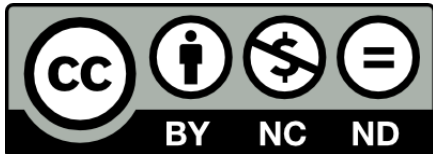


Εισαγωγή στην **Python**

6





Copyright

Το παρόν εκπαιδευτικό υλικό προσφέρεται ελεύθερα υπό τους όρους της άδειας Creative Commons:

- Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Όχι Παράγωγα Έργα 3.0.

Για να δείτε ένα αντίγραφο της άδειας αυτής επισκεφτείτε τον ιστότοπο

<https://creativecommons.org/licenses/by-nc-nd/3.0/gr/>

Στ. Δημητριάδης, 2015

List

Λίστα



Περιεχόμενα

- Λίστα: τα Βασικά χαρακτηριστικά
 - Δομή, δεικτοδότηση, αλλαγή τιμών
 - List & For (iteration) / Τελεστής in / Συνάρτηση List()
- Δημιουργία Λίστας
 - (1) Δημιουργία με **List & Range**
 - (2) Δημιουργία με **Append**
 - (3) Δημιουργία με **Περιγραφή λίστας** (list comprehension)
- Λίστα: Μεταλλάξιμη δομή (mutable)
 - Μεταλλάξιμη & μη-μεταλλάξιμη δομή
- Τομές (Slices)
 - Τομή λίστας, πχ. L[2:5]
- Λίστες λιστών (List of Lists)
 - Λίστες 2D & 3D
- Συναρτήσεις Λίστας
- Μέθοδοι Λίστας

Λίστα

τα **βασικά** χαρακτηριστικά



List

```
>>> lista=[1,2,3,4]
>>> lista
[1, 2, 3, 4]
```

- Η **Λίστα** είναι η πιο γενική και ευέλικτη δομή ακολουθίας της Python
- **Ordered: Ακολουθιακή**
 - Είναι μια **ακολουθιακή** συλλογή αντικειμένων, δηλ. τα απλά στοιχεία της αποτελούν μια ακολουθία με κάποια σειρά
- **Indexed: Δεικτοδοτημένη**
 - η θέση ενός αντικειμένου σε μια Λίστα προσδιορίζεται με χρήση **δείκτη μέσα σε αγκύλες** [index] που γράφεται **δίπλα** στο όνομα της λίστας, πχ. lista[2]
 - **Zero indexed**: ο δείκτης [0] προσδιορίζει το πρώτο στοιχείο / δεν έχει κάποιο προκαθορισμένο περιορισμό στο μέγεθός της
- **Iterable: επαναληπτική**
 - Είναι **επαναληπτική** δομή (iterable) και μπορεί να χρησιμοποιείται σε μια δομή επανάληψης
- **Mutable: Μεταλλάξιμη**
 - Είναι **μεταλλάξιμη** δομή (mutable): δηλ. τα δεδομένα της μπορούν να μεταβληθούν στη θέση μνήμης ('in place') χωρίς να δημιουργηθεί νέα λίστα
- **Heterogeneous: Ανομοιογενής**
 - Μπορεί να περιλαμβάνει δεδομένα διαφορετικού τύπου

List

δεικτοδότηση (indexing) - 1

```
>>> lista=[1,2,3,4,5]
>>> lista[0]
1
>>> lista[4]
5
>>> lista[2]
3
>>> lista[5]
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    lista[5]
IndexError: list index out of range
```

- Η λίστα lista έχει 5 δεδομένα: τους ακεραίους 1 ως και 5
- Ο δείκτης [0] προσδιορίζει το πρώτο στοιχείο
- Ο δείκτης [4] το τελευταίο στοιχείο
- Αν χρησιμοποιήσετε το δείκτη [5] η Python θα σας απαντήσει με IndexError

List

δεικτοδότηση (indexing) - 2

```
>>> lista=['ένα', 'δύο', 'τρία']  
>>> lista[1]  
'δύο'  
>>> lista[0]  
'ένα'  
>>> len(lista)  
3
```

- Λίστα με τρία δεδομένα τύπου αλφαριθμητικού

- Η μέθοδος **len()** επιστρέφει το μήκος της λίστας, δηλ. το πλήθος των αντικειμένων που την αποτελούν

- Η **len()** λειτουργεί παρόμοια και στα αλφαριθμητικά

```
>>> lista[len(lista)-1]  
'τρία'  
>>>
```

- Αναφορά στο τελευταίο αντικείμενο μιας λίστας με χρήση της **len()**

List

δεικτοδότηση (indexing) - 3

```
>>> lista = [1,2,3,4,5]
>>> print(lista[-2])
4
>>> print(lista[-5])
1
>>> print(lista[-6])
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    print(lista[-6])
IndexError: list index out of range
```

- Ο δείκτης μπορεί να είναι και αρνητικός
- Στην περίπτωση αυτή θεωρείται πως η αρίθμηση αρχίζει από το τελευταίο στοιχείο της λίστας
- Γενικά `list_name[-i]` δεικτοδοτεί το **στοιχείο στη θέση i μετρώντας από το τέλος**

List

αλλαγή τιμών changing values

```
>>> lista[0]='δέκα'
>>> lista
['δέκα', 'δύο', 'τρία']
>>> |
```

- Τα αντικείμενα στις θέσεις μιας λίστας μπορούν να αλλάξουν με απλές εντολές ανάθεσης

```
>>> lista[0]=1
>>> lista
[1, 'δύο', 'τρία']
>>> lista[0]=lista[1]+lista[2]
>>> lista
['δύοτρία', 'δύο', 'τρία']
>>> |
```

- Μια λίστα μπορεί να περιέχει αντικείμενα διαφορετικών τύπων

```
>>> lista=[1,2,3,4,5]
>>> lista[1]=lista[2]+lista[4]
>>> lista
[1, 8, 3, 4, 5]
>>> |
```

- Οι πράξεις μεταξύ των αντικειμένων μιας λίστας γίνεται με τους τελεστές που ισχύουν για τον κάθε τύπο δεδομένων

List & For (iteration)

```
for i in [1,2,3]:  
    print(i)
```

```
lista=[1,2,3,4,5]  
for i in lista:  
    print(i)
```

```
zoo=['elephant', 'tiger', 'lion']  
for animal in zoo:  
    print(animal)
```

```
name='Python'  
for i in name:  
    print(i)
```

- Η Λίστα είναι **επαναληπτική δομή (iterable)**
- Έτσι μπορεί να **εμφανίζεται σε μια for δομή επανάληψης** στη θέση όπου χρειάζεται μια επαναληπτική δομή (iterable)

Το ίδιο ισχύει και για τα αλφαριθμητικά

Pythonic syntax:

- Στην **πρώτη** for χρησιμοποιείται ένας γενικός αριθμητικός δείκτης και η range()
- Στη **δεύτερη** for χρησιμοποιείται η λίστα zoo[] και ένας κατάλληλος δείκτης (animal)
- Ο δείκτης animal παίρνει τιμές από τη λίστα zoo και επομένως και τον τύπο δεδομένων της λίστας, δηλ. **αλφαριθμητικό**

Συγκρίνετε:

```
zoo=['elephant', 'tiger', 'lion']  
  
for i in range(3):  
    print(zoo[i])  
  
for animal in zoo:  
    print(animal)
```

- Το αποτέλεσμα είναι το ίδιο, αλλά:
- Η δεύτερη λύση εκτελείται πιο **γρήγορα** και υιοθετεί **απλούστερη** σύνταξη

- **Pythonic:** Η δεύτερη λύση τονίζει το ιδιαίτερο στυλ προγραμματισμού της Python

Αν στην for χρειαζόμαστε και τον αριθμητικό δείκτη επανάληψης;

- Χρησιμοποιήστε τη συνάρτηση **enumerate()** με όρισμα την επαναληπτική δομή όπως βλέπετε στο παράδειγμα δεξιά

```
zoo = ['elephant', 'tiger', 'lion']  
for i, animal in enumerate(zoo):  
    print(i, animal)
```

- Η συνάρτηση επιστρέφει μια **δυάδα** στοιχείων:
- A) τον αριθμητικό δείκτη της επανάληψης (στο παράδειγμα i)
- B) ένα απλό δεδομένο της δομής (στο παράδειγμα animal)
- Δοκιμάστε τον κώδικα αλλάζοντας τη λίστα zoo με:
- range(10)**
- range(ord('A'), ord('Z'))**
- Εξηγήστε τις τιμές που βλέπετε να εμφανίζει η print

List

Ο τελεστής 'in'

```
zoo=['elephant', 'tiger', 'lion']

for i in range(3):
    print(zoo[i])

for animal in zoo:
    print(animal)

if 'dog' in zoo:
    '<do something-1>'
else:
    '<do something-2>|'

if 'dog' in zoo:
    pass
else:
    pass

print('tiger' in zoo)
print('cat' in zoo)
```

