mon', 777, 888, 999, 40, 50, 60, 70]
```

Το νέο τμήμα που δεν χρειάζεται να είναι απαραίτητα ίδιου μεγέθους με εκείνο που αντικαθιστούμε

Αντιστροφή λίστας (όπως ακριβώς και στις συμβολοσειρές)

```
>>> L[::-1]
```
επιστρέφει την λίστα με τα στοιχεία της αντεστραμμένα

Πολυδιάστατες Λίστες

a[0]			a[1]			a[2]			
a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]	a[2][0]	a[2][1]	a[2][2]	a[2][3]
1	2	3	34	35	36	'A'	'B'	'C'	'D'

```
>>> a = [[1,2,3], [34,35,36], ['A','B','C','D']]
>>> a[2][1]
'B'
>>> a[1][2] = 99
>>> a
[[1,2,3],[34,35,99],['A','B','C','D']]
>>> a[1][1] = [100,200]
>>> a
[[1,2,3],[34,[100, 200],99],['A','B','C','D']]
>>> a[1][1][0]
100
```

Προσέξτε και κατανοήστε τις διαφορές!

```
>>> a[1]
[34, 35, 36]
>>> a[:1]
[[1, 2, 3]]
```

```
>>> a[2][2]
'C'
>>> a[2][:2]
['A', 'B']
```

Συναρτήσεις και μέθοδοι χειρισμού λιστών

`len()`	Επιστρέφει τον αριθμό των στοιχείων της λίστας (μήκος λίστας) πχ αν `L=[1,[5,'test'],8,[25,7,(36,4),39]]` τότε το `len(L)` επιστρέφει `4`
`min()/ max()`	Επιστρέφει το μικρότερο / μεγαλύτερο στοιχείο της λίστας *(όταν αυτό έχει νόημα)* πχ αν `L=[1,8,2]` τότε το `max(L)` επιστρέφει `8` αν `L=['a','p','e']` τότε το `max(L)` επιστρέφει `'p'` αν `L=[1,[5,'test'],8,[25,2]]` τότε το `max(L)` επιστρέφει *σφάλμα*
`in / not in`	Ελέγχει αν ένα στοιχείο ανήκει / δεν ανήκει σε μια λίστα πχ `3 in [1,5,2]` επιστρέφει `False`
`+`	Συνενώνει δυο λίστες πχ `[2,'test'] + [1,5,4]` επιστρέφει `[2,'test',1,5,4]`
`*`	"Πολλαπλασιάζει" μια λίστα με έναν ακέραιο αριθμό πχ `[2,'test'] * 3` επιστρέφει `[2,'test', 2,'test', 2,'test']`

Όπως ακριβώς και στις συμβολοσειρές

Συναρτήσεις και μέθοδοι χειρισμού λιστών

`.append(z)`	Προσθέτει το *στοιχείο* `z` *στο τέλος της λίστας* πχ αν `L=[1,2,5]` τότε με `L.append(4)` η `L` γίνεται `[1,2,5,4]` αν `L=[1,2,5]` τότε με `L.append([2,7])` η `L` γίνεται `[1,2,5,[2,7]]`
`.extend(L1)`	Προσθέτει τα στοιχεία της *λίστας* `L1` *στο τέλος της λίστας* *(ισοδύναμη με τον τελεστή +)* *Εκτός από λίστα, το L1 μπορεί να είναι γενικά οποιασδήποτε επαναληπτικός τύπος* πχ αν `L=[1,2,5]` τότε με `L.extend([2,7])` η `L` γίνεται `[1,2,5,2,7]` ενώ `L.extend(2,7)` επιστρέφει *σφάλμα* *(γιατί;)*
`.insert(i,z)`	Προσθέτει το *στοιχείο* `z` στην θέση `i` της λίστας *(*)* πχ αν `L=[1,3,5,8]` τότε με `L.insert(2,999)` η `L` γίνεται `[1,3,999,5,8]` *αν το `i` είναι ίσο με το μήκος της λίστας, τότε η εντολή είναι ισοδύναμη με την append*
`.remove(z)`	Αφαιρεί το *στοιχείο* `z` *την πρώτη φορά* που το συναντάει στην λίστα πχ αν `L=[1,2,5,2,7]` τότε με `L.remove(2)` η `L` γίνεται `[1,5,2,7]`

(*) Αντί για ένα μόνο στοιχείο, πως μπορούμε άραγε να προσθέσουμε μια ολόκληρη λίστα μέσα σε μια άλλη;

Συναρτήσεις και μέθοδοι χειρισμού λιστών

`.index(z)` `.index(z,i,j)`	Επιστρέφει τον δείκτη *(θέση)* του στοιχείου `z` την πρώτη φορά που το συναντάει στη λίστα. Αν δίνονται επιπλέον οι δείκτες `i` και `j` τότε ο έλεγχος γίνεται μόνο στην περιοχή από `i` μέχρι `j-1`. πχ αν `L=['a','b','e','b','c','a']` τότε το `L.index('b')` επιστρέφει `1` ενώ το `L.index('b',2,4)` επιστρέφει `3`
`.count(z)`	Επιστρέφει το πόσες φορές συναντάμε το στοιχείο `z` μέσα στη λίστα. πχ αν `L=['a','b','e','b','c','a']` τότε το `L.count('b')` επιστρέφει `2` ενώ το `L.count('w')` επιστρέφει `0`
