ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Εισαγωγή στους Υπολογιστές

Ενότητα 5: Ακολουθιακές δομές, Αρχεία και μόνιμη αποθήκευση, δυναμικές δομές

Αβούρης Νικόλαος

Πολυτεχνική Σχολή

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Σκοποί ενότητας

- Ακολουθιακές δομές, Αρχεία και μόνιμη αποθήκευση, δυναμικές δομές

Περιεχόμενα ενότητας

- Ακολουθιακές δομές, Αρχεία και μόνιμη αποθήκευση, δυναμικές δομές

Ακολουθιακές δομές, Αρχεία και μόνιμη αποθήκευση, δυναμικές δομές

Ακολουθίες δεδομένωνστην Python (sequences)

- **strings,**
- Unicode strings,
- buffers,
- **Lists**
- **Tuples**
- **Dictionaries**
- xrange objects

Strings : αλφαριθμητικά (σειρές χαρακτήρων ή συμβολοσειρές)

Escape characters\

```
>>> text = “του είπαν\"γεια\" "
>>> print text
του είπαν "γεια «
```

```
>>> title="Αγαμέμνων"
>>> title[1]
'\xe3'
>>> print title[1]
γ
>>>
```

Α	γ	α	μ	έ	μ	ν	ω	ν
0	1	2	3	4	5	6	7	8

strings

κ	ά	τ	ω		σ	τ	ο	υ	ς		π	έ	ρ	α	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

b = “κάτω στους πέρα “

b [5:10] = “στους“‘slice από 5 μέχρι 10 χωρίς το 10

b [:3] = “κατ”; b[11:]=“πέρα “

m = b.find(“στ”) ‘το m γίνεται 5

(-1 αν δεν το βρει)

s.find(substring)

κ	ά	τ	ω		σ	τ	ο	υ	ς		π	έ	ρ	α	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

b = “κάτω στους πέρα “

m = b.find(“στους“)

το m γίνεται 5(-1 αν δεν το βρει)

m = b.find("τ", 4) ψάξε από 4 και μετά

s.split(διαχωριστικό)

```
>>> date="10-12-2015"
>>> d=date.split("-")
>>> d
['10', '12', '2015']
```

(η προκαθορισμένη τιμή τουδιαχωριστικού είναι το κενό)

s.isdigit() s.isalpha()

S=“καλημέρα123“

S[3].isdigit()

False

S[2].isalpha()

True

Τελεστές in και not in

```
names=“Γιώργος Κώστας Τάκης“
if “Τάκης" in names :
    print “υπάρχει ο Τάκης“
```

Πώς αλλάζουμε ένα sting?

```
>>> s = "andrew"
>>> s[0] = "A"
Traceback(most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
?
>>> s = "A" + s[1:]
>>> s
'Andrew‘9
```

Lists στην Python [a, b, c]

ages = [12, ‘Γιώργος’, 157]

print ages[0] #prints **12**

ages[1] = -5

#αντικαθιστά το περιεχόμενο της θέσης 1 με την τιμή**–5;**

#agesπεριέχει[12, **-5, 157]**

ages[2] = ages[2] –1

#μειώνει την τιμή στη θέση2 κατά1;

#agesπεριέχει[12, -5, **156]**

Lists

hives = [16, 2, 105]

numHives = **len(hives)**

#περιέχει το μήκος τηςλίστας hives

#numHives == 3

hives[**numHives] = 40** **#ERROR!**

16	**2**	**105**
0	1	2

Προσθήκη στοιχείου στο τέλος της λίστας: μέθοδος append

```
numbers = [ ]              #creates an empty list
numbers.append(47)         #numbers == [47]
numbers.append(10)         #numbers == [47,10]
numbers.append(-3)         #numbers == [47,10,-3]
```

Προσθήκη στοιχείου σε θέση i μέθοδος insert

- **insert(i, x**) εισάγει το στοιχείο x στη θέση i.
- **a.insert(0, x)** εισάγει το στοιχείο x στην αρχή της λίστας,
- **a.insert(len(a), x)** εισάγει το στοιχείο x στο τέλος της λίστας (ισοδύναμη με την a.append(x).

Πώς εισάγουμε 200ονόματα φοιτητών σε μια λίστα;

```
classList = []
print ( ‘ δώσε ονόματα:’ )
for i in range(200):
        name = raw_input( ‘όνομα#’ + str(i+1) )
        classList.append(name)
```

Πώς διαγράφουμε ένα στοιχείο από τη λίστα; (μέθοδοι pop και remove)

- classList.**pop(3)**

  Διαγράφει το στοιχείο #3 της λίστας

- classList.pop()

  Διαγράφει το τελευταίο στοιχείο

- classList.**remove("nikos")**

  Διαγράφει το πρώτο στοιχείο της λίστας που έχει αυτή την τιμή

Επεξεργασία μιας λίστας με βρόχο for

```
names = [“nikos”, “george”, “kostas”]
for i in range(len(names)):
        print "Name " + str(i+1) + ": " + names[i]
        #prints:
        Name 1: nikos
        Name 2: george
        Name 3: kostas
```

Άσκηση: να γράψετε συνάρτηση searchList που βρίσκει τη θέση ενός στοιχείου στη λίστα

```
numbers = [3,7,2,8]
    index1 = searchList(numbers, 8)      #index1 == 3
    index2 = searchList(numbers, 14)     #index2 == -1
```

(αν δεν το βρει να επιστρέφει -1)

```
def searchList(list, item):
    for i in range(len(list)):
        if list[i]==item:                #found it!
            return i                     #at index i!
            #έξοδος από δω
    return –1
        #δεν βρέθηκε
```