- Ο τελεστής **in** γενικά **τοποθετεί** ένα απλό αντικείμενο σε μια δομή αντικειμένων (ακολουθία, λίστα κλπ.)
 - όπως πχ. στη σύνταξη της for

- **Membership:**
- Επίσης **ελέγχει αν** ένα αντικείμενο **είναι μέλος** μιας δομής αντικειμένων και επιστρέφει τη λογική τιμή **True** ή **False** ανάλογα

List

η συνάρτηση **list()**

```
>>> spam = 'TEST'
>>> spam
'TEST'
>>> listspam = list(spam)
>>> listspam
['T', 'E', 'S', 'T']
```

- Η `list()` μετατρέπει μια επαναληπτική δομή σε **→ Λίστα**
- Είναι ο 'κατασκευαστής' αντικειμένων Λίστας
- Λειτουργεί όπως οι `int()`, `float()`, `str()` για τους αντίστοιχους τύπους δεδομένων



Δημιουργία Λίστας



(1) Δημιουργία Λίστας με **list()** & **range()**

```
>>> spam=list(range(10))  
>>> spam  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Συνδυάστε τις **list()** & **range()** για να δημιουργήσετε μια λίστα διαδοχικών αριθμών
- Η **range()** επιστρέφει μια ακολουθιακή δομή
- Η **list(range())** μετατρέπει τη δομή σε λίστα

List

Προσθήκη στοιχείων με **append** Διαγραφή στοιχείων με **del**

Η διαγραφή και προσθήκη στοιχείων σε λίστα γίνεται με τις μεθόδους **del** & **append** αντίστοιχα

```
zoo = ['elephant', 'tiger', 'lion']  
  
for animal in zoo:  
    print(animal)  
  
del zoo[1]  
print(zoo) → ['elephant', 'lion']  
  
zoo.append('tiger')  
print(zoo)  
→ ['elephant', 'lion', 'tiger']
```

- **del**: αφαιρεί ένα στοιχείο λίστας
 - Η del είναι **εντολή** (statement)
- **append**: προσθέτει ένα στοιχείο στη λίστα
 - Στην **τελευταία** θέση
 - Η append είναι **μέθοδος** (method) της δομής λίστας

(2) Δημιουργία Λίστας με τη μέθοδο **append**

```
listnum=[]  
for i in range(10):  
    listnum.append(i)  
print(listnum)
```

- Πρώτα δημιουργήστε μια **κενή** λίστα
- Στη συνέχεια γράψτε ένα **βρόχο** for και **καλέστε την append** τόσες φορές ώστε να συμπληρωθεί η λίστα με τα δεδομένα που θέλετε

→ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
zoo = []  
for i in range(2):  
    zoo.append('elephant')  
    zoo.append('tiger')  
    zoo.append('lion')  
  
print(zoo)
```

- Τί λίστα θα δημιουργήσει ο κώδικας του παραδείγματος αριστερά;

Δημιουργία Λίστας με την **append**

Παράδειγμα - 1

```
import random

lista=[]
for i in range(10):
    lista.append(random.randint(1,20))

print(lista)
```

- Τί λίστα θα δημιουργήσει ο κώδικας του παραδείγματος;
- Χρησιμοποιούμε ένα βρόχο for και την **append** για να δημιουργήσουμε τη λίστα lista με 10 ψευδοτυχαίους αριθμούς στο διάστημα [1,20]

Δημιουργία Λίστας με την **append**

Παράδειγμα - 2

```
lista=[]  
for i in range(2):  
    lista.append(1)  
  
for i in range(2,10):  
    lista.append(lista[i-2]+lista[i-1])  
  
print(lista)
```



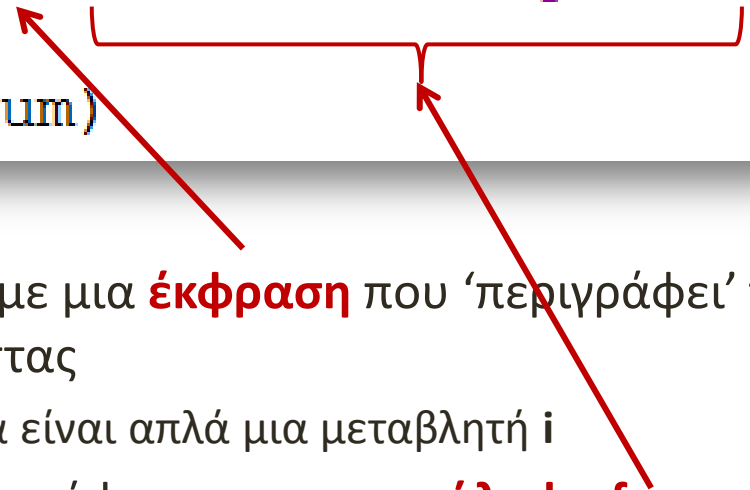
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

- Τί λίστα θα δημιουργήσει ο κώδικας του παραδείγματος;
- Χρησιμοποιούνται δύο βρόχοι for και η **append** για να δημιουργηθεί μια λίστα με τους 10 πρώτους αριθμούς της ακολουθίας Fibonacci

(3) Δημιουργία Λίστας με την τεχνική: **‘περιγραφή λίστας’** (list comprehension)

- Στην **‘περιγραφή λίστας’** δίνουμε μια “περιγραφή” των στοιχείων της λίστας περικλείοντάς την μέσα σε αγκύλες

```
listnum = [i for i in range(10)]  
print(listnum)
```



Γρηγορότερος και οικονομικότερος προγραμματιστικά τρόπος δημιουργίας λίστας

- **(α)** Πρώτα δίνουμε μια **έκφραση** που ‘περιγράφει’ τη **μορφή** του κάθε στοιχείου της λίστας
 - Στο παράδειγμα είναι απλά μια μεταβλητή *i*
- **(β)** Στη **συνέχεια** γράφουμε μια **επανάληψη for** που περιγράφει το εύρος μεταβολής του απαριθμητή *i*
- Η συγκεκριμένη **περιγραφή** δημιουργεί λίστα που περιλαμβάνει 10 ακραίους με τιμές στο διάστημα [0,9]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

- Η περιγραφή μπορεί να χρησιμοποιεί μια **απλή ή σύνθετη έκφραση** για να προσδιορίσει τα στοιχεία της λίστας

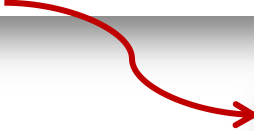
```
listnum = [i-1 for i in range(10)]
```

 [-1, 0, 1, 2, 3, 4, 5, 6, 7, 8]

```
listnum = [2*i for i in range(10)]
```

 [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

```
listnum = [(2*(i**2)+4*i) for i in range(10)]
```

 [0, 6, 16, 30, 48, 70, 96, 126, 160, 198]

- Η περιγραφή μπορεί να χρησιμοποιεί **αλφαριθμητικά** και κάθε επιτρεπόμενο τελεστή

```
lista = ['Λέξη' for i in range(5)]
```

→ ['Λέξη', 'Λέξη', 'Λέξη', 'Λέξη', 'Λέξη']

```
lista = ['Λέξη'+ str(i) for i in range(5)]
```

→ ['Λέξη0', 'Λέξη1', 'Λέξη2', 'Λέξη3', 'Λέξη4']

- **Απλές δομές if/else** (όχι elif) μπορούν να χρησιμοποιούνται στην έκφραση της μορφής δεδομένων της λίστας

ΕΚΦΡΑΣΗ

```
lista = ['1' if i<3 else '2' for i in range(1,6)]
```

['1', '1', '2', '2', '2']

```
lista = [0 if i%2==0 else 1 for i in range(10)]
```

[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]


- Όμως δεν μπορείτε να κάνετε αναφορά στη λίστα μέσα στην δήλωση περιγραφής της

```
lista = [1 if i in [0,1] else (i-1)+(i-2) for i in range(20)]
```



```
[1, 1, 1, 3, 5, 7, 9, 11, 13, 15]
```

```
lista = [1 if i in [0,1] else lista[i-1]+lista[i-2] for i in range(20)]
```



```
Traceback (most recent call last):  
  File "C:/Python34/myApps/todel.py", line  
    lista = [1 if i in [0,1] else lista[i-1]  
20)]  
  File "C:/Python34/myApps/todel.py", line  
    lista = [1 if i in [0,1] else lista[i-1]  
20)]  
NameError: name 'lista' is not defined
```

- **Κλήση απλών συναρτήσεων** μπορεί να γίνει μέσα στην περιγραφή λίστας
- Στο παράδειγμα η `chr()` επιστρέφει τον ASCII χαρακτήρα της τιμής του `i`

```
lista = [chr(i) for i in range(65, 70)]
```



```
['A', 'B', 'C', 'D', 'E']
```

```
lista = [2**i if i<5 else chr(65+2*i) for i in range(10)]
```



```
[1, 2, 4, 8, 16, 'K', 'M', 'O', 'Q', 'S']
```

