ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

# Εισαγωγή στους Υπολογιστές

**Εργαστήριο 6**

Καθηγητές: Αβούρης Νικόλαος, Παλιουράς Βασίλης, Κουκιάς Μιχαήλ, Σγάρμπας Κυριάκος

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

# Εργαστήριο 6: Τρίτη Άσκηση Προγραμματισμού Python

## 6.1 Γενικά

Στην άσκηση αυτή θα εξεταστούν μερικές πιο **σύνθετες δομές δεδομένων** της Python,συγκεκριμένα οι **λίστες (lists)** και τα **λεξικά (dictionaries)** και θα χρησιμοποιηθούν σε παραδείγματα παραγωγής ήχων και μουσικής. Θα γίνει επίσης αναφορά στις **μεθόδους (methods)** και θα εξηγηθεί ο συσχετισμός τους με τις συναρτήσεις.

## 6.2 Λίστες

Στην προηγούμενη άσκηση είδαμε για πρώτη φορά τη χρήση λιστών. Οι λίστες είναι **δομές δεδομένων** οι οποίες μπορούν να περιέχουν οποιοδήποτε πλήθος στοιχείων, για παράδειγμα:

```
>>> a=[14,17,11,22,19,11,33]
>>> print a
[14, 17, 11, 22, 19, 11, 33]
```

Από το παραπάνω παράδειγμα είναι εμφανές το πλεονέκτημα που αποκομίζουμε όταν τοποθετούμε τα δεδομένα μας σε δομές δεδομένων, όπως οι λίστες: μπορούμε να τα φυλάξουμε όλα μαζί σε μια μεταβλητή και μετά να τα χειριστούμε κατά βούληση.

Προσέξτε ότι σε μια λίστα μπορεί να επαναλαμβάνονται κάποια στοιχεία. Επιπλέον, τα στοιχεία μιας λίστας δεν είναι απαραίτητο να είναι του ίδιου τύπου. Μπορεί σε μία λίστα κάποια στοιχεία να είναι αριθμοί, κάποια αλφαριθμητικά, κάποια άλλα λίστες, κλπ. Πχ.:

```
>>> q=[34, 'hello', -12.17, ['a', 'b']]
>>> print q
[34, 'hello', -12.17, ['a', 'b']]
```

Η λίστα q περιέχει τέσσερα στοιχεία: τον ακέραιο 34, το αλφαριθμητικό 'hello', τον πραγματικό αριθμό -12.17 και τη λίστα ['a', 'b'].

Το μήκος μιας λίστας επιστρέφει η συνάρτηση len():

```
>>> print len(a), len(q)
74
```

Μπορούμε να επιλέξουμε κάποιο συγκεκριμένο στοιχείο από μία λίστα δίνοντας το όνομα της λίστας και τη θέση του στοιχείου με τον εξής τρόπο:

```
>>> print a[0], q[1]
14 hello
```

Το 1ο στοιχείο μιας λίστας θεωρείται ότι βρίσκεται στη θέση 0, το 2ο στη θέση 1, κ.ο.κ. Μπορούμε να επιλέξουμε και αρίθμηση από το τέλος, αν χρησιμοποιήσουμε αρνητικούς αριθμούς: το -1 αντιστοιχεί στην τελευταία θέση, το -2 στην προτελευταία, κ.ο.κ. Π.χ.:

```
>>> print a[-2], q[-1]
11 ['a', 'b']
```

Μπορούμε ακόμα να επιλέξουμε τμήμα λίστας (υπολίστα) δίνοντας τις θέσεις των ακραίων στοιχείων της ως εξής:

```
>>> print a, a[2:5]
[14, 17, 11, 22, 19, 11, 33] [11, 22, 19]
```

Ενώνουμε δύο λίστες με τον σύμβολο (τελεστή) +:

```
>>> a=[1,2]
>>> b=[34,56]
>>> c=a+b
>>> print c
[1, 2, 34, 56
```

Ενώ ο τελεστής * επαναλαμβάνει μια λίστα όσες φορές ορίσουμε (εδώ 3):

```
>>> d=a*3
>>> print d
[1, 2, 1, 2, 1, 2]
```

Οι λίστες έχουν και άλλες πολλές δυνατότητες, κάποιες από τις οποίες θα δούμε αργότερα, όμως προς το παρόν μπορούμε να εφαρμόσουμε όσα γνωρίζουμε για τις λίστες στην παραγωγή ήχων με τον υπολογιστή μας.