Slices σε λίστες

```
>>> li
['a', 'b', ‘test', 'z', 'example']
>>> li[:3]
['a', 'b', ‘test']
>>> li[3:]
['z', 'example']
>>> li[:]
['a', 'b', ‘test', 'z', 'example']
>>> li[-1]        # το τελευταίο στοιχείο
['example‘]
```

Άσκηση1. Ποιο το αποτέλεσμα:
li[:n]+li[n:]

Άσκηση2: li[1:-1]

Η μαθηματική έννοια του πίνακα μπορεί να υλοποιηθεί με list

Πώς ορίζεται ένας δισδιάστατος πίνακας;

list = [[2,6,4,7], [1,0,3,2]]

print **list[0][2]** **#prints row 0, column 2 == 4**

print **list[0]** **#prints row 0 == [2,6,4,7]**

	0	1	2	3
0	**2**	**6**	**4**	**7**
1	**1**	**0**	**3**	**2**

Δισδιάστατη λίστα

```
list = [ [2,6,4], [1,0,3] ]
for i in range(2):
    for j in range(3):
        print list[i][j],
    print        #αρχίζει την επόμενη γραμμή
    #τυπώνει:
    2 6 4
    1 0 3
```

Πράξεις σε ακολουθίες

τελεστής	**αποτέλεσμα**
<seq> + <seq>	σύνδεση
<seq> * <int-expr>	επανάληψη
<seq>[]	δείκτης
len(<seq>)	Μήκος ακολουθίας
<seq>[:]	Slicing
for <var> in <seq>:	επανάληψη
<expr> in <seq>	συμμετοχή (Boolean)

Μέθοδοι για λίστες (1/2)

- **append(x)** Add an item to the end of the list; equivalent to a[len(a):] = [x].
- **extend(L)** Extend the list by appending all the items in the given list; equivalent to a[len(a):] = L.
- **insert(i, x)** Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).
- **remove(x)** Remove the first item from the list whose value is x. It is an error if there is no such item.
- **pop([I])** Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the iin the method signature denote that the parameter is optional, not that you should type square brackets at that position.

Μέθοδοι για λίστες (2/2)

- **index(x)** Return the index in the list of the first item whose value is x. It is an error if there is no such item.
- **count(x)** Return the number of times xappears in the list.
- **sort()** Sort the items of the list, in place.
- **reverse()** Reverse the elements of the list, in place.

Εφαρμογή φίλτρου σε λίστα

LI2=**filter**(συνάρτηση, LI1)

Επιστρέφει μια νέα λίστα LI2 που προκύπτει από την LI1με την εφαρμογή της συνάρτησης σε όλα τα στοιχεία της

Άσκηση: φίλτρα σε λίστες

- Γράψτε πρόγραμμα Python που καθαρίζει μια λίστα ακεραίων από τους ζυγούς αριθμούς
- Υπο-ερώτημα: γράψτε μια συνάρτηση odd(n) που επιστρέφει True αν ο n είναι μονός και False αν είναι ζυγός

Άσκηση: φίλτρα σε λίστες, λύση

```
>>> def odd(n):
... return n % 2
>>> li = [1, 2, 3, 5, 9, 10, 256, -3]
>>> filter(odd, li)
[1, 3, 5, 9, -3]
>>> filteredList = []
>>> for n in li:
... if odd(n):
... filteredList.append(n)
...
>>> filteredList
[1, 3, 5, 9, -3]
```

Πλειάδες (tuples)

- Οι tuples μοιάζουν με τις λίστες. Ορίζονται σαν ακολουθίες στοιχείων χωρισμένων με κόμματα. Η πρόσβαση σε αυτά είναι γρηγορότερη, αλλά δεν μπορούμε να προσθέσουμε ή να διαγράψουμε στοιχεία.

```
>>> my_tuple= 1,2,3,4
>>> my_tuple[1:3]
(2,3)
```

Dictionaries

d={κλειδί: τιμή, …}

```
phonebook = {
"Alice": 6943456789,
"Boris": 6934456789,
"Doris": 6987345678 }
print phonebook["Alice"]
6943456789
```

Dictionaries : κλειδιά

Μοναδικά κλειδιά. Μπορεί να είναι αριθμοί, string, tuples όχι list, dictionary

```
atomic_number_to_name a= {
1: "hydrogen",
6: "carbon",
7: "nitrogen",
8: "oxygen",
}
nobel_prize_winners= {
(1979, "physics"): ["Glashow", "Salam", "Weinberg"],
(1962, "chemistry"): ["Hodgkin"],
(1984, "biology"): ["McClintock"],
}
```

Dictionaries

```
a= { 1: "hydrogen", 6: "carbon", 7:
  "nitrogen",8: "oxygen"}
a.keys()
[8, 1, 6, 7]
a.values()
['oxygen', 'hydrogen', 'carbon',
  'nitrogen']
a.items()
[(8, 'oxygen'), (1, 'hydrogen'), (6,
  'carbon'), (7, 'nitrogen')]
```

Dictionaries

```
a= { 1: "hydrogen", 6: "carbon",
  7: "nitrogen",8: "oxygen"}


a.update( {"P": "phosphorous",
  "S": "sulfur"} )
a["P‟] = "phosphorous"
del a['C']
```

Άσκηση: Να μεταφερθεί ο αλγόριθμος σε python

```
for  ( I = 2 έως 10 )
    J ← 10
    While ( J >= I )
            { if      A[J-1] > A[J]
              then {  X ← A[J-1]
                      A[J-1] ← A[J]
                      A [ J] ← X  }
              J ← J -1 }
```