Δυο λόγια για τον όρο ‘list comprehension’ ή ‘περιγραφή λίστας’

- Στην Αγγλική ο όρος ‘**comprehension**’ έχει τη σημασία: (α) της αντίληψης, κατανόησης ενός αντικειμένου, και (β) της συμπερίληψης, δηλ. της ένταξης σε ένα σύνολο ([δείτε online ορισμό](#))
- Στη θεωρία συνόλων ([set theory](#)) και την επιστήμη υπολογιστών ([computer science](#)) χρησιμοποιείται ο όρος ‘**set comprehension**’ (ή **set abstraction**) για να αναφερθούμε στον ορισμό ενός συνόλου μέσω του ακριβούς προσδιορισμού των ιδιοτήτων που πρέπει να πληρούν τα στοιχεία ώστε να συμπεριληφθούν στο σύνολο.
- Στον προγραμματισμό ο όρος ‘**list comprehension**’ αναφέρεται σε μια τεχνική προγραμματισμού κατά την οποία μια λίστα δημιουργείται μέσω της αναλυτικής περιγραφής των ιδιοτήτων των στοιχείων που περιλαμβάνει. Στα Ελληνικά έχει αποδοθεί ως «**συμπερίληψη**» .
- Καθώς η τεχνική ‘list comprehension’ απαιτεί την **αφηρημένη περιγραφή** των ιδιοτήτων των στοιχείων που θα συμπεριληφθούν στη λίστα, προτείνουμε την απόδοση του όρου στα Ελληνικά ως ‘**περιγραφή λίστας**’ , θεωρώντας ότι είναι απλούστερος και περισσότερο κατανοητός.

Λίστα: **μεταλλάξιμη** δομή

Mutable



Τρέξτε τον παρακάτω κώδικα

```
list1 = [1, 2, 3, 4, 5]
list2 = list1
list2[0] = 'X'
print(list1, '\n', list2)
```

```
['X', 2, 3, 4, 5]
['X', 2, 3, 4, 5]
```

- Γιατί αφού αλλάζουμε τιμή στη θέση [0] της list2 **αλλάζει και η τιμή του πρώτου στοιχείου στην list1** ;

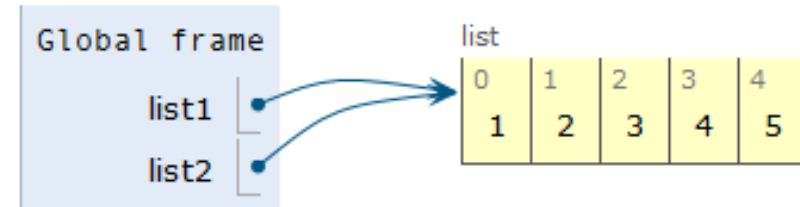
Ο ίδιος κώδικας στον Online Python Tutor

Python 3.3

```
1 list1 = [1, 2, 3, 4, 5]
2
→ 3 list2 = list1
4
→ 5 list2[0] = 'X'
6
7 print(list1, '\n', list2)
```

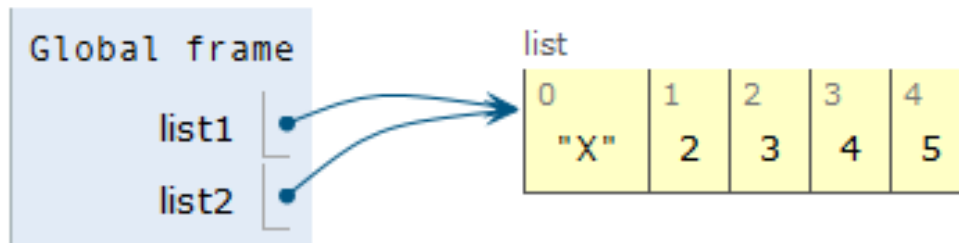
Frames

Objects



Frames

Objects



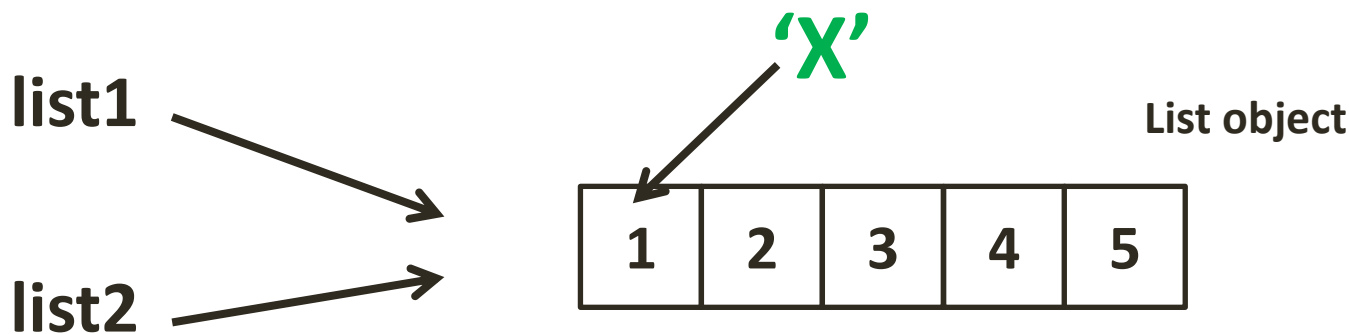
- Τώρα μπορείτε να εξηγήσετε τι συμβαίνει;



Η απλή εξήγηση: **αναφορά** ονομάτων στο **ίδιο** αντικείμενο

- Η εντολή `list2 = list1`

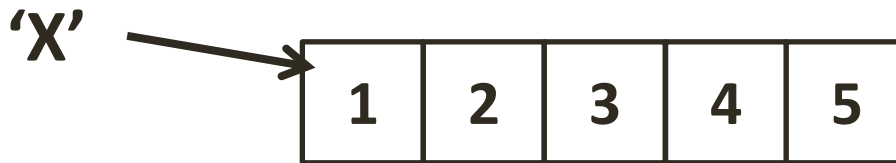
απλά συνδέει το νέο όνομα list2 με την υπάρχουσα δομή λίστας με την οποία συνδέεται ήδη το όνομα list1



- Τα ονόματα list1 & list2 **αναφέρονται** στην ίδια δομή (shared reference)
- Επομένως κάθε αλλαγή τιμής στη δομή συνδέεται και με τα δύο ονόματα

Η σύνθετη εξήγηση: **Μεταλλάξιμο** αντικείμενο (**mutable** object)

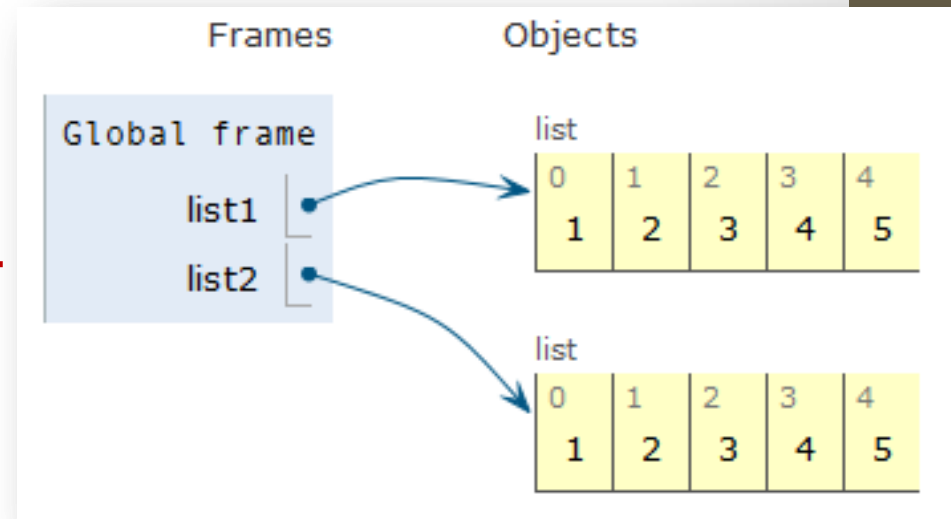
- Μια λίστα είναι **μεταλλάξιμο** αντικείμενο: μπορούν να γίνουν αλλαγές στις τιμές των στοιχείων της στο χώρο μνήμης που καταλαμβάνει το αντικείμενο, χωρίς να δημιουργηθεί νέο (δηλ. νέα λίστα)



- Μεταλλάξιμο (mutable)** είναι κάθε αντικείμενο το οποίο μπορεί να **αλλάξει μορφή** (πχ. να αλλάξει την τιμή του) **χωρίς** να δημιουργηθεί **νέο** αντικείμενο στη μνήμη
 - δηλ. η αλλαγή γίνεται στον χώρο μνήμης που ήδη καταλαμβάνει το αντικείμενο
 - mutable** → μεταλλάξιμο
 - immutable** → μη μεταλλάξιμο, αμετάλλακτο