`list()`	Δημιουργεί μια καινούρια λίστα από έναν *απαριθμήσιμο* τύπο πχ δίνοντας `list(range(5))` δημιουργείται η λίστα `[0, 1, 2, 3, 4]` δίνοντας `list((11,22,33))` δημιουργείται η λίστα `[11, 22, 33]` δίνοντας `list('Hello')` δημιουργείται η λίστα `['H', 'e', 'l', 'l', 'o']` όμως προσοχή: `list(11,22,33)` επιστρέφει *σφάλμα* *(γιατί;)*

Συναρτήσεις και μέθοδοι χειρισμού λιστών

`del L`	Διαγράφει ολόκληρη τη λίστα *Μετά από αυτό, το όνομα L είναι πλέον άγνωστο στην Python (διαγράφεται από τον χώρο ονομάτων)*
`del L[i]` `del L[i:j]`	Διαγράφει το στοιχείο `i` (αντίστοιχα τα στοιχεία από `i` έως `j-1`) της λίστας πχ αν `L=[11,22,33,44,55,66]` τότε με `del L[2]` η L γίνεται `[11,22,44,55,66]` αν `L=[11,22,33,44,55,66]` τότε με `del L[1:3]` η L γίνεται `[11,44,55,66]`
`.clear()`	Καθαρίζει την λίστα, διαγράφοντας όλα τα στοιχεία της. πχ αν `L=[1,2,'test',[8,4]]` τότε με `L.clear()` η L γίνεται `[]`
`.pop(i)`	Διαγράφει και επιστρέφει *(στη ουσία εξάγει)* το στοιχείο της λίστας στη θέση `i` Αν παραλείψουμε το `i`, τότε εξάγεται το τελευταίο στοιχείο της λίστας πχ αν `L=[11,22,33,44,55,66]` τότε `L.pop(3)` επιστρέφει `44` και ταυτόχρονα η L γίνεται `[11,22,33,55,66]`
`.copy()`	Δημιουργεί ένα αντίγραφο της αρχικής λίστας πχ `L1 = L.copy()`

Συναρτήσεις και μέθοδοι χειρισμού λιστών

`.sort()` `.sort(reverse = True)`	Ταξινομεί *(όταν αυτό έχει νόημα)* τη λίστα κατά αύξουσα /φθίνουσα σειρά επιτόπου πχ. αν `L=[5, 6, 2, 7, 2, 9, 1, 4, 1]` τότε με `L.sort()` η L γίνεται `[1, 1 ,2, 2, 4, 5, 6, 7, 9]` ενώ με `L.sort(reverse = True)` η L γίνεται `[9, 7, 6, 5, 4, 2, 2, 1, 1]` ενώ πχ. αν `L=[5, 6, 'hello']` τότε η `L.sort()` επιστρέφει *σφάλμα*
`sorted(L)` `sorted(L, reverse = True)`	Δημιουργεί *(όταν αυτό έχει νόημα)* ένα αντίγραφο της λίστας ταξινομημένο κατά αύξουσα /φθίνουσα σειρά πχ με `L1 = sorted(L, reverse= True)` δημιουργείται η αντίστροφα ταξινομημένη λίστα L1 ενώ η L παραμένει ως έχει
`.reverse()`	Αντιστρέφει τη λίστα επιτόπου πχ αν `L=['a', 'b', 'aaaa', 4, [3, 6]]` τότε με `L.reverse()` η L γίνεται `[[3, 6], 4, 'aaaa', 'b', 'a']`

Συνοπτικός πίνακας συναρτήσεων και μεθόδων πάνω στις λίστες

`len()`	Επιστρέφει τον αριθμό των στοιχείων της λίστας (μήκος λίστας)
`min()/ max()`	Επιστρέφει το μικρότερο / μεγαλύτερο στοιχείο της λίστας
`in / not in`	Ελέγχει αν ένα στοιχείο ανήκει / δεν ανήκει σε μια λίστα
`+`	Συνενώνει δυο λίστες
`*`	Πολλαπλασιάζει μια λίστα με έναν ακέραιο αριθμό
`.append(z)`	Προσθέτει το *στοιχείο* `z` στο τέλος της λίστας
`.extend(L1)`	Προσθέτει την *λίστα* `L1` στο τέλος της λίστας *(ισοδύναμη με τον τελεστή +)*
`.insert(i,z)`	Προσθέτει το *στοιχείο* `z` στην θέση `i` της λίστας
`.remove(z)`	Αφαιρεί το *στοιχείο* `z` την πρώτη φορά που το συναντάει στην λίστα
`.index(z)`	Επιστρέφει τον δείκτη του στοιχείου `z` την πρώτη φορά που το συναντάει στη λίστα.
`.count(z)`	Επιστρέφει το πόσες φορές συναντάμε το στοιχείο `z` μέσα στη λίστα.
`list()`	Δημιουργεί μια καινούρια λίστα από έναν απαριθμήσιμο τύπο
`del L`	Διαγράφει μέρος ή/και ολόκληρη τη λίστα
`.clear()`	Καθαρίζει την λίστα, διαγράφοντας όλα τα στοιχεία της.
`.pop(i)`	Διαγράφει και επιστρέφει (στη ουσία εξάγει) το στοιχείο της λίστας στη θέση `i`
`.copy()`	Δημιουργεί ένα αντίγραφο της αρχικής λίστας
`.sort()`	Ταξινομεί *επιτόπου* τη λίστα κατά αύξουσα /φθίνουσα σειρά *(όταν αυτό έχει νόημα)*
`sorted(L)`	Δημιουργεί ένα ταξινομημένο *αντίγραφο* της λίστας *(όταν αυτό έχει νόημα)*
`.reverse()`	Αντιστρέφει *επιτόπου* τη λίστα

Εφαρμογές και παραδείγματα πάνω στις λίστες *(1/10)*

Μήκος λίστας (πλήθος στοιχείων που περιέχει)