- Αν ο πίνακας Α περιέχει τα στοιχεία **(5, 3, 2, 1, 4, 6, 7, 8, 0, -1)**, ποιο θα είναι το περιεχόμενό του μετά την εκτέλεση του αλγορίθμου της άσκησης;

Άσκηση

- Να γράψετε σε Python μια συνάρτηση η οποία δέχεται ως όρισμα μια λίστα και επιστρέφει την ίδια λίστα στην οποία έχουν διαγραφεί τα διπλά στοιχεία

Modules

- Οι πιο χρήσιμες συναρτήσεις και κλάσεις τοποθετούνται σε βιβλιοθήκες (modules), που στην πραγματικότητα είναι αρχεία κειμένου που περιέχουν κώδικα.
- Για να χρησιμοποιηθεί ένα module πρέπει να γίνει import.

```
import string
x = string.split(y)
```

(Κάνουμε import το module και καλούμε τη συνάρτηση

που θέλουμε

όνομα_module.όνομα_συνάρτησης)

Modules

- **import X** εισάγει το module X, και δημιουργεί αναφορά στο module στο τρέχον namespace. Μπορεί να χρησιμοποιηθεί το X.name για αναφορά στο X.
- **from X import *** εισάγει το module X, δημιουργεί αναφορά στο namespace όλων των δημόσιων αντικειμένων του Module. Μπορεί να γίνει αναφορά σε αυτά με απλή χρήση του name στο module X.
- **from string import ***

Εξαιρέσεις

- Μερικές λειτουργίες (όπως η διαίρεση με το μηδέν ή η ανάγνωση από ένα αρχείο που δεν υπάρχει) παράγουν μια συνθήκη σφάλματος ή **εξαίρεση.** Τότε το πρόγραμμα τελειώνει και εκτυπώνει ένα μήνυμα σφάλματος.
- Μπορούμε να το αποφύγουμε αυτό με την δομή **try/except**–πρόταση.

Πρόταση try-except

```
>>> def divide(x, y):
... try:
...     result = x / y
... except ZeroDivisionError:
...     print "division by zero!"
... else:
...     print "result is", result
... finally:
...     print "executing finally clause"
```

- Πρώτα εκτελείται η εντολή **try.**
- Αν δεν υπάρχει κανένα σφάλμα η εντολή **except** παραλείπεται και εκτελείται η **else.**
- Τέλος είτε υπάρχει εξαίρεση είτε όχι πριν τελειώσει η try εκτελείται η **finally.**

Try/except για έλεγχο εισόδου

```
# -*-coding: utf-8 -*-
while True:
   try:
       x = int(raw_input("Δώστε ένα ακέραιο: "))
       break
   except ValueError:
       print"Παρακαλώ! ακέραιο αριθμό..."
```

Άσκηση πώς ελέγχουμε το χρόνο εκτέλεσης (φ5);

```
while True:
    try:
        n=raw_input ()
        n= int (n)
        start=time.time ()
        print nth_prime (n)
        t= str (time.time () -start)
        print ("time= %s seconds" % t)
    except ValueError:
        if n.upper () == "END":
            break
```

Αρχεία

- Χώροι μόνιμης φύλαξης δεδομένων
- Τρόποι προσπέλασης

  Σειριακή προσπέλαση: Απαιτεί μεγάλο χρόνο αναζήτησης

  Τυχαία προσπέλαση: Απαιτεί την ύπαρξη κλειδιού, αρχείου ευρετηρίου (index file)

Αρχεία

- Αρχεία κατακερματισμού**(hashing functions**)

  -Τρόπος άμεσης προσπέλασης

  -Μεγάλη ταχύτητα αναζήτησης

  -Αυξημένη πολυπλοκότητα υλοποίησης

Αρχεία στην Python

```
f1=open(filename,"r")ή f1=file(filename,”r”)
text = f1.readline() #διαβάζει μια γραμμή
  ενός αρχείου κειμένου
text = f1.readlines() #διαβάζει όλες τις
  γραμμές ενός αρχείου κειμένου
f1.close()
Επίσης
f2.write(“hello world”)
```

File buffer

```
f= open(file, “r”)
for line in f:
  print line
```

This is the first line of the file

Second line of the file

With-expression in files

```
with open("x.txt") as f:
  data = f.read()
  do something with data
```

Open modes

Modes	Description
r	Opens a file for reading only (default)
rb	Opens a file for reading only in binary format.
r+	Opens a file for both reading and writing.
rb+	Opens a file for both reading and writing in binary format.
w	Opens a file for writing only. Overwrites the file if the file exists. If not exist, creates a new file
wb	Opens a file for writing only in binary format.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists.
wb+	Opens a file for both writing and reading in binary format.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists.
ab	Opens a file for appending in binary format.
a+	Opens a file for both appending and reading.
ab+	Opens a file for both appending and reading in binary format.