Και αν θέλω μια πραγματικά **νέα** λίστα;

```
list1 = [1, 2, 3, 4, 5]
list2 = list1[:]
list2[0] = 'X'
print(list1, '\n', list2)
```



```
[1, 2, 3, 4, 5]
['X', 2, 3, 4, 5]
```

- Δημιουργήστε νέα λίστα με την τεχνική της **Τομής** (slicing)
 - (δείτε επόμενη ενότητα)

Τομές σε λίστες

Slices



List

Τομή

- Μια λίστα μπορεί να "κοπεί" σε τμήματα (slices) και κάθε τμήμα είναι μια νέα υπο-λίστα (sublist)

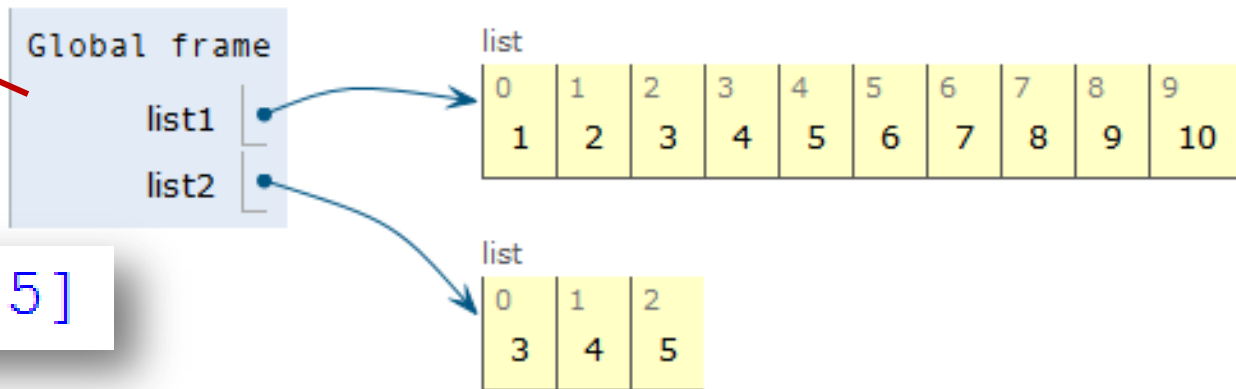
list2 = list1[start : end : step]

```
list1 = [1,2,3,4,5,6,7,8,9,10]
```

```
list2 = list1[2:5]
```

```
print(list2)
```

[3, 4, 5]



- Η list2 θα έχει τα στοιχεία της list1 που βρίσκονται στο διάστημα [2, 5)
 - το στοιχείο στη θέση με δείκτη 5 **δεν** περιλαμβάνεται

List

Τομή

Παραδείγματα

```
spam = ['A', 'B', 'Γ', 'Δ', 'Ε']
```

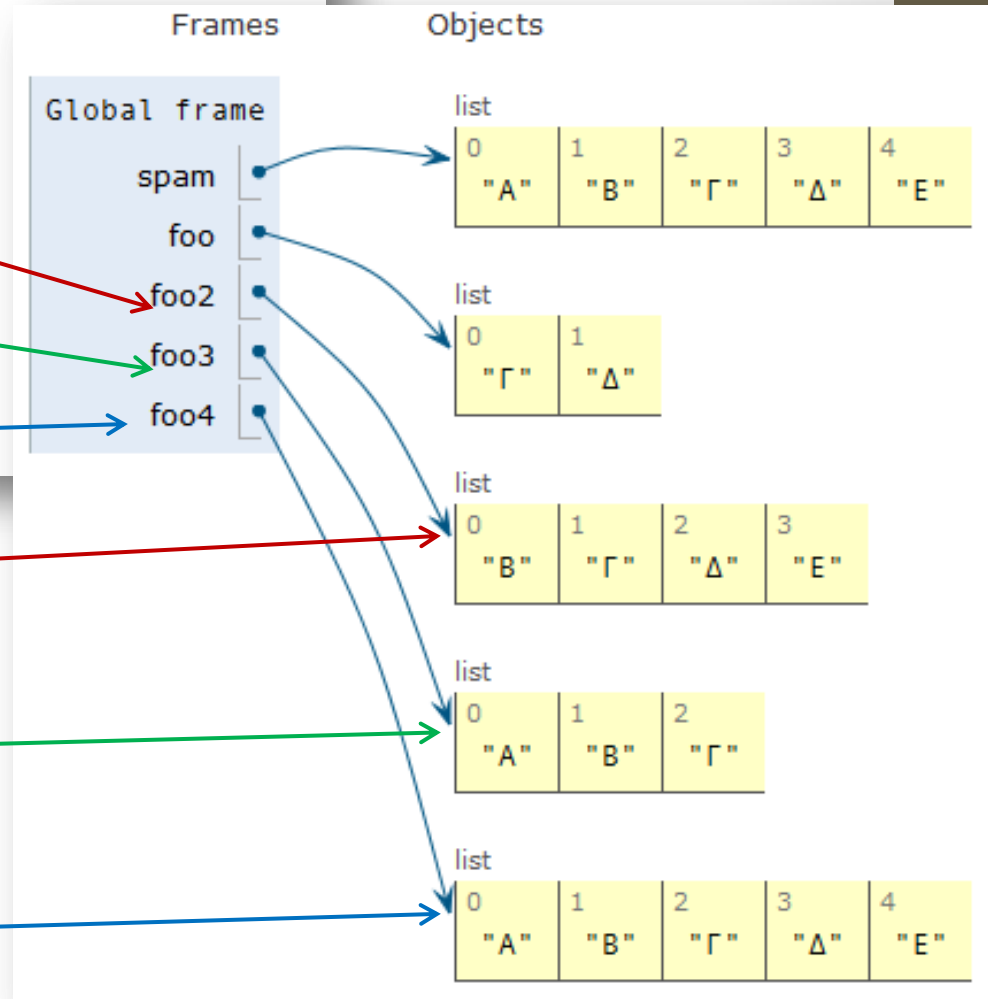
```
foo = spam[2:4]
```

```
foo2 = spam[1:]
```

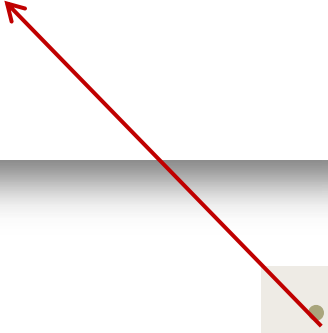
```
foo3 = spam[:3]
```

```
foo4 = spam[:]
```

- Αν λείπει, τότε ως **πάνω όριο** θεωρείται το τελικό στοιχείο της λίστας (περιλαμβάνεται)
- Αν λείπει, τότε ως **κάτω όριο** θεωρείται το πρώτο στοιχείο της λίστας (περιλαμβάνεται)
- Αν **λείπουν και τα δύο** πάνω και κάτω όριο τότε δημιουργείται μια νέα ακριβώς ίδια λίστα



```
>>> lista = [1,2,3,4,5]
>>> lista[-3:-1]
[3, 4]
>>> lista[-2:]
[4, 5]
```



- Μπορούν να δημιουργηθούν τομές και με αρνητικούς δείκτες
- Στην περίπτωση αυτή η μέτρηση θεωρείται πως αρχίζει από το τελευταίο στοιχείο της λίστας

```
lista[start:end]      # Από start μέχρι end-1
lista[start:]         # Από start μέχρι τέλους της λίστας
lista[:end]           # Από την αρχή της λίστας μέχρι end-1
lista[:]              # Αντίγραφο ολόκληρης της λίστας
lista[start:end:step] # Από start μέχρι πριν το end, με βήμα step
```

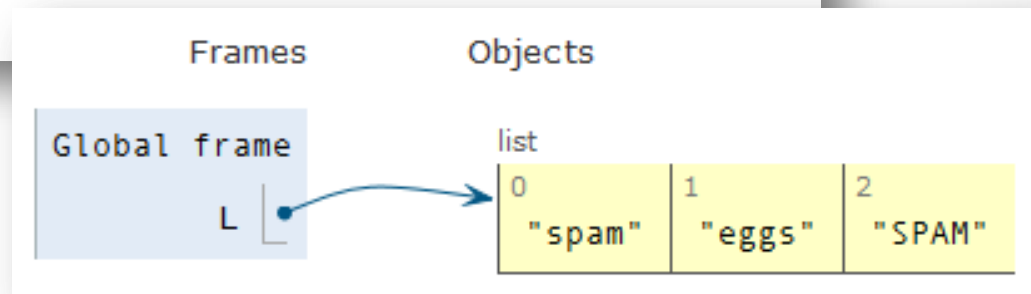
- Προσοχή στο end-1 όταν η τομή της λίστας έχει άνω όριο το στοιχείο end