```
L = [1, [2,3,4], (5, "six"), 7, 8]
print(len(L))
```

5 (περιέχει τρεις αριθμούς, μια άλλη λίστα, και μια πλειάδα)

```
print(len(L[1]))
```

3 *(το στοιχείο της λίστας με δείκτη 1 είναι μια λίστα και έχει μήκος 3)*

```
print(len(L[0]))
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    print(len(L[0]))
TypeError: object of type 'int' has no len()
```

Εφαρμογές και παραδείγματα πάνω στις λίστες *(2/10)*

Απαρίθμηση όλων των στοιχείων μιας λίστας

```
L1= ["apple", "banana", "cherry"]
for x in L1:
  print(x, end = " ")
```

```
apple banana cherry
```

```
L2= [1, [2,3,4], (5, "six"), 7, 8]
for x in L2:
    print(x, end= ' ')
```

```
1 [2, 3, 4] (5, 'six') 7 8
```

Έλεγχος αν ένα στοιχείο υπάρχει ή όχι στη λίστα

```
L1= ["apple", "banana", "cherry"]
if "apple"  in L1:
  print("Yes, apple exists in this list")
```

```
Yes, apple exists in this list
```

Εφαρμογές και παραδείγματα πάνω στις λίστες *(3/10)*

Προσθήκη στοιχείων λίστας (μέθοδοι append και insert)

```
L = ["apple", "banana", "cherry"]
L.append("orange")
print(L)
['apple', 'banana', 'cherry', 'orange']

L = ["apple", "banana", "cherry"]
L.insert(1, "orange")
print(L)
['apple', 'orange', 'banana', 'cherry']

L = ["apple", "banana", "cherry"]
L.insert(len(L), "pineapple")   <- ισοδύναμο με το append
print(L)
['apple', 'banana', 'cherry', 'pineapple']
```

Εφαρμογές και παραδείγματα πάνω στις λίστες *(4/10)*

Διαγραφή στοιχείων λίστας (μέθοδος remove)