Open file try/except

```
>>> fsock = open("/notthere", "r")          ❶
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
IOError: [Errno 2] No such file or directory: '/notthere'
>>> try:
...     fsock = open("/notthere")            ❷
... except IOError:                          ❸
...     print "The file does not exist, exiting gracefully"
... print "This line will always print"      ❹
The file does not exist, exiting gracefully
This line will always print
```

Άσκηση

- (α) Έστω αρχείο που περιέχει ένα αριθμό ανά γραμμή, να γράψετε πρόγραμμα Python που διαβάζει το αρχείο και βρίσκει το πλήθος, άθροισμα και μέσο όρο των αριθμών
- (β) τροποποιήστε την (α) ώστε να καλύπτει αρχεία στα οποία κάθε γραμμή τους έχει πολλούς αριθμούς χωρισμένους με κόμματα (αρχεία csv)

Σημείωση:

eval(“11.2”) επιστρέφει την τιμή 11.2

string.split(“1,2,3”, “,”) επιστρέφει [“1”, “2”, “3”]

Εντολές του λειτουργικού συστήματος

import os	**# posix functions**
os.chdir(path)	#αλλαγή καταλόγου
os.getcwd()	#επιστρέφει τον τρέχοντα κατάλογο
os.listdir(path)	#περιεχόμενο καταλόγου
os.mkdir(name)	#δημιουργεί κατάλογο
os.rename(old, new)	#αλλαγή ονόματος
import os.path	**# posixpath functions**
os.path.isdir(name)	#έλεγχος για κατάλογο
os.path.split(path)	#χωρίζει όνομα αρχείου path
os.path.join(path1, path2)	#δημιουργεί full path

(η os.path χειρίζεται τα ονόματα κατάλληλα για το συγκεκριμένο λειτουργικό σύστημα)

Άσκηση : περιγράψτε τη λειτουργία της συνάρτησης

```
def print_tree(dir_path):
  for name in os.listdir(dir_path):
      full_path = os.path.join(dir_path,
  name)
      print name,
      if os.path.isdir(full_path):
            print_tree(full_path)
print_tree(raw_input('give me dir path: ' ))
```

Πώς λειτουργούν οι δυναμικές Δομές Δεδομένων(μια βόλτα στα εσωτερικά της Python)

- Οι λίστες και τα λεξικά της Python είναι δομές που καταλαμβάνουν δυναμικά μνήμη, ανάλογα με τον αριθμό στοιχείων που έχουν κάθε στιγμή

  -Συνδεδεμένες λίστες

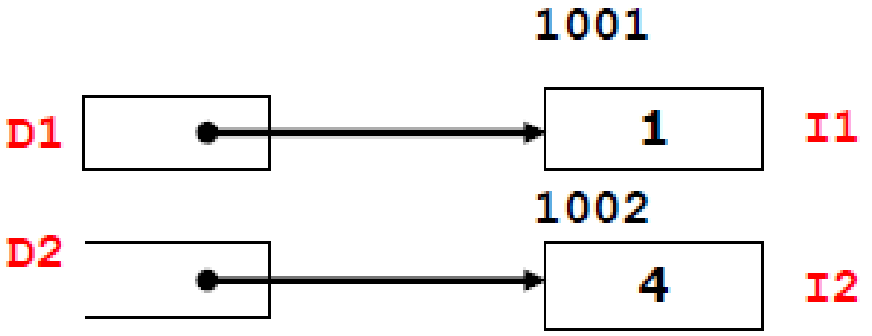
  -ουρές (queues)

  -σωροί (stacks)

  -δένδρα (trees)

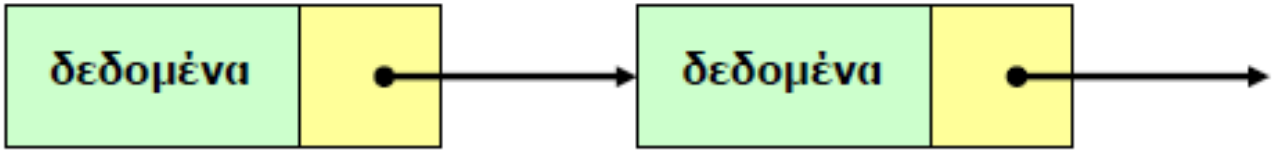
Δείκτες (pointers)

- Μεταβλητές που περιέχουν τη διεύθυνση κάποιας άλλης μεταβλητής
- Βασικό συστατικό εργαλείο για τη δημιουργία δυναμικών δομών δεδομένων

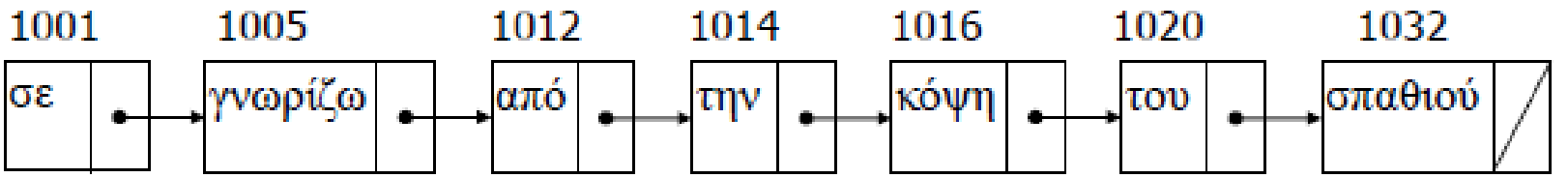


Συνδεδεμένες λίστες

- Δομές που απαρτίζονται από σειρά διατεταγμένων στοιχείων, καθένα από τα οποία εκτός από τα κυρίως δεδομένα περιέχει έναν δείκτη προς το επόμενο στοιχείο της λίστας

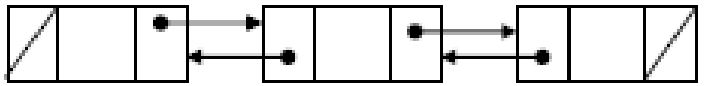


Απλά συνδεδεμένες λίστες

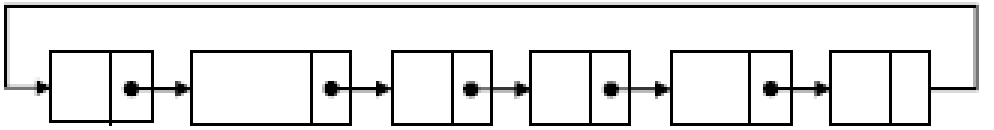


Συνδεδεμένες Λίστες (συνέχεια)