## 6.3 Μουσική με Λίστες

Όπως και στα γραφικά, έτσι και στη μουσική χρειάζεται να φορτώσουμε μια βιβλιοθήκη (module) συναρτήσεων. Και πάλι, υπάρχουν πολλές σχετικές βιβλιοθήκες και συναρτήσεις, όμως εμείς θα ασχοληθούμε μόνο με την απλούστερη (λέγεται Beep() ) από τη βιβλιοθήκη winsound η οποία είναι άμεσα διαθέσιμη από την βασική εγκατάσταση της Python στα Windows:

```
>>> from winsound import Beep
```

Μια που χρειαστήκαμε όχι όλη τη βιβλιοθήκη αλλά μόνο μια συνάρτηση, με την εντολή import εισάγαμε μόνο αυτή τη συνάρτηση, που έχει το όνομα Beep. Προσέξτε ότι τη γράφουμε με Β κεφαλαίο και όχι πεζό. Η Python κάνει διάκριση μεταξύ πεζών και κεφαλαίων χαρακτήρων και για αυτό προκειμένου να αναφερθούμε σωστά σε μία συνάρτηση (ή σε μια μεταβλητή) πρέπει να γράφουμε το όνομά της ακριβώς όπως έχει οριστεί. Επίσης πρέπει να τονίσουμε ότι η βιβλιοθήκη winsound είναι διαθέσιμη μόνο σε συστήματα Windows και μάλιστα δε δουλεύει το ίδιο καλά σε όλες τις εκδόσεις των Windows. Αν τυχόν δε δουλέψει στο σύστημά σας δε θα ακούτε τους ήχους της άσκησης, όμως θα καταλάβετε πως λειτουργεί ο κώδικας των παραδειγμάτων γιατί εκτός από τους ήχους θα φροντίσουμε να τυπώνουμε κατάλληλα μηνύματα στην οθόνη. Ας δοκιμάσουμε αν δουλεύει σωστά:

```
>>>  Beep(1200,500)
```

Αν όλα πήγαν καλά θα πρέπει να ακούσουμε έναν ήχο από το μεγάφωνο του υπολογιστή μας. Ο ήχος θα έχει συχνότητα 1200Hz και διάρκεια μισό δευτερόλεπτο (500msec). Το πρώτο όρισμα στη συνάρτηση Beep είναι ένας ακέραιος στο διάστημα [37, 32767] που δηλώνει τη συχνότητα σε Hertz του ήχου που θα παραχθεί. Το δεύτερο όρισμα δηλώνει τη χρονική διάρκεια του ήχου σε milliseconds. Δυνατότητα να καθορίσουμε την ένταση του ήχου ή να συνδυάσουμε ήχους σε συγχορδίες δεν παρέχει η συνάρτηση Beep, αλλά ακόμα κι έτσι μπορούμε να κάνουμε αρκετά πράγματα. Πχ.:

```
>>> from random import *
>>> for i in range(20):
        freq=randint(1200,20000)
        print freq,"Hz"
        Beep(freq,200)

10594 Hz
4543 Hz
15006 Hz
...
14551 Hz
6327 Hz
9590 Hz
```

Βέβαια αυτό απέχει πολύ από το να μπορεί να χαρακτηριστεί "μουσική". Είναι απλώς τυχαίες συχνότητες ίδιας διάρκειας που διαδέχονται η μία την άλλη. Για να πετύχουμε κάτι πλησιέστερο στη μουσική, θα πρέπει τουλάχιστον να επιλέξουμε εκείνες τις συχνότητες που αντιστοιχούν στις νότες της μουσικής. Ο τρόπος επιλογής είναι αλγοριθμικός.