List

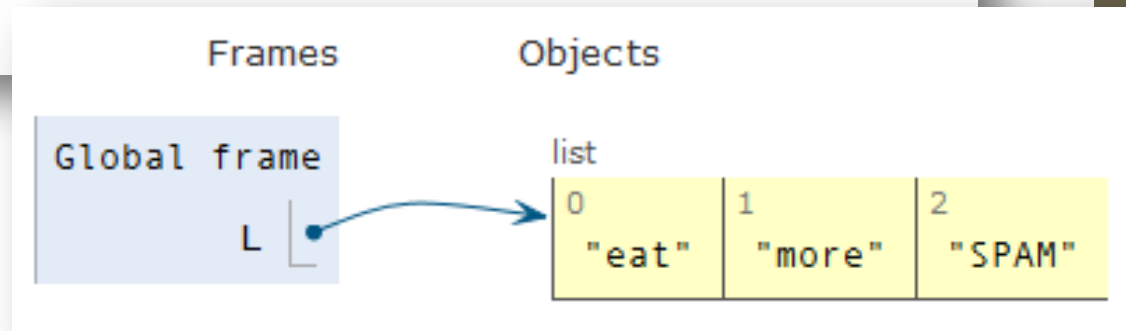
Αλλαγή Τιμών με Τομή

Παραδείγματα - 1

```
>>> L = ['spam', 'Spam', 'SPAM!']  
>>> L[1] = 'eggs' ← # ΑΛΛΑΓΗ ΤΙΜΗΣ ΣΕ ΜΙΑ ΘΕΣΗ  
>>> L  
['spam', 'eggs', 'SPAM!']
```



```
>>> L[0:2] = ['eat', 'more'] ← # ΑΛΛΑΓΗ ΤΙΜΗΣ ΣΕ ΤΜΗΜΑ  
>>> L  
['eat', 'more', 'SPAM!']  
# ΑΛΛΑΖΟΥΝ ΤΙΜΗ ΤΑ ΣΤΟΙΧΕΙΑ 0 & 1
```



List

Αλλαγή Τιμών με Τομή

Παραδείγματα - 2

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

```
L[1:2] = [4, 5]
```

```
L[1:1] = [6, 7]
```

```
L[1:2] = []
```

Frames

Objects

Global frame

L

list

0	1	2	3
1	4	5	3

ΑΝΤΙΚΑΤΑΣΤΑΣΗ & ΕΙΣΑΓΩΓΗ

Frames

Objects

Global frame

L

list

0	1	2	3	4	5
1	6	7	4	5	3

ΕΙΣΑΓΩΓΗ (ΧΩΡΙΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗ)

Frames

Objects

Global frame

L

list

0	1	2	3	4
1	7	4	5	3

ΔΙΑΓΡΑΦΗ (ΧΩΡΙΣ ΕΙΣΑΓΩΓΗ)



List

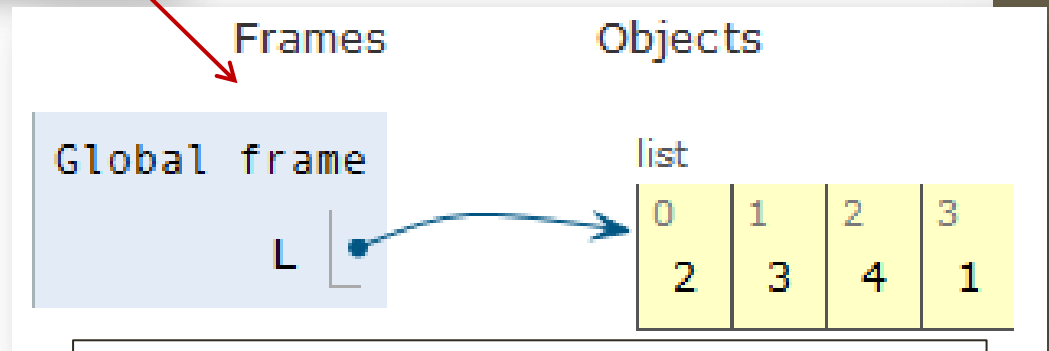
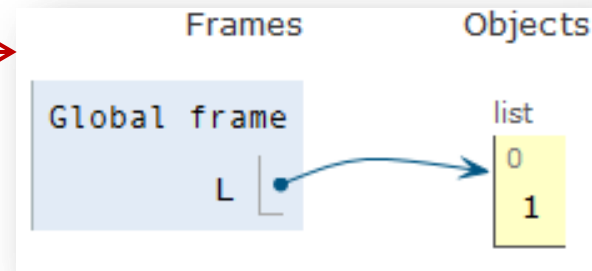
Αλλαγή Τιμών με Τομή

Παραδείγματα - 3

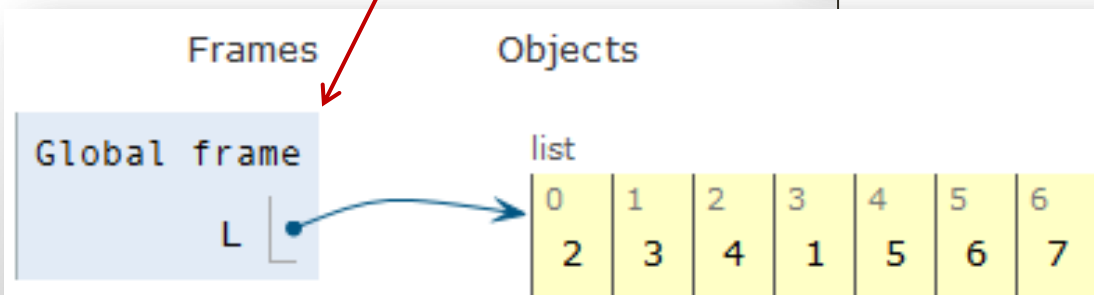
```
L = [1]
```

```
L[:0] = [2, 3, 4]
```

```
L[len(L):] = [5, 6, 7]
```



ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΡΧΗ ΤΗΣ ΛΙΣΤΑΣ



ΕΙΣΑΓΩΓΗ ΣΤΟ ΤΕΛΟΣ ΤΗΣ ΛΙΣΤΑΣ



Συμπεράσματα: Αλλαγή Τιμών με Τομή

- **ΑΛΛΑΓΗ:** Όταν προσδιορίζουμε **μια τομή** της λίστας τότε οι νέες τιμές αντικαθιστούν τις επιλεγμένες τιμές, όπως στα παραδείγματα:
 - $L[1:2] = [4, 5]$
 - $L[:0] = [2, 3, 4]$
- **ΕΙΣΑΓΩΓΗ:** Όταν προσδιορίζουμε **ένα σημείο (όχι τομή)** της λίστας τότε οι νέες τιμές εισάγονται χωρίς να επηρεάζονται οι άλλες τιμές, όπως στα παραδείγματα:
 - $L[1:1] = [6, 7]$
 - $L[len(L):] = [5, 6, 7]$

Python indexes and slices for a six-element list.

Indexes enumerate the elements, slices enumerate the spaces between the elements.

Index from rear:	-6	-5	-4	-3	-2	-1	<code>a=[0,1,2,3,4,5]</code>	<code>a[1:]==[1,2,3,4,5]</code>	
Index from front:	0	1	2	3	4	5	<code>len(a)==6</code>	<code>a[:5]==[0,1,2,3,4]</code>	
	+---+---+---+---+---+---+						<code>a[0]==0</code>	<code>a[:-2]==[0,1,2,3]</code>	
	a	b	c	d	e	f	<code>a[5]==5</code>	<code>a[1:2]==[1]</code>	
	+---+---+---+---+---+---+						<code>a[-1]==5</code>	<code>a[1:-1]==[1,2,3,4]</code>	
Slice from front:	:	1	2	3	4	5	:	<code>a[-2]==4</code>	
Slice from rear:	:	-5	-4	-3	-2	-1	:		
							<code>b=a[:]</code>		
							<code>b==[0,1,2,3,4,5]</code>	(shallow copy of a)	

Πηγή: <https://wiki.python.org/moin/MovingToPythonFromOtherLanguages>

Ερωτήσεις

Ποια θα είναι η μορφή της λίστας που θα προκύψει μετά την εκτέλεση των εντολών ανάθεσης;

```
L = [1, 2, 3, 4, 5]  
L[1:3] = ['A', 'B']  
#-----
```

```
L = [1, 2, 3, 4, 5]  
L[0:1] = [10]  
#-----
```

```
L = [1, 2, 3, 4, 5]  
L[4:] = [20]  
#-----
```

```
L = [1, 2, 3, 4, 5]  
L[5:] = [20]  
#-----
```

```
L = [1, 2, 3, 4, 5]  
L[2:2] = [10]  
#-----
```

```
L = [1, 2, 3, 4, 5]  
L[4:4] = [20, 30]  
#-----
```

```
L = [1, 2, 3, 4, 5]  
L[1: len(L)-1] = [10, 20]  
#-----
```

```
L = [1, 2, 3, 4, 5]  
L[1:3] = []  
#-----
```

```
L = [1, 2, 3, 4, 5]  
L[1:1] = []
```