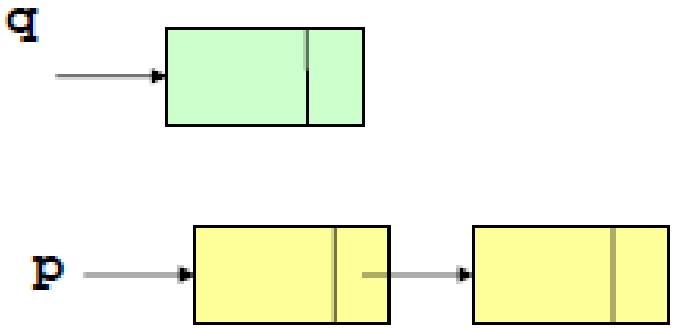
- Διπλά συνδεδεμένες λίστες

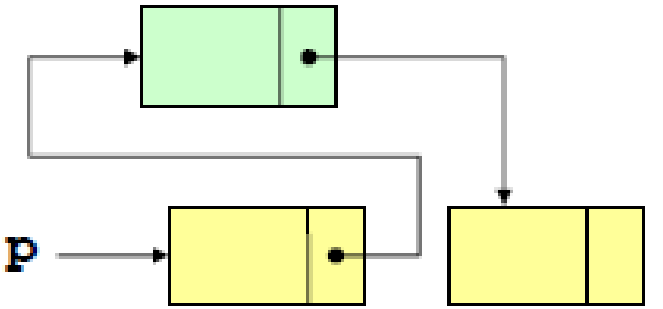


- Κυκλικές λίστες

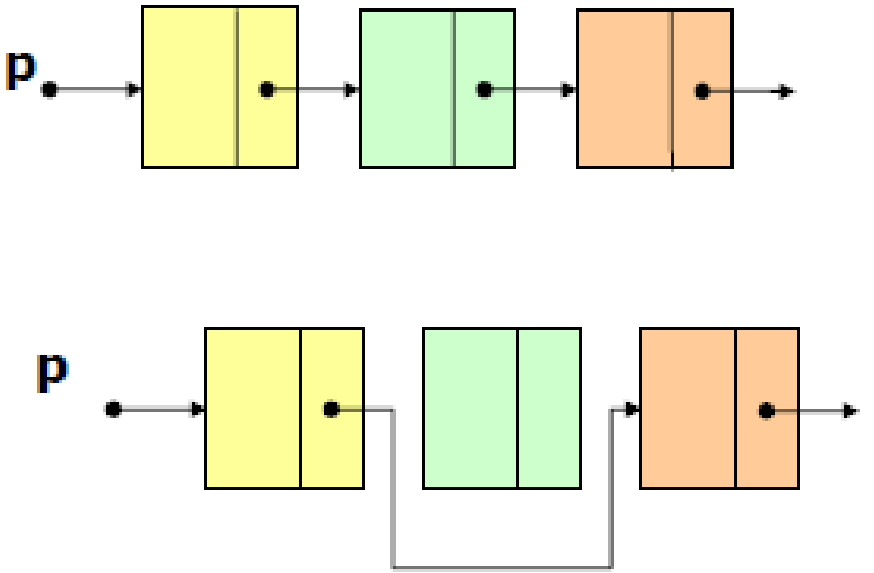


Εισαγωγή στοιχείων σε συνδεδεμένη λίστα





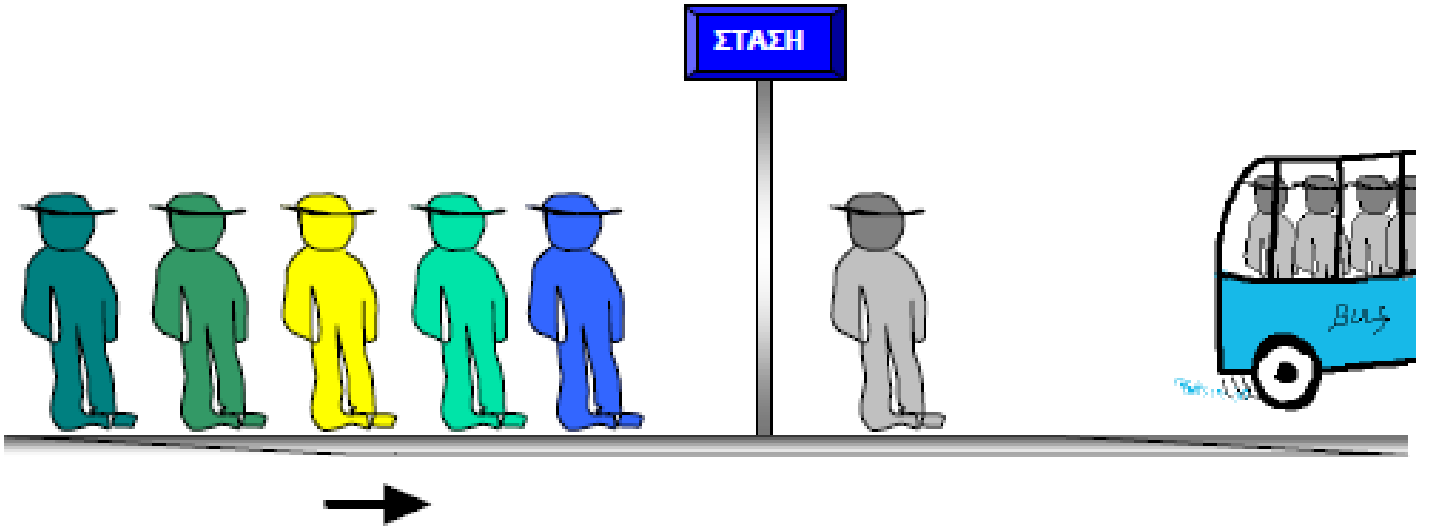
Διαγραφή στοιχείων από συνδεδεμένη λίστα



- Πώς υλοποιείται η εισαγωγή και διαγραφή στοιχείων από λίστα στην Python;

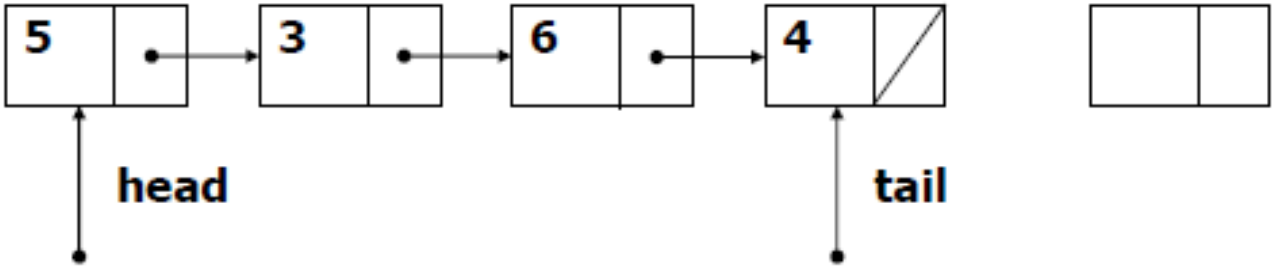
Ουρές (queues)

- First In First Out (FIFO)

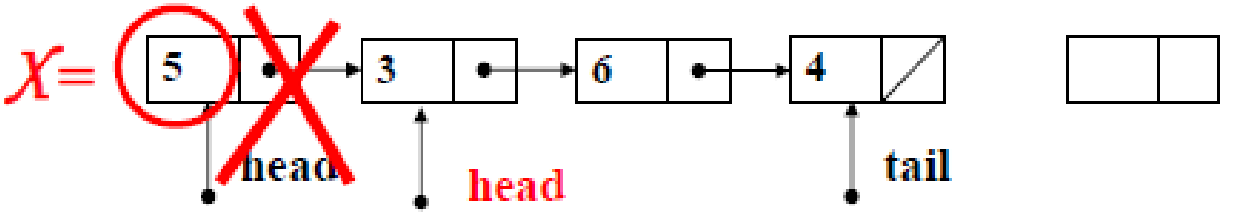


Ουρές (queues)

- Δυναμικές δομές στις οποίες το πρώτο στοιχείο που εισάγεται είναι και το πρώτο που θα εξαχθεί (First In First Out)
- Υλοποίηση με πίνακες ή λίστες



Ανάκληση στοιχείου από ουρά




```
if head=tail
then μήνυμα "άδειαουρά"
else{   X←key(head)
        if(next(head)=NIL
                then head←tail
                else head←next(head)
        }
```

Υλοποίηση ουρών με list της Python

```
>>> queue = ["Eric", "John", "Michael"]
>>> queue.append("Terry") # Terry arrives
>>> queue.append("Graham") # Graham arrives
>>> queue.pop(0) 'Eric'
>>> queue.pop(0) 'John'
>>> queue
['Michael', 'Terry', 'Graham']
```

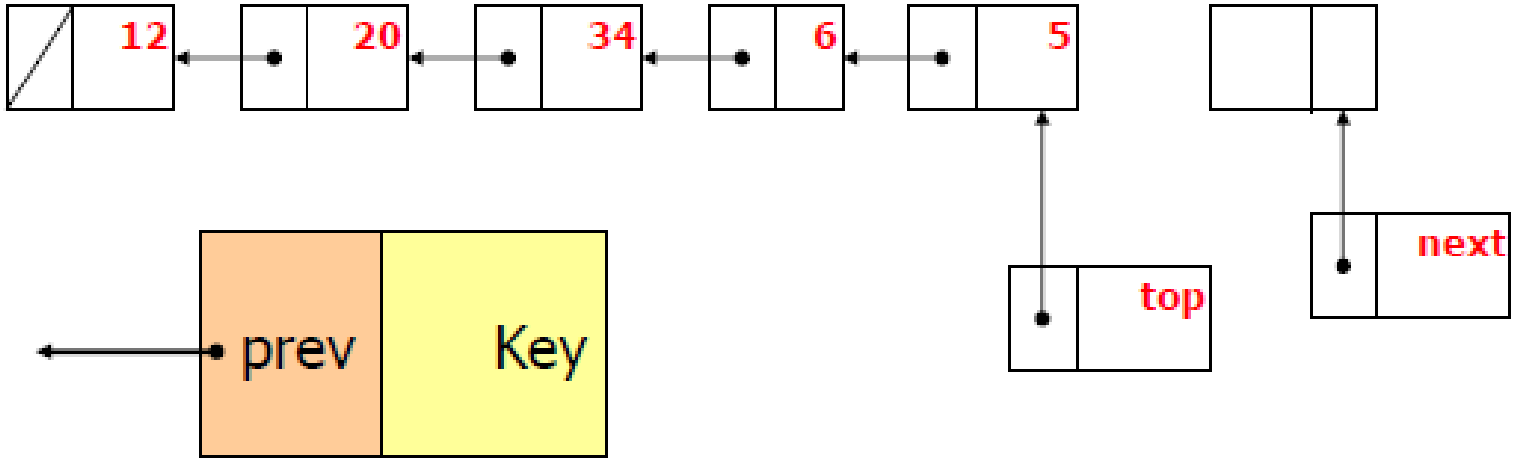
Σωροί (stacks)