Ως βασική συχνότητα στη μουσική έχει οριστεί η συχνότητα των 440Hz. Αντιστοιχεί στη νότα Λα που βρίσκεται περίπου στη μέση ενός πιάνου. Από αυτή τη συχνότητα προκύπτουν όλες οι υπόλοιπες. Το αμέσως επόμενο Λα έχει ακριβώς διπλάσια συχνότητα (880Hz) και λέμε ότι βρίσκεται μια οκτάβα ψηλότερα. Το αμέσως προηγούμενο έχει ακριβώς μισή συχνότητα (220Hz) και λέμε ότι βρίσκεται μια οκτάβα χαμηλότερα. Να πως ακούγονται:

```
>>> List=[220,440,880]
>>> for freq in List:
        print freq,"Hz"
        Beep(freq,500)
```

Και με την ίδια λογική, συνεχώς διπλασιάζοντας τη συχνότητα πηγαίνουμε σε όλο και υψηλότερες νότες Λα, συνεχώς υποδιπλασιάζοντας πηγαίνουμε σε νότες Λα σε όλο και χαμηλότερες οκτάβες. Να πως μπορούμε να βάλουμε σε μια λίστα όλες τις νότες Λα που μπορεί να παράγει η συνάρτηση Beep():

```
>>> La_List=[]
>>> F=440
>>> while F<=32767:
        La_List=La_List+[F]
        F=F*2
```

Ξεκινήσαμε με μία λίστα (La_List) αρχικά κενή και μία μεταβλητή (F) με αρχική τιμή 440. Στη συνέχεια χρησιμοποιήσαμε μια επαναληπτική διαδικασία με την εντολή while. Την εντολή while είναι η πρώτη φορά που τη χρησιμοποιούμε. Εκτελεί επαναλήψεις όπως η for, μόνο που οι επαναλήψεις της διαρκούν όσο ισχύει η συνθήκη που γράφουμε δίπλα στη while. Εδώ γράψαμε F<=32767, επομένως όσο η F (που τώρα έχει τιμή 440 αλλά προφανώς στη συνέχεια θα αλλάξει) έχει τιμή μικρότερη ή ίση με 32767 οι εσωτερικές εντολές της while θα επαναλαμβάνονται. Και οι εσωτερικές εντολές λένε να προστεθεί η F στο τέλος της λίστας και στη συνέχεια να διπλασιαστεί. Αν τυπώσουμε τη λίστα θα δούμε:

```
>>> print La_List
[440, 880, 1760, 3520, 7040, 14080, 28160]
```

Αυτές είναι οι συχνότητες Λα από 440 Hz και πάνω. Με παρόμοιο τρόπο προσθέτουμε και τις χαμηλότερες οκτάβες:

```
>>> F=440
```

```
>>> while F>=37:
        if not(F in La_List):
                La_List=[F]+La_List
        F=F/2
```

Εδώ για να μην προστεθεί δύο φορές στη λίστα η συχνότητα 440, γίνεται πρώτα ένας έλεγχος if. Αν η F δεν περιέχεται ήδη στη λίστα, τότε μόνο προστίθεται. Προσέξτε επίσης ότι, αντίθετα από ότι προηγουμένως, η F προστίθεται στην αρχή της λίστας. Και τελικά:

```
>>> print La_List
[55, 110, 220, 440, 880, 1760, 3520, 7040, 14080, 28160]
```

Αυτές είναι όλες οι νότες Λα που μπορεί να παράγει η Beep(). Ας τις ακούσουμε:

```
>>> def play_list(L,Duration):
        for freq in L:
                print freq, "Hz"
                Beep(freq,Duration)
```