```
L = ["apple", "banana", "cherry", "banana"]
L.remove("banana")
print(L)
['apple', 'cherry', 'banana'] (Διαγράφει το πρώτο που συναντάει)

L = ["apple", "banana", "cherry", "banana"]
L.remove("carrot")
Traceback (most recent call last):
  File "<pyshell#55>", line 1, in <module>
    L.remove("carrot")
ValueError: list.remove(x): x not in list
```

Εφαρμογές και παραδείγματα πάνω στις λίστες *(5/10)*

Εξαγωγή ενός στοιχείου από μια λίστα (μέθοδος pop)

```
L = ["apple", "banana", "cherry"]
x= L.pop()
print(x, L)
'cherry' ['apple', 'banana']

L = ["apple", "banana", "cherry"]
x= L.pop(1)
print(x, L)
'banana' ['apple', 'cherry']
```

Εφαρμογές και παραδείγματα πάνω στις λίστες *(6/10)*

Διαγραφή στοιχείων λίστας (λέξη κλειδί del)

```
L = ["apple", "banana", "cherry"]
del L[0]
print(L)
['banana', 'cherry']
```

Διαγραφή ολόκληρης της λίστας (λέξη κλειδί del)

```
L = ["apple", "banana", "cherry"]
del L
print(L)
Traceback (most recent call last):
  File "C:/Users/PYTHON Programs/foo.py", line 3, in <module>
    print( L)
NameError: name 'L' is not defined
```

Το del αποτελεί κλασσική περίπτωση υπερφόρτωσης τελεστή, καθώς διαγράφει είτε ολόκληρα αντικείμενα είτε ένα τμήμα τους ανάλογα με το δοσμένο όρισμα

Εφαρμογές και παραδείγματα πάνω στις λίστες *(7/10)*

Δημιουργία λίστας από επαναλήψιμα αντικείμενα με την συνάρτηση `list()`

```
L1 = list("abc123")  μετατρέπει κάθε χαρακτήρα σε ξεχωριστό στοιχείο λίστας
print(L1)
['a', 'b', 'c', '1', '2', '3']

L2 = list((3,11,2)) πλειάδα → λίστα
print(L2)
[3, 11, 2]

L3 = list(range(3,11,2))
print(L3)
[3, 5, 7, 9]

L4 = list(3,11,2)
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: list expected at most 1 argument, got 3
```

Επαναλήψιμο (iterable) είναι κάθε αντικείμενο στην Python που μπορεί να επιστρέψει τα μέλη του ένα κάθε φορά πχ. μέσω ενός for loop *(υπάρχουν και άλλοι τρόποι να γίνεται αυτό)*

Παραδείγματα: κάθε συμβολοσειρά, λίστα, πλειάδα, η συνάρτηση `range` κ.α.

Εφαρμογές και παραδείγματα πάνω στις λίστες *(8/10)*

Συνένωση δυο Λιστών

Μπορεί να γίνει με διάφορους τρόπους

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
['a', 'b', 'c', 1, 2, 3]
```

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
for x in list2
    list1.append(x)
print(list1)
['a', 'b', 'c', 1, 2, 3]
```

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list1.extend(list2)
print(list1)
['a', 'b', 'c', 1, 2, 3]
```

Εφαρμογές και παραδείγματα πάνω στις λίστες *(9/10)*

Μέτρηση του πλήθους εμφανίσεων ενός στοιχείου σε μια λίστα

Η μέθοδος count() επιστρέφει το πόσες φορές εμφανίζεται ένα συγκεκριμένο στοιχείο σε μια λίστα

```
>>> L = [1, 3, 2, "test", 3, 7, 4]
>>> print(L.count(3), L.count("test"), L.count("Test") )
 2 1 0
```

Εύρεση της πρώτης θέσης που εμφανίζεται ενα στοιχείο σε μια λίστα

Η μέθοδος index() επιστρέφει την θέση που εμφανίζεται για πρώτη φορά ένα στοιχείο σε μια λίστα. Αν το στοιχείο δεν υπάρχει, τότε επιστρέφεται μήνυμα λάθους

```
>>> L = [1, 3, 2, "test", 3, 7, 4]
>>> print(L.index(3), L.index("test"))
 1 3
 >>> print(L.index("Test"))
 ValueError: 'Test' is not in list
```