- Δυναμικές δομές στις οποίες το τελευταίο στοιχείο που εισάγεται είναι και το πρώτο που θα εξαχθεί

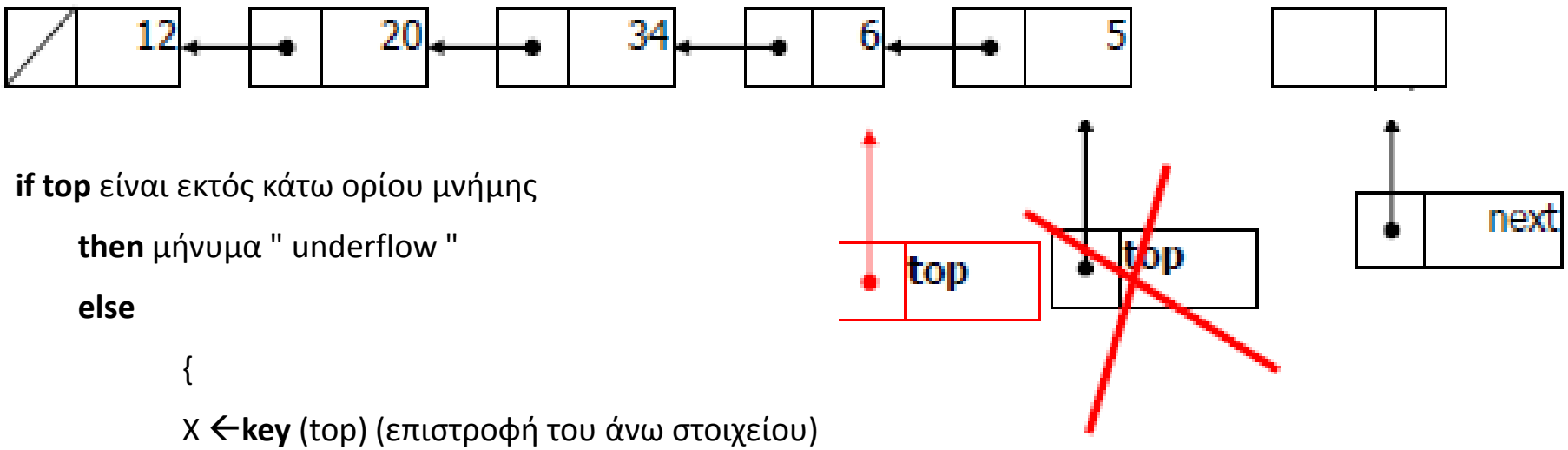
  Last in First Out (LIFO)

Σωροί (stacks)

- Last In First Out
- Υλοποίηση με πίνακες ή λίστες



Ανάκληση στοιχείου από σωρό



if top είναι εκτός κάτω ορίου μνήμης

then μήνυμα " underflow "

else

{

X ←**key** (top) (επιστροφή του άνω στοιχείου)

next ←top (ελευθέρωσε το άνω στοιχείο)

if prev(top) ≠NIL

then top ←prev(top)

else στείλε **μήνυμα "άδειος σωρός"**

}

Υλοποίηση σωρών στην Python

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
```

```
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

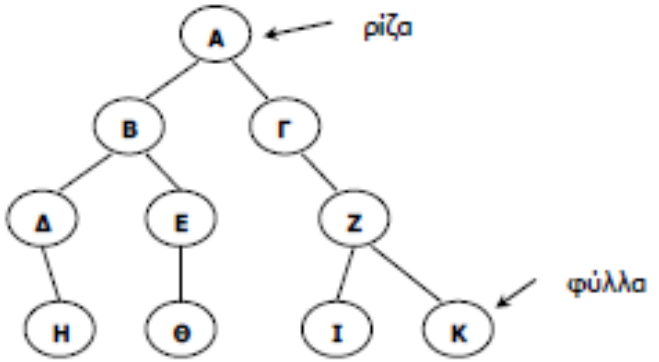
Δέντρα (trees)

- Ιεραρχικές δομές δεδομένων που θυμίζουν τα οικογενειακά δένδρα

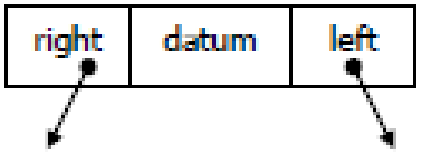
  -Τα στοιχεία του δένδρου λέγονται κόμβοι

  -Ένας κόμβος χωρίς απογόνους λέγεται τερματικός κόμβος ή φύλλο

Δέντρα (συνέχεια)



- **Δυαδικά δέντρα**, κάθε κόμβος έχει το πολύ δύο «παιδιά»



Αναζήτηση σε δυαδικό δένδρο

```
TREE-SEARCH(x,k)
if x=NIL ή k=datum(x)
   then(να επιστραφεί η τιμή k=datum(x))
   else
        if k<datum(x)
                then TREE-SEARCH(left(x),k)
                else TREE-SEARCH(right(x),k)
```

Φροντιστηριακή Άσκηση

- 5.1 Έστω ένας **σωρός** στον οποίο εκτελούνται διαδοχικά οι εντολές push(5), push(3), pop(), push(7), pop(), push(8), push(9), pop(). Να περιγράψετε βήμα προς βήμα το περιεχόμενο του σωρού.
- 5.2 Σε ένα φυλλομετρητή να αναφέρετε ένα παράδειγμα χρήσης λειτουργίας του σωρού
- 5.3 Να περιγράψετε τον αλγόριθμο εισαγωγής στοιχείων σε διπλά συνδεδεμένη λίστα

Άσκηση

- Να επαληθεύσετε την τυχαιότητα της γεννήτριας τυχαίων αριθμών random.