Επειδή στη συνέχεια θα ακούσουμε πολλές λίστες συχνοτήτων, για να διευκολυνθούμε ορίσαμε μια συνάρτηση που δέχεται ως ορίσματα μια λίστα συχνοτήτων L και μια μεταβλητή Duration και παίζει τις νότες της λίστας με διάρκεια Duration msec την κάθε μια. Και τώρα αρκεί να γράψουμε:

```
>>> play_list(La_List,500)
55 Hz
110 Hz
220 Hz
440 Hz
880 Hz
1760 Hz
3520 Hz
7040 Hz
14080 Hz
28160 Hz
```

Κατά πάσα πιθανότητα την τελευταία νότα δε θα μπορέσετε να την ακούσετε. Το ανθρώπινο αυτί ακούει ήχους περίπου στην περιοχή 20 - 22000 Hz. Ήχοι με συχνότητα πάνω από 22000 Hz λέγονται **υπέρηχοι**. Ήχοι κάτω από 20 Hz λέγονται **υπόηχοι**.

Βέβαια, η μουσική δεν αποτελείται μόνο από νότες Λα. Ανάμεσα σε δύο διαδοχικά πλήκτρα Λα στο πιάνο υπάρχουν άλλα 11 πλήκτρα, δηλαδή κάθε οκτάβα έχει συνολικά 12 διαφορετικές νότες. Με δεδομένο ότι κι αυτές ισαπέχουν μεταξύ τους με λογαριθμικό τρόπο, μπορούμε να υπολογίσουμε τη συχνότητα κάθε επόμενου πλήκτρου από κάθε προηγούμενο, όχι διπλασιάζοντας, αλλά πολλαπλασιάζοντας με τον συντελεστή $\delta = 2^{(1/12)} = 1{,}05946309436$.

Ο συντελεστής αυτός προκύπτει πολύ εύκολα, αν σκεφτούμε ως εξής: Ξεκινώντας από τη νότα Λα = 440 Hz, πολλαπλασιάζοντας με δ, πηγαίνουμε στην επόμενη (Λα# = δ*440). Ξαναπολλαπλασιάζοντας με δ, πηγαίνουμε στη μεθεπόμενη (Σολ = δ*δ*440) κ.ο.κ. Αν πολλαπλασιάσουμε ακριβώς 12 φορές θα έχουμε πέσει πάνω στη Λα της επόμενης οκτάβας που έχει τη διπλάσια συχνότητα από την αρχική. Επομένως $\delta^{12}*440=2*440$ και $\delta = 2^{(1/12)}$. Να η επαλήθευση με κώδικα Python:

```
>>> A=440.
>>> d=2.**(1./12.)
>>> print d
1.05946309436

>>> L=[A]
>>> for i in range(12):
```

```
        A=A*d
        L=L+[A]

>>> print L
[440.0, 466.1637615180899, 493.8833012561241, 523.2511306011974, 554.3652619537443,
 587.3295358348153, 622.253967444162, 659.2551138257401, 698.456462866008, 739.988845423269,
783.9908719634989, 830.6093951598906, 880.0000000000003]
```

Η λίστα L περιέχει όλες τις συχνότητες μεταξύ 440-880 Hz που αντιστοιχούν σε μουσικές νότες, υπολογισμένες με αρκετά καλή ακρίβεια (προσέξτε το τελευταίο στοιχείο που αντιστοιχεί στα 880 Hz). Όμως δεν μπορούμε να τις τροφοδοτήσουμε ακόμα στην Beep(). Η Beep() περιμένει ακέραιους αριθμούς ενώ η λίστα περιέχει πραγματικούς. Θα πρέπει να τους μετατρέψουμε.