Εφαρμογές και παραδείγματα πάνω στις λίστες *(10/10)*

Αντιγραφή λίστας

Για να αντιγράψουμε μια λίστα χρησιμοποιήσουμε τη μέθοδο .copy()

```
>>> list1 = ["apple", "banana", "cherry"]
>>> list2 = list1.copy()
>>> print(list2)
['apple', 'banana', 'cherry']

>>> list1 = ["apple", "banana", "cherry"]
>>> list2 = list1
>>> print(list2)
['apple','banana','cherry']
>>> list1[1] = "blueberry"
>>> print(list1, '\n', list2)
['apple','blueberry','cherry']
['apple','blueberry','cherry']
```

Προσοχή:

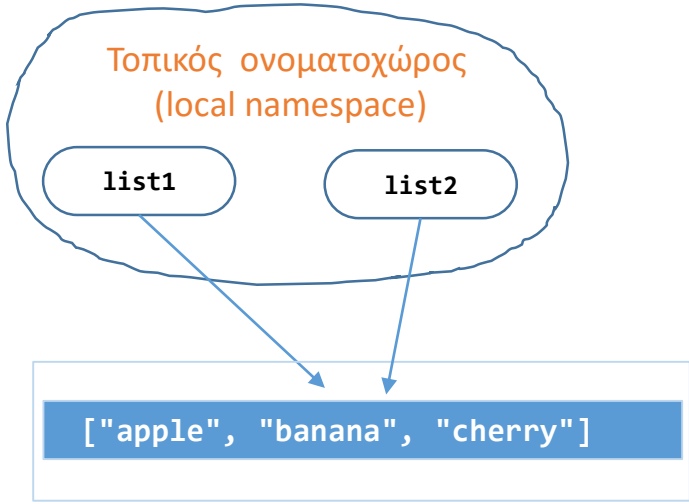
Είναι λάθος να δώσουμε την εντολή

```
list2 = list1
```

για αντιγραφή της λίστας *(γιατί;)*

Διότι έτσι απλά *δημιουργούμε ένα ακόμα όνομα* στον ονοματοχώρο που αναφέρεται στο ίδιο αντικείμενο

(η ίδια λίστα αποκτά δυο ονόματα)



Πλειάδες (Tuples)

Αμετάβλητες διατεταγμένες συλλογές αντικειμένων (immutable ordered set of objects)

Τα αντικείμενα στις πλειάδες γράφονται μέσα σε *παρενθέσεις* και χωρίζονται με κόμμα

```
mytuple = ("apple",  "banana",  "cherry", "orange", "kiwi", "melon")

print(mytuple[2])       ⇨ 'cherry'

print(mytuple[2:5])     ⇨ ('cherry', 'orange', 'kiwi')

print(mytuple[-4:-1])   ⇨ ('cherry', 'orange', 'kiwi')

mytuple[2] = "mango"
TypeError: 'tuple' object does not support item assignment
```

Το περιεχόμενο των πλειάδων δεν επιτρέπεται να αλλάξει (*immutable objects*)

Το περιεχόμενο των πλειάδων δεν επιτρέπεται να αλλάξει

(Όμως αν χρειαστεί, υπάρχει έμμεσος τρόπος αλλαγής)

```
T = ("apple", 'banana', "cherry", "orange", "kiwi", 'melon')
L = list(T)                 # Μετατροπή πλειάδας σε λίστα
L[2] = 'mango'              # Αλλαγή περιεχομένου στοιχείου λίστας
T = tuple(L)                # Μετατροπή λίστας πίσω σε πλειάδα
print(T)
('apple', 'banana', 'mango', 'orange', 'kiwi', 'melon')
```

Συναρτήσεις και μέθοδοι πάνω στις πλειάδες

Ισχύουν εν γένει οι ίδιες που μάθαμε για τις λίστες,

εκτός από εκείνες που επηρεάζουν το περιεχόμενό τους

(ως γνωστόν οι λίστες είναι *mutables* ενώ οι πλειάδες είναι *immutables*)

Πχ οι επόμενες συναρτήσεις και μέθοδοι που δεν αλλοιώνουν το περιεχόμενο των αντικειμένων πάνω στα οποία επιδρούν, *ισχύουν και για τις πλειάδες*

- `len(),min(),max(), +, *`
- `in, not in, del` *(το del μόνο για διαγραφή ολόκληρης της πλειάδας)*
- `.index(), .count()`