Λίστα Λιστών

List of Lists (LoL)

ή Λίστες διάστασης >1

List of lists

Λίστα λιστών

- Τα στοιχεία μιας **λίστας** μπορεί να είναι τα ίδια **λίστες**
- Τότε μιλάμε για **λίστα λιστών** ή **λίστα 2D**

```
>>> list1=[1,2,3]
>>> list1
[1, 2, 3]
>>>
>>> list2=[[0,5,10],['A','B','C'],[7.3, 20.2]]
>>> list2
[[0, 5, 10], ['A', 'B', 'C'], [7.3, 20.2]]
```

- Παράδειγμα
- list2: κάθε στοιχείο της είναι μια **απλή λίστα**
- Για τον προσδιορισμό ενός απλού στοιχείου της χρειάζονται **2 δείκτες**
- Πχ. list2[0][1] είναι το στοιχείο 5

List of lists Λίστα λιστών - Δεικτοδότηση

```
>>> list2=[[0,5,10],['A','B','C'],[7.3, 20.2]]
>>> list2[1][2]
'C'
>>> list2[2][1]
20.2
>>> list2[0][0]
0
```

- Ο πρώτος δείκτης αναφέρεται πάντοτε στα στοιχεία της **κύριας λίστας**
- Ο δεύτερος δείκτης προσδιορίζει τα στοιχεία της **λίστας-στοιχείου**
- **Ερώτηση:** ποιοι δείκτες προσδιορίζουν τα απλά στοιχεία 5, 7.3, 'B'
- **List2[:,:]**
- **list2[1]** Ποιο στοιχείο είναι;


```
>>> list1 = [[0 for i in range(5)] for k in range(3)]
>>> list1
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

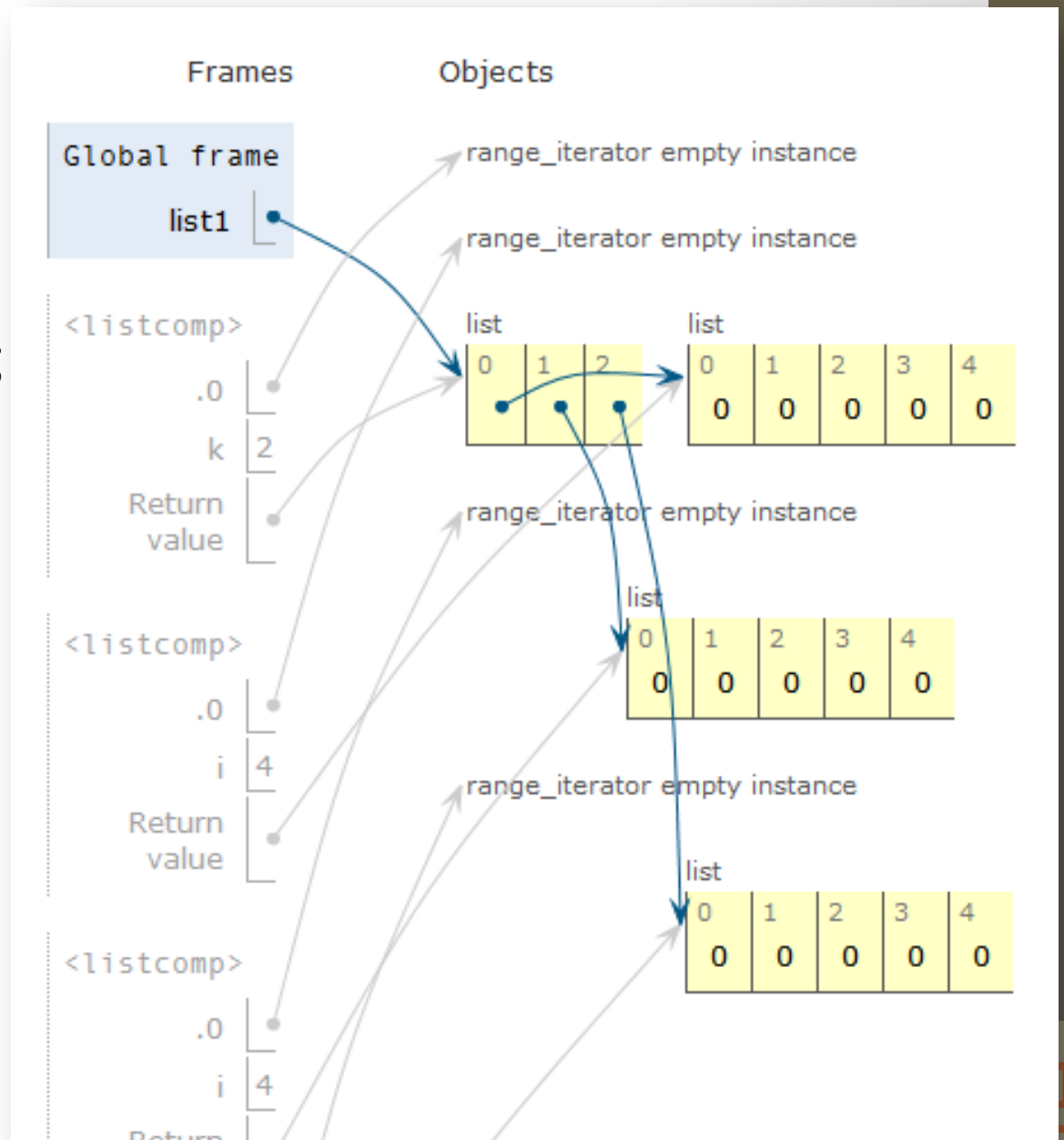
- Για την περιγραφή λίστας λιστών (2D) χρειάζεστε 2 ‘φωλιασμένους’ βρόχους for (όπως στο παράδειγμα)
- Παράδειγμα
- **for k in range(3):** δημιουργεί τη λίστα list1 η οποία έχει 3 στοιχεία
- **for i in range(5):** δημιουργεί τις 3 λίστες-στοιχεία
- Κάθε λίστα-στοιχείο είναι λίστα με 5 απλά δεδομένα
- Κάθε απλό δεδομένο έχει την τιμή 0

3 X 5

στη μνήμη...

- Η οργάνωση της λίστας λιστών στη μνήμη που δημιουργεί ο προηγούμενος κώδικας:

- **Λίστα 3 X 5**
 - Δηλ. λίστα με 3 στοιχεία καθένα από τα οποία είναι λίστα 5 στοιχείων



3 X 5 στην εκτύπωση ...

- Συνολικά η list1

```
list1 = [[0 for i in range(5)] for k in range(3)]  
print(list1)
```

```
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

- Ανά λίστα-στοιχείο

```
for j in list1:  
    print(j)
```

```
[0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0]
```

- Αναλυτικά τα απλά στοιχεία

```
for j in list1:  
    for n in j:  
        print(n, end=' ')
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```



```
list1 = [[1 for i in range(4)] for k in range(2)]  
print(list1)
```

- Τι μορφή έχει η λίστα list1;

2 X 4

- Τι θα τυπώσει η print;

```
[[1, 1, 1, 1], [1, 1, 1, 1]]
```

```
for j in range(2):  
    print(list1[j])
```

```
for j in list1:  
    print(j)
```

- Τι θα τυπώσει τώρα η print;
- Γιατί;

```
[1, 1, 1, 1]  
[1, 1, 1, 1]
```

```
list1 = [[0 if k<=2 else 1 for i in range(3)] for k in range(5)]  
print(list1)
```

- Τι μορφή έχει η λίστα list1;
- Τι θα τυπώσει η print;

5 X 3

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [1, 1, 1], [1, 1, 1]]
```

```
for j in range(5):  
    print(list1[j])
```

```
for j in list1:  
    print(j)
```

- Τι θα τυπώσει τώρα η print;
- Γιατί;

```
[0, 0, 0]  
[0, 0, 0]  
[0, 0, 0]  
[1, 1, 1]  
[1, 1, 1]
```

- Τι θα τυπώσει η
print(len(list1))
- Γιατί;



List of lists

Δεικτοδότηση λίστας 2D

```
list1 = [[i+3*k for i in range(3)] for k in range(5)]  
for j in list1:  
    print(j)
```

```
[0, 1, 2]  
[3, 4, 5]  
[6, 7, 8]  
[9, 10, 11]  
[12, 13, 14]
```

- Ποια στοιχεία προσδιορίζουν οι παρακάτω εκφράσεις:
 - list1[0][0]
 - list1[1][2]
 - list1[2][1]
 - list1[3][3]
 - list1[4][2]
- Ποιες εκφράσεις προσδιορίζουν τα στοιχεία:
 - 2
 - 4
 - 8
 - 9
 - 13

