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

HMY01K06
Επιστημονικός Προγραμματισμός με Python

6+1 ερωτήσεις επανάληψης για τη Διάλεξη 05

Αντικείμενα και Ονόματα - Δομημένοι τύποι δεδομένων (Λίστες / Πλειάδες)

Φθινόπωρο 2025

Ερώτηση 1/7

Αντικείμενα και ονόματα στην Python. Σωστό ή λάθος;

✅ Η *ταυτότητα* κάθε αντικειμένου είναι *μοναδική (δεν μπορούμε να την αλλάξουμε)*

✅ Ο *τύπος* κάθε αντικειμένου είναι *μοναδικός (δεν μπορούμε να τον αλλάξουμε)*

❌ Το *όνομα* κάθε αντικειμένου είναι *μοναδικό (δεν μπορούμε να το αλλάξουμε)*

✅ Ένα αντικείμενο μπορεί να μην έχει *καθόλου όνομα*

✅ Κάποια αντικείμενα *μας επιτρέπουν* να αλλάξουμε το *περιεχόμενό τους*

❌ Οι *εντολές εκχώρησης* τροποποιούν *τα αντικείμενα* και *όχι* τους *ονοματοχώρους*
(συμβαίνει ακριβώς το αντίθετο)

Ερώτηση 2/7

Μπορούμε να έχουμε *πολυδιάστατες* λίστες στην Python; - Πως τις χειριζόμαστε;

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']
```

Ερώτηση 3/7

Ποια είναι η βασική διαφορά ανάμεσα στις μεθόδους `.append()` και `.extend()`;

`.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;)*

Ερώτηση 4/7

Αν `L` είναι μια λίστα, ποια η διαφορά ανάμεσα στα `del L[:]` και `del L`

- `del L[:]` διαγράφει όλα τα στοιχεία της λίστας αφήνοντάς την κενή
- `del L` διαγράφει εξ' ολοκλήρου τη λίστα από τον ονοματοχώρο

```
>>> L = [1,2,3]
>>> del L[:]
>>> L
[]
```

```
>>> L = [1,2,3]
>>> del L
>>> L
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module> L
NameError: name 'L' is not defined
```

Θυμόμαστε ακόμα ότι:

- `del L[i]` διαγράφει το στοιχείο `i`
- `del L[i:j]` διαγράφει τα στοιχεία από `i` έως `j-1`

Ερώτηση 5/7

Ποια η διαφορά ανάμεσα στα `x = (1)` και `y = (1,)`

Το `x` είναι *ακέραιος*

Το `y` είναι μια *πλειάδα ενός στοιχείου*

```
>>> x = (1)
>>> print(x, type(x))
1 <class 'int'>
>>>
```

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

Ερώτηση 6/7

Υπάρχει *έμμεσος τρόπος αλλαγής* του περιεχόμενου μιας πλειάδας;

- Μετατροπή *πλειάδας* σε *λίστα* με την συνάρτηση `list()`
- *Αλλαγή* περιεχομένου της λίστας
- Μετατροπή *λίστας* πίσω σε *πλειάδα* με την συνάρτηση `tuple()`

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

y = list(x)              #Μετατροπή πλειάδας σε λίστα

y[2] = 'mango'           #Αλλαγή περιεχομένου στοιχείου λίστας

x = tuple(y)             #Μετατροπή λίστας πίσω σε πλειάδα

print(x)

('apple', 'banana', 'mango', 'orange', 'kiwi', 'melon')
```

Bonus Ερώτηση 7/7

Τι θα συμβεί αν προσθέσουμε ένα ακόμα κόμμα
μετά το τελευταίο στοιχείο μιας *λίστας* η μιας *πλειάδας*;

Τίποτα. Η Python μας επιτρέπει να κάνουμε πράγματα όπως:

```
>>> L = [1,2,3,]
>>> L
[1, 2, 3]
```

```
>>> T=(1,2,3,)
>>> T
(1, 2, 3)
```

```
>>> T = 1,2,3,
>>> T
(1, 2, 3)
```

```
>>> T = 1,
>>> T
(1,)
```

Εδώ το κόμμα επιβάλλεται αν θέλουμε να δημιουργήσουμε πλειάδα ενός στοιχείου

Προσοχή όμως!

```
>>> L = [1,2,3],
>>> L
([1, 2, 3],)
```

δημιουργείται μια πλειάδα ενός στοιχείου,
όπου το στοιχείο αυτό είναι η λίστα `[1,2,3]`