ενώ οι παρακάτω μέθοδοι *δεν ισχύουν για τις πλειάδες*

- `.append(), .extend(), .insert(), .remove()`
- `.clear(), .pop(), .copy()`
- `.sort(), .reverse()`

Εφαρμογές και παραδείγματα στις πλειάδες *(1/4)*

Δημιουργία πλειάδων με αναγραφή των στοιχείων τους

```
t1 = ("a", "b" , "c")
t2 = (1, 2, "Hello")
```

```
t1 = "a","b","c"
t2 = 1,2,"Hello"
```

γίνεται εναλλακτικά και χωρίς παρένθεση

Δημιουργία πλειάδων με την συνάρτηση `tuple()`

```
t1 = tuple(range(1,8,2))
```

δημιουργεί την πλειάδα `(1, 3, 5, 7)`

```
t1 = tuple('Hello')
```

δημιουργεί την πλειάδα `('H', 'e', 'l', 'l', 'o')`

Συνένωση πλειάδων

```
t1 = ("a", "b" , "c")
t2 = (1, 2, 3)
t3 = t1 + t2
print(t3)
('a', 'b', 'c', 1, 2, 3)
```

Δημιουργία πλειάδας με ένα μόνο στοιχείο

Προσθέτουμε ***υποχρεωτικά*** ένα κόμμα στο τέλος

```
x = ("apple",)
print(type(x))
<class 'tuple'>
```

το κόμμα είναι απαραίτητο

Τι θα συμβεί αν δεν βάλουμε κόμμα;

```
x = ("apple")
print(type(x))
<class 'str'>
```

```
y = (1)
print(type(y))
<class 'Int'>
```

Η Python θα νομίζει ότι ορίσαμε κάτι άλλο (στο παράδειγμά μας, μια συμβολοσειρά ή έναν ακέραιο)

Εφαρμογές και παραδείγματα στις πλειάδες *(2/4)*

Μήκος πλειάδας (πλήθος στοιχείων που περιέχει)

```
thistuple = ("apple",  "banana",  "cherry")
print(len(thistuple))
```

```
3
```

Απαρίθμηση όλων των στοιχείων μιας πλειάδας

```
thistuple = ("apple", "banana", "cherry")
for x in  thistuple:
  print(x, end = " ")
```

```
apple banana cherry
```

Έλεγχος αν ένα στοιχείο υπάρχει ή όχι στην πλειάδα

```
thistuple = ("apple", "banana", "cherry")
if "apple"  in  thistuple:
  print("Yes, apple exists in this tuple")
```

```
Yes, apple exists in this list
```

Όπως ακριβώς και στις λίστες

Εφαρμογές και παραδείγματα στις πλειάδες *(3/4)*

Διαγραφή μεμονωμένων στοιχείων πλειάδας

Δεν υποστηρίζεται, καθώς οι πλειάδες είναι *αμετάβλητα* αντικείμενα

Αν *οπωσδήποτε* πρέπει να διαγράψουμε μεμονωμένα στοιχεία, τότε

1. μετατρέπουμε την *πλειάδα* σε *λίστα*,
2. *διαγράφουμε* τα στοιχεία από τη *λίστα* και
3. μετατρέπουμε πίσω την *λίστα* σε *πλειάδα*

```
>>> t = (1,2,3,4)
>>> L = list(t)
>>> L
[1, 2, 3, 4]
>>> del L[1]
>>> t = tuple(L)
>>> t
(1, 3, 4)
```

Διαγραφή ολόκληρης της πλειάδας

```
thistuple = ("apple", "banana", "cherry")
del(thistuple)
print(thistuple)   ⇨  NameError: name 'thistuple' is not defined
```

Εφαρμογές και παραδείγματα στις πλειάδες *(4/4)*

Μέτρηση του πλήθους εμφανίσεων ενός στοιχείου σε μια πλειάδα

Η μέθοδος `count()` επιστρέφει το πόσες φορές εμφανίζεται ένα συγκεκριμένο στοιχείο σε μια πλειάδα

```
>>> mytuple = (1, 3, 2, "test", 3, 7, 4)
>>> print(mytuple.count(3), mytuple.count("test"), mytuple.count("Test") )
 2, 1, 0
```