List of lists

Λίστα λιστών 3D

```
lista=[[0 for i in range(7)] for j in range(5)] for k in range(3)]
```

- Τι μορφής λίστα δημιουργεί ο παραπάνω κώδικας; **3 x 5 x 7**
- Τι τιμή θα επιστρέψει η len(lista); **3**
- Πόσες λίστες-στοιχεία έχει η lista; **3**
- Πόσες λίστες-στοιχεία 2^{ου} επιπέδου έχει η lista; **3 x 5 = 15**
- Πόσα απλά στοιχεία έχει η lista; **3 x 5 x 7 = 105**
- Πόσους δείκτες χρειάζεστε για να αναφερθείτε σε ένα απλό στοιχείο της lista; **3**
- Γράψτε την print που εμφανίζει τη lista: (α) ανά λίστα-στοιχείο και (β) ανά λίστα-στοιχείο 2^{ου} επιπέδου στην οθόνη
- Γράψτε τις εκφράσεις που αναθέτουν τιμή στο “πρώτο” απλό στοιχείο της λίστας καθώς και στο “τελευταίο”

Συναρτήσεις Λίστας



Συναρτήσεις Λίστας

Πίνακας συναρτήσεων της Python 3.3

- | | |
|---|--|
| 1 | <u><code>len(list)</code></u>
Επιστρέφει το μήκος (πλήθος στοιχείων της λίστας) |
| 2 | <u><code>max(list)</code></u>
Επιστρέφει το στοιχείο με τη μέγιστη τιμή στη λίστα |
| 3 | <u><code>min(list)</code></u>
Επιστρέφει το στοιχείο με την ελάχιστη τιμή στη λίστα |
| 4 | <u><code>list(seq)</code></u>
Μετατρέπει μια ακολουθιακή δομή (πχ. string) σε Λίστα |



Μέθοδοι Λίστας



Μέθοδοι Λίστας (1/2)

περισσότερα...

1

list.append(x)

Προσθέτει το x στο τέλος της λίστας. Ισοδύναμη με $a[\text{len}(a):] = [x]$.

2

list.extend(L)

Επεκτείνει τη λίστα προσθέτοντας την επαναληπτική δομή L στο τέλος της λίστας. Ισοδύναμη με $a[\text{len}(a):] = L$.

3

list.insert(i, x)

Εισάγει το x στη θέση i. Το i είναι ο δείκτης θέσης του στοιχείου πριν από το οποίο θα γίνει η εισαγωγή. Πχ. `a.insert(0, x)` εισάγει στην αρχή της λίστας.

4

list.remove(x)

Απομακρύνει το στοιχείο x από τη λίστα (από την πρώτη θέση όπου το x εντοπίζεται). Αν δεν υπάρχει x επιστρέφει λάθος.

5

list.pop([i])

Αφαιρεί ΚΑΙ επιστρέφει το στοιχείο που βρίσκεται στη θέση i της λίστας. Αν δεν προσδιοριστεί η θέση τότε η `a.pop()` αφαιρεί και επιστρέφει το τελευταίο στοιχείο της λίστας.

6

list.clear()

Αφαιρεί όλα τα στοιχεία της λίστας. Ισοδύναμη με `del a[:]`.

Μέθοδοι Λίστας (2/2)

[περισσότερα...](#)

- | | |
|----|---|
| 7 | list.index(x)
Επιστρέφει το δείκτη θέσης όπου βρίσκεται το x (στην πρώτη θέση όπου εντοπίζεται). Αν δεν υπάρχει επιστρέφει λάθος. |
| 8 | list.count(x)
Επιστρέφει το πόσες φορές εντοπίζεται το x στη λίστα. |
| 9 | list.sort()
Ταξινομεί τα στοιχεία της λίστας ('in place'). |
| 10 | list.reverse()
Αντιστρέφει τη σειρά των στοιχείων της λίστας ('in place'). |
| 11 | list.copy()
Δημιουργεί αντίγραφο της λίστας. Ισοδύναμη με a[:]. |



Σειριακή αναζήτηση

5	22	54	21	12	7	35	87	92	41
↑?	↑?	↑?							

```
import random as rn

mylist = [rn.randint(1,100) for k in range(10)]
print('Μέγεθος Λίστας:', len(mylist))
print('Μέγιστος:', max(mylist), 'Ελάχιστος:', min(mylist))
print(mylist)

# input
num = int(input('Αριθμός (1-100): '))

print('Σειριακή αναζήτηση')
for index, x in enumerate(mylist):
    if x==num:
        print('Ο αριθμός', num, 'βρέθηκε στη θέση: ', index+1)
        break
else:
    print('Ο αριθμός', num, 'δεν βρέθηκε')
```

Δυαδική αναζήτηση

start		end						end	
5	7	12	21	22	35	41	54	87	92
	↑?			↑?					

```
print('Δυαδική αναζήτηση')
num = int(input('Αριθμός (1-100): '))
sortlist = sorted(mylist)
print(sortlist)
start = 0
end = len(sortlist)-1
while start<=end:
    mid = (start + end)//2
    if sortlist[mid] == num:
        print('Ο αριθμός', num, 'βρέθηκε στη θέση: ', mid+1)
        break
    else:
        if num < sortlist[mid]:
            end = mid-1
        else:
            start = mid+1
else:
    print('Ο αριθμός', num, 'δεν βρέθηκε')
```

Ουρά (Queue)

FIFO



```
# Ουρά Queue
qlist = [0 for i in range(5)]
print(qlist)
inp = None

while inp!='q':
    inp = input('Νέο στοιχείο: ')
    qlist.pop(0)
    qlist.append(inp)
    print(qlist)
```

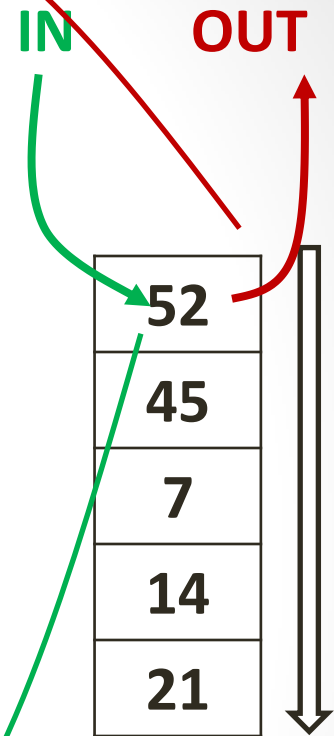
Στοίβα (Stack)

LIFO

```
# Στοίβα Stack
lista = ['0']
print(lista)
inp = None

while True:
    inp = input('Νέο στοιχείο/r/q:')
    if inp == 'q':
        break
    elif inp == 'r':
        lista.pop()
    else:
        lista.append(inp)
    print(lista)

print('Τέλος')
```



Σύνοψη

- Λίστα: τα Βασικά χαρακτηριστικά
 - Δομή, δεικτοδότηση, αλλαγή τιμών
 - List & For (iteration) / Τελεστής in / Συνάρτηση List()
- Δημιουργία Λίστας
 - (1) Δημιουργία με **List & Range**
 - (2) Δημιουργία με **Append**
 - (3) Δημιουργία με **Περιγραφή λίστας** (list comprehension)
- Λίστα: Μεταλλάξιμη δομή (mutable)
 - Μεταλλάξιμη & μη-μεταλλάξιμη δομή
- Τομές (Slices)
 - Τομή λίστας, πχ. L[2:5]
- Λίστες λιστών (List of Lists)
 - Λίστες 2D & 3D
- Συναρτήσεις Λίστας
- Μέθοδοι Λίστας

Ασκήσεις



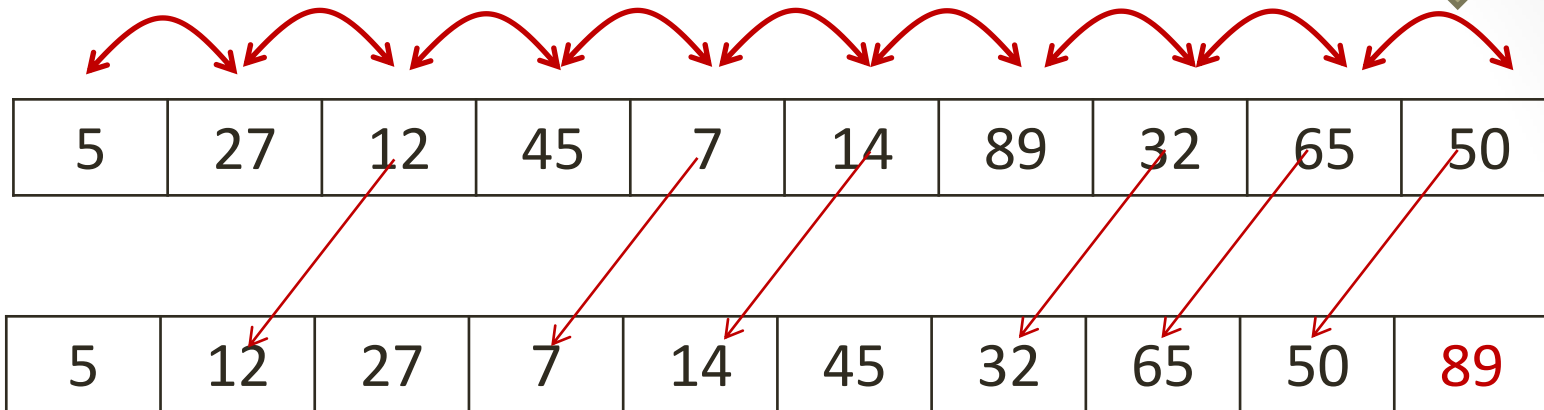
Άσκηση 3.1: Απομάκρυνση πολλαπλών εμφανίσεων