Η συνάρτηση που μετατρέπει έναν πραγματικό αριθμό σε ακέραιο λέγεται int():

```
>>> print int(2.4), int(2.5), int(2.6)
2 2 2
```

Η int() ουσιαστικά "κόβει" το δεκαδικό μέρος από τον πραγματικό αριθμό και επιστρέφει μόνο το ακέραιο μέρος του. Έτσι, και το 2.4 και το 2.5 και το 2.6 στρογγυλοποιούνται στο 2. Αν θέλουμε να πετύχουμε στρογγυλοποίηση στον πλησιέστερο ακέραιο, ένα απλό κόλπο είναι να προσθέσουμε 0.5 στον αριθμό που μετατρέπουμε. Πχ.:

```
>>> print int(0.5+2.4), int(0.5+2.5), int(0.5+2.6)
2 3 3
```

Και τώρα ξέρουμε πως να φτιάξουμε από την L μια λίστα ακεραίων:

```
>>> K=[]
>>> for f in L:
        K=K+[int(0.5+f)]

>>> print K
[440, 466, 494, 523, 554, 587, 622, 659, 698, 740, 784, 831, 880]
```

Σημειωτέον ότι μπορούμε να κάνουμε ακριβώς το ίδιο και με έναν ακόμα πιο απλό τρόπο. Η Python επιτρέπει να ορίζουμε λίστες περιγραφικά. Πχ.:

```
>>> M=[int(0.5+x) for x in L]
```

Αυτό λέει ότι η M είναι μία λίστα με στοιχεία που προκύπτουν από τον υπολογισμό int(0.5+x), για κάθε x που βρίσκεται στη λίστα L. Το αποτέλεσμα είναι το ίδιο:

```
>>> print M
[440, 466, 494, 523, 554, 587, 622, 659, 698, 740, 784, 831, 880]
```

Και τώρα μπορούμε να ακούσουμε όλες τις νότες:

```
>>> play_list(M,300)
440 Hz
466 Hz
494 Hz
523 Hz
554 Hz
587 Hz
622 Hz
659 Hz
698 Hz
740 Hz
784 Hz
```

```
831 Hz
880 Hz
```

Με παρόμοιο τρόπο μπορούμε να παράγουμε τις νότες και για τις υπόλοιπες οκτάβες. Ή, εναλλακτικά και πιο εύκολα μπορούμε να χρησιμοποιήσουμε τον περιγραφικό τρόπο ορισμού των λιστών για να πάρουμε την επόμενη ή την προηγούμενη οκτάβα με διπλασιασμό ή υποδιπλασιασμό όλων των στοιχείων:

```
>>> M2=[2*x for x in M]
>>> print M2
[880, 932, 988, 1046, 1108, 1174, 1244, 1318, 1396, 1480, 1568, 1662, 1760]
>>> M0=[x/2 for x in M]
>>> print M0
[220, 233, 247, 261, 277, 293, 311, 329, 349, 370, 392, 415, 440]
```

Και τώρα έχουμε τις νότες από 3 ολόκληρες οκτάβες και μπορούμε αν θέλουμε να τις ενώσουμε όλες σε μια μεγάλη λίστα:

```
>>> Mall=M0+M+M2
>>> print Mall
[220, 233, 247, 261, 277, 293, 311, 329, 349, 370, 392, 415, 440, 440, 466, 494, 523, 554, 587, 622, 659,
698, 740, 784, 831, 880, 880, 932, 988, 1046, 1108, 1174, 1244, 1318, 1396, 1480, 1568, 1662, 1760]
```

Αρκετά καλό αποτέλεσμα, όμως όχι τέλειο. Προσέξτε ότι οι συχνότητες 440 και 880 επαναλαμβάνονται 2 φορές στη λίστα Mall, επειδή κάθε λίστα που συνθέσαμε περιείχε 13 στοιχεία (περιείχε και τη νότα Λα της επόμενης οκτάβας). Αυτό το διορθώνουμε εύκολα, αν θυμηθούμε ότι μπορούμε να επιλέξουμε τμήματα λιστών, όπως είδαμε στην ενότητα 6.2:

```
>>> Mall=M0[:12]+M+M2[1:]
>>> print Mall
[220, 233, 247, 261, 277, 293, 311, 329, 349, 370, 392, 415, 440, 466, 494, 523, 554, 587, 622, 659, 698,
740, 784, 831, 880, 932, 988, 1046, 1108, 1174, 1244, 1318, 1396, 1480, 1568, 1662, 1760]
```