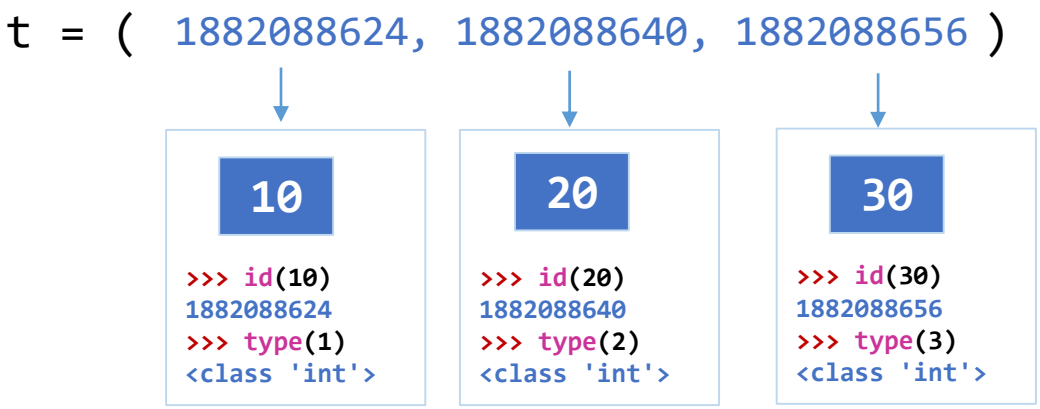
Εύρεση της πρώτης θέσης που εμφανίζεται ενα στοιχείο σε μια πλειάδα

Η μέθοδος `index()` επιστρέφει την θέση που εμφανίζεται για πρώτη φορά ένα στοιχείο σε μια πλειάδα. Αν το στοιχείο δεν υπάρχει, τότε επιστρέφεται μήνυμα λάθους

```
>>> mytuple = (1, 3, 2, "test", 3, 7, 4)
>>> print(mytuple.index(3), mytuple.index("test"))
 1, 3
 >>> print(mytuple.index("Test"))
 ValueError: 'Test' is not in tuple
```

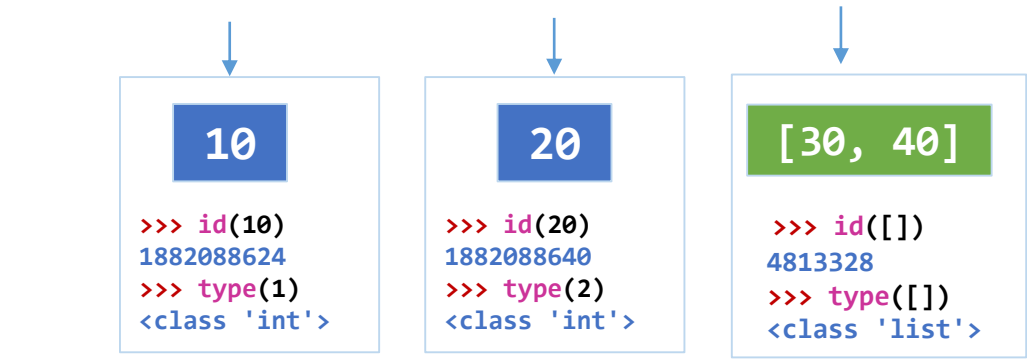
Οι πλειάδες είναι αμετάβλητες. Όμως προσοχή…

```
>>> t = (10, 20, 30)
>>> t[2]
30
>>> t[2]=50 #Αυτό είναι λάθος!
TypeError: 'tuple' object does not
support item assignment
```



Μέσα στην Python, μια πλειάδα αναπαρίσταται από τους δείκτες των αντικειμένων που περιέχει

```
>>> q = (10, 20, [30,40])
>>> q[2]
[30, 40]
>>> q[2]='Hi' #Αυτό είναι λάθος!
TypeError: 'tuple' object does not
support item assignment
>>> q[2][1] = "Hi"  #Ενώ αυτό είναι Ok
>>> q
(10, 20, [30,'Hi'])
```



Αν μια πλειάδα περιέχει μεταβλητά αντικείμενα (πχ λίστες, λεξικά, σύνολα) το περιεχόμενο αυτών των αντικειμένων μπορεί να αλλάζει ελεύθερα

Τέλος Διάλεξης

Ερωτήσεις;

Τμήματα αυτής της διάλεξης περιέχουν πληροφορίες από πηγές που είναι ελεύθερες στο διαδίκτυο όπως η Βικιπαίδεια και ανοιχτές σημειώσεις παρεμφερών διαλέξεων.