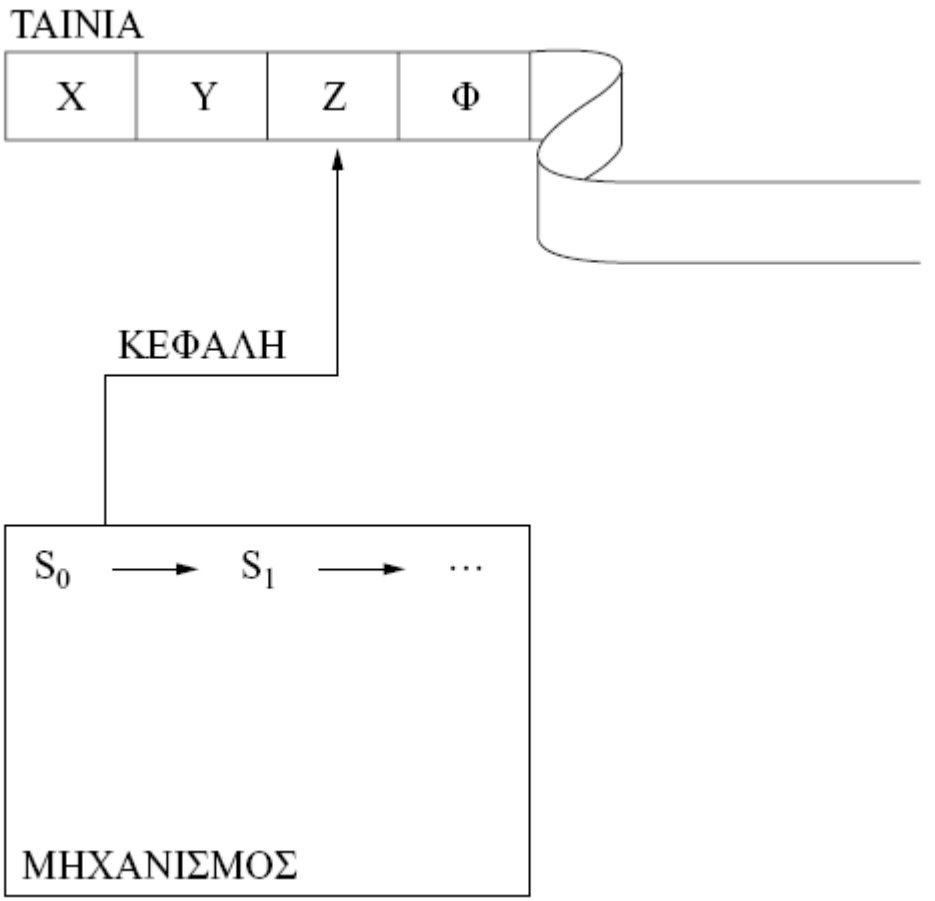
  1. Έστω ότι η random.random() παράγει N τυχαίους αριθμούς στο διάστημα 0..1. Έαν χωρίσουμε το διάστημα σε κ ίσα τμήματα (backets), να αποδείξετε ότι τα πλήθη των τυχαίων αριθμών ανά τμήμα, όσο το N παίρνει μεγαλύτερες τιμές, τείνουν να εξισωθούν. Ελέγξετε την υπόθεση αυτή για διάφορες τιμές του N.

  2. (προαιρετικό ερώτημα) Χρησιμοποιώντας τη γραφική βιβλιοθήκη της Python turtle (from turtle import *) να σχεδιάσετε το ιστόγραμμα πλήθους τιμών στα κ τμήματα. (βλέπε Κεφάλαιο 4 του βιβλίου για τη βιβλιοθήκη turtle)

Πολυπλοκότητα

- Η εκτίμηση του χρόνου εκτέλεσης του αλγορίθμου O(x) order x ως συνάρτηση του μεγέθους του προς επίλυση προβλήματος.
- <u>Παράδειγμα:</u> ποια η πολυπλοκότητα του αλγορίθμου αναζήτησης ενός ονόματος σε μια ταξινομημένη λίστα με δυαδική αναζήτηση και σειριακή αναζήτηση.

Μοντέλα υπολογιστών : η Μηχανή Turing



Άσκηση

- Να εκφράσετε το θεώρημα μη πληρότητας του Goedel και το θεώρημα της υπολογησιμότητας του Turing. Μελετήστε τις συνέπειες των θεωρημάτων αυτών σε θεωρία αλγορίθμων.

Τέλος Ενότητας

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.

Έχουν προηγηθεί οι κάτωθι εκδόσεις:

- Έκδοση 1.0 διαθέσιμη εδώ.

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Σημείωμα Χρήσης Έργων Τρίτων

Το Έργο αυτό κάνει χρήση των ακόλουθων έργων:

Εικόνες/Σχήματα/Διαγράμματα/Φωτογραφίες

Διαφάνειες 42, 49: χρήση του Python Editor IDLE