Το M0[:12] είναι συντομογραφία του M0[0:12] και σημαίνει "τα πρώτα 12 στοιχεία της M0". Παρομοίως, η συντομογραφία M2[1:] σημαίνει "τα στοιχεία της M2 από το δεύτερο μέχρι το τελευταίο". Και τώρα μπορούμε να ξαναδοκιμάσουμε την τυχαία σύνθεση μουσικής γράφοντας κάτι τέτοιο:

```
>>> for i in range(20):
        note=randint(0,36)
        freq=Mall[note]
        print "Note=", note, "   Freq=", freq, "Hz"
        Beep(freq,200)

Note= 13    Freq= 466 Hz
Note= 15    Freq= 523 Hz
Note= 21    Freq= 740 Hz
...
Note= 9    Freq= 370 Hz
Note= 24    Freq= 880 Hz
Note= 31    Freq= 1318 Hz
```

Το παραπάνω πρόγραμμα επιλέγει τυχαία νότες από τη λίστα Mall και τις παίζει με τη συνάρτηση Beep(). Και πάλι δεν πρόκειται για ποιοτική μουσική, αλλά ακούγεται κάπως καλύτερα από τον θόρυβο των τελείως τυχαίων συχνοτήτων. Για να πετύχουμε κάτι καλύτερο χρειάζεται να δούμε μια ακόμα δομή δεδομένων που λέγεται **Λεξικό**.

## 6.4 Λεξικά

Το **λεξικό** (**Dictionary**) είναι μία δομή δεδομένων στην οποία καταχωρούμε δεδομένα τα οποία μπορούμε να ανακτήσουμε με το όνομα τους που λέγεται **κλειδί** (**key**). Να πως ορίζεται ένα λεξικό:

```
>>> a={'La':440, 'La#':466, 'Si':494, 'Do':523}
```

Εδώ ορίσαμε ένα λεξικό που αποτελείται από 4 στοιχεία, αυτά που βρίσκονται μεταξύ των αγκυλών { }. Κάθε στοιχείο αποτελείται από ένα ζεύγος "κλειδί : τιμή". Το κλειδί είναι η ετικέτα ή το όνομα του στοιχείου και χωρίζεται από την τιμή με το σύμβολο ":". Το λεξικό καταχωρείται στη μεταβλητή a. Τώρα μπορούμε να ανακτήσουμε τις τιμές των στοιχείων του λεξικού ως εξής:

```
>>> print a['La#']
466
>>> print a['Si'], a['Do']
494 523
```

Αντίθετα με τις λίστες, δε χρειάζεται να γνωρίσουμε τη θέση του στοιχείου για να τυπώσουμε την τιμή του. Αρκεί να γνωρίζουμε το όνομά του. Εξάλλου, η σειρά των στοιχείων δε διατηρείται μέσα σε ένα λεξικό, όπως φαίνεται από την πλήρη εκτύπωσή του:

```
>>> print a
{'Do': 523, 'Si': 494, 'La#': 466, 'La': 440}
```

Νέα στοιχεία μπορούν να προστεθούν στο λεξικό με τον εξής τρόπο:

```
>>> a['Sol']=392
>>> a['Sol#']=415
>>> print a
{'Do': 523, 'La': 440, 'Sol#': 415, 'La#': 466, 'Si': 494, 'Sol': 392}
```