- Γράψτε πρόγραμμα το οποίο να:
- 1) Διαβάζει έναν ακέραιο N από το πληκτρολόγιο
- 2) Δημιουργεί μια λίστα N θέσεων με ψευδοτυχαίους ακεραίους στο διάστημα [1,100] (*υπόδειξη: εφαρμόστε την τεχνική με τη μέθοδο `append`*).
- 3) Ελέγχει αν στη λίστα αυτή υπάρχουν αριθμοί που εμφανίζονται δύο ή περισσότερες φορές και δημιουργεί μια **νέα λίστα** όπου κάθε αριθμός της αρχικής εμφανίζεται μία και μόνον φορά.
- 4) Εμφανίζει στην οθόνη την αρχική λίστα και το πλήθος θέσεων και τα περιεχόμενα της δεύτερης λίστας

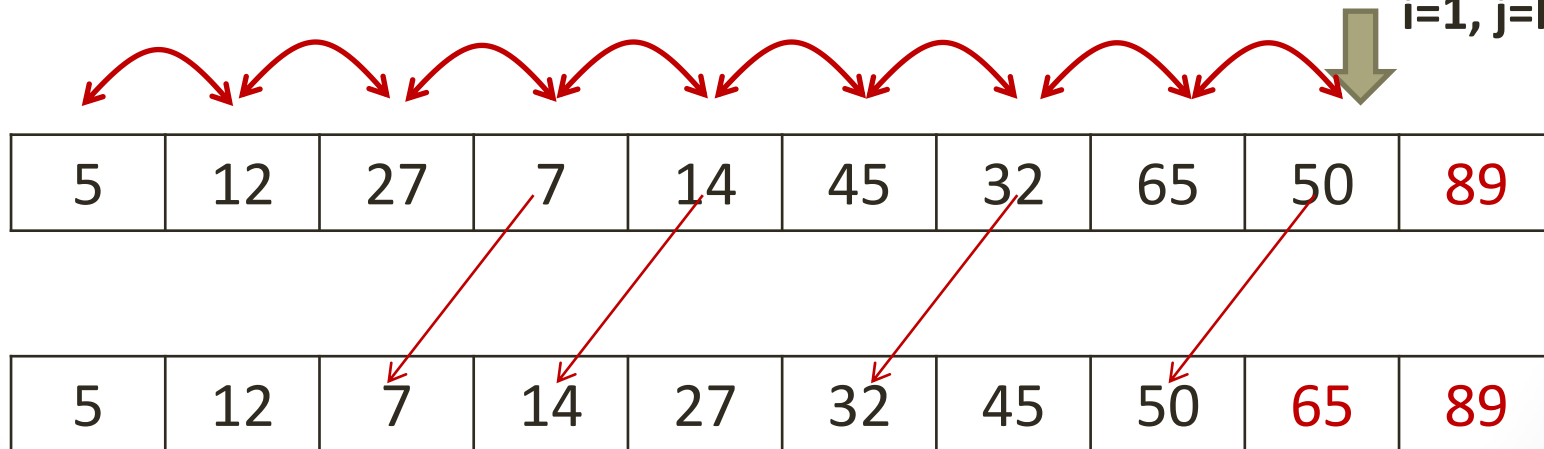
Άσκηση 3.2: Αλγόριθμος φυσαλίδας

- Γράψτε πρόγραμμα το οποίο να:
 - 1) Διαβάζει έναν ακέραιο **N** από το πληκτρολόγιο
 - 2) Δημιουργεί μια λίστα **L**, **N** θέσεων με ψευδοτυχαίους αριθμούς στο διάστημα $[1, 100]$ (υπόδειξη: χρησιμοποιήστε τη μέθοδο περιγραφής λίστας list comprehension).
 - 3) Εφαρμόζει τον αλγόριθμο ταξινόμησης φυσαλίδας για να ταξινομήσει τη λίστα κατ' αύξουσα σειρά
 - 4) Εμφανίζει την ταξινομημένη λίστα στην οθόνη
-
- *Υπόδειξη:* Για λίστα **L** με **N** θέσεις (zero-indexed άρα ο δείκτης του τελευταίου στοιχείου είναι $N-1$), ο ψευδοκώδικας του αλγόριθμου φυσαλίδας δίνεται παρακάτω:
Για i από 0 μέχρι και $N-2$
 Για j από 0 μέχρι και $N-2-i$
 αν $L[j+1] < L[j]$:
 αντιμετάθεσε $L[j]$ με $L[j+1]$

$i=0, j=N-1$



$i=1, j=N-2$



Άσκηση 3.3: Πίνακες

- Χρησιμοποιήστε λίστα λιστών για να υλοποιήσετε τη δομή διδιάστατου τετραγωνικού πίνακα. Το πρόγραμμά σας να λειτουργεί ως εξής:
- A) Διαβάζει από το πληκτρολόγιο έναν ακέραιο αριθμό **N**
- B) Δημιουργεί μια λίστα λιστών με όνομα **L1** η οποία περιλαμβάνει **N** υπολίστες με **N** απλά στοιχεία η καθεμιά. Χρησιμοποιήστε την τεχνική περιγραφής (list comprehension) και γεμίστε τη λίστα με ψευδοτυχαίους ακραίους στο διάστημα $[0,9]$.
- Δ) Στη συνέχεια δημιουργήστε μια λίστα με όνομα **L2** και θέσεις $(N+1) \times (N+1)$. Στις θέσεις $N \times N$ καταχωρήστε και πάλι τους αριθμούς της λίστας **L1**, στην πρόσθετη στήλη (τελευταία) καταχωρήστε το άθροισμα της κάθε σειράς ενώ στην πρόσθετη σειρά (τελευταία) καταχωρήστε το άθροισμα της κάθε στήλης. Τέλος στη θέση $[N+1, N+1]$ καταχωρήστε το άθροισμα των στοιχείων της κύριας διαγωνίου.
- Δ) Εμφανίστε τη λίστα **L2** στην οθόνη σε διάταξη τετραγωνικού πίνακα (matrix) με μορφή $(N+1) \times (N+1)$. Παρακάτω βλέπετε ένα παράδειγμα για $N=3$:

L1

```
[6, 5, 8]  
[5, 4, 4]  
[6, 5, 7]
```

L2

```
[6, 5, 8, 19]  
[5, 4, 4, 13]  
[6, 5, 7, 18]  
[17, 14, 19, 17]
```

Άσκηση 3.4: Ουρά διπλής εισόδου

- Γράψτε πρόγραμμα το οποίο να διαχειρίζεται μια **ουρά διπλής εισόδου** (double-ended queue ή deque) ως εξής:
- Α) Το πρόγραμμα εκτελείται συνεχώς και δέχεται στην είσοδο κάποιο αριθμητικό δεδομένο το οποίο εισάγεται στην ουρά. Στη συνέχεια το πρόγραμμα παρουσιάζει στην οθόνη τη νέα μορφή της ουράς και περιμένει το επόμενο δεδομένο. Όταν ο χρήστης πληκτρολογήσει 'q' το πρόγραμμα τερματίζει την εκτέλεσή του.
- Β) *Είσοδος δεδομένων στην ουρά (enqueue)*: Εφόσον στην είσοδο δίνονται αριθμητικά δεδομένα με τη συνήθη δεκαδική μορφή (πχ. 15, 245, 7, κλπ.) αυτά εισάγονται στο τέλος της ουράς. Εάν δοθεί δεδομένο όπου προτάσσεται το 0 (πχ. 015, 0245, 07) τότε το πρόγραμμα εισάγει το δεδομένο στην αρχή της ουράς (αφού αφαιρέσει τον χαρακτήρα '0')
- Γ) *Έξοδος δεδομένων από την ουρά (dequeue)*: Εάν δοθεί 'r' (από το remove) εξάγεται δεδομένο από το τέλος της ουράς. Εάν δοθεί '0r' εξάγεται δεδομένο από την αρχή της ουράς.