Αν προσθέσουμε ένα στοιχείο με όνομα που ήδη υπάρχει, τότε γίνεται αντικατάσταση:

```
>>> a['Do']=261
>>> print a
{'Do': 261, 'La': 440, 'Sol#': 415, 'La#': 466, 'Si': 494, 'Sol': 392}
```

Για να σβήσουμε ένα στοιχείο χρησιμοποιούμε την εντολή del:

```
>>> del a['Do']
>>> print a
{'La': 440, 'Sol#': 415, 'La#': 466, 'Si': 494, 'Sol': 392}
```

Τελικά, αν βάλουμε σε ένα λεξικό όλες τις νότες μιας οκτάβας:

```
>>> a={'Do':261, 'Do#':277, 'Re':293, 'Re#':311, 'Mi':329, 'Fa':349, 'Fa#':370, 'Sol':392, 'Sol#':415, 'La':440,
'La#':466, 'Si':494}

>>> print a
{'Do': 261, 'Re': 293, 'Do#': 277, 'La': 440, 'Mi': 329, 'Sol': 392, 'Sol#': 415, 'Fa': 349, 'Si': 494, 'Fa#': 370,
'Re#': 311, 'La#': 466}
```

... μπορούμε να προγραμματίσουμε μουσική στον υπολογιστή μας δίνοντας απευθείας τα ονόματα των νοτών. Πχ. να μια μουσική φράση:

```
>>> p=['Do','Mi','Sol','La','La#','La','Sol','Mi']
```

Και να πώς μετατρέπεται αυτόματα σε λίστα συχνοτήτων με τη βοήθεια του λεξικού:

```
>>> f=[a[x] for x in p]
>>> print f
[261, 329, 392, 440, 466, 440, 392, 329]
```

Και τώρα μπορούμε να την ακούσουμε:

```
>>> play_list(f,250)
261 Hz
329 Hz
392 Hz
440 Hz
466 Hz
440 Hz
392 Hz
329 Hz
```

... και πολλές φορές αν θέλουμε:

```
>>> play_list(f*4,250)
261 Hz
329 Hz..
392 Hz
329 Hz
```

Μπορούμε ακόμη να μετατοπίσουμε τη φράση κατά πχ. 5 νότες ψηλότερα:

```
>>> f5=[int(0.5+x*d**5) for x in f]
>>> print f5
[348, 439, 523, 587, 622, 587, 523, 439]
>>> play_list(f5,250)
```

... ή 7 νότες ψηλότερα από την αρχική φράση:

```
>>> f7=[int(0.5+x*d**7) for x in f]
>>> print f7
[391, 493, 587, 659, 698, 659, 587, 493]
>>> play_list(f7,250)
```

Μπορούμε επίσης να συνδυάσουμε την αρχική φράση με τις μετατοπίσεις της γράφοντας μια σύνθεση λιστών ως εξής:

```
>>> play_list(f*2+f5+f+f7+f5+f*2,250)
261 Hz
329 Hz
392 Hz
...
440 Hz
392 Hz
329 Hz
```

Κι όσο για τη σύνθεση τυχαίας μουσικής, ακούγεται καλύτερα αν περιορίσουμε τη λίστα επιλογών σε κάποιο μικρό υποσύνολο των νοτών της οκτάβας (στη μουσική λέγεται **κλίμακα**). Να μερικά παραδείγματα:

```
>>> p1=['Do','Re','Mi','Fa','Sol','La','Si']
>>> f1=[a[x] for x in p1]
>>> for i in range(20):
        Beep(f1[randint(0,6)],250)

>>> p2=['Do','Mi','Sol','La','La#']
>>> f2=[a[x] for x in p2]
```

```
>>> for i in range(20):
        Beep(f2[randint(0,4)],200)

>>> p3=['Do#','Re#','Fa#','Sol#','La#']
>>> f3=[a[x] for x in p3]
>>> for i in range(20):
Beep(f3[randint(0,4)],200)
```

Ουσιαστικά πρόκειται για τον ίδιο κώδικα που τρέχει σε διαφορετικές κλίμακες και κάθε μια ακούγεται λίγο διαφορετικά. Θα μπορούσαμε να τον εξελίξουμε και να τον γενικεύσουμε σε συνάρτηση, ως εξής:

```
>>> def rand_music(KLIM,OCT,Notes,Speed):
        f=[a[x] for x in KLIM]
        q=f
        while OCT>1:
                OCT=OCT-1
                r=[x*2 for x in f]
                q=q+r
        L=len(q)-1
        for i in range(Notes):
                note=randint(0,L)
                freq=q[note]
                Beep(freq,Speed)
```

KLIM είναι η λίστα με την κλίμακα, OCT το πλήθος των οκτάβων στις οποίες θα επαναλαμβάνεται η κλίμακα, Notes το πλήθος των τυχαίων νοτών που θα παιχτούν και Speed η διάρκεια της κάθε νότας. Η κλήση γίνεται ως εξής:

```
>>> rand_music(['Do#','Re#','Fa#','Sol#','La#'],2,20,250)
```

Μέχρι τώρα έχουμε κρατήσει σταθερές τις χρονικές διάρκειες των νοτών. Αν θέλουμε όμως να παίξουμε ακριβέστερη μουσική θα πρέπει για κάθε νότα να προσδιορίσουμε και τη διάρκειά της. Στη μουσική οι χρονικές διάρκειες είναι κβαντισμένες. Αφού καθοριστεί η διάρκεια μιας ολόκληρης νότας σε msec, όλες οι άλλες νότες έχουν διάρκειες υποπολλαπλάσιες δυνάμεις του 2 ως προς την αρχική: 1/2, 1/4, 1/8, 1/16, 1/32 και 1/64. Βολεύει λοιπόν να ορίσουμε τη διάρκεια της ολόκληρης νότας ως ακέραιο πολλαπλάσιο του 64, πχ. 30*64=1920 msec.

Επίσης χρειαζόμαστε και έναν ήχο παύσης (ησυχίας). Άρκεί να προσθέσουμε στο λεξικό έναν υπέρηχο:

```
>>> a['P']=28000
```

Η κωδικοποίηση μιας μουσικής φράσης θα είναι τώρα λίγο πιο σύνθετη:

```
>>> music=[['La',16],['Sol',16],['La',2],['P',8],['Sol',16], ['Fa',16],['Mi',16],['Re',16],['Do#',4],['Re',2]]
```

Κάθε στοιχείο της λίστας είναι μια άλλη λίστα που περιέχει δύο στοιχεία. Το πρώτο είναι το όνομα της νότας, το δεύτερο η χρονική διάρκεια. Να μια συνάρτηση που μετατρέπει σε μουσική την παραπάνω λίστα:

```
>>> def play_musiclist(L):
        for x in L:
                [note,time]=x
                freq=a[note]
                duration=1920/time
                print "Note=", note, "(", freq, "Hz)   Dur= 1/",
                print time, "(", duration, "msec)"
                Beep(freq,duration)

>>> play_musiclist(music)
Note= La ( 440 Hz)    Dur= 1/16 ( 120 msec)
Note= Sol ( 392 Hz)    Dur= 1/16 ( 120 msec)
Note= La ( 440 Hz)    Dur= 1/ 2 ( 960 msec)
```

```
Note= P ( 28000 Hz)    Dur= 1/ 8 ( 240 msec)
Note= Sol ( 392 Hz)    Dur= 1/ 16 ( 120 msec)
Note= Fa ( 349 Hz)    Dur= 1/ 16 ( 120 msec)
Note= Mi ( 329 Hz)    Dur= 1/ 16 ( 120 msec)
Note= Re ( 293 Hz)    Dur= 1/ 16 ( 120 msec)
Note= Do# ( 277 Hz)    Dur= 1/ 4 ( 480 msec)
Note= Re ( 293 Hz)    Dur= 1/ 2 ( 960 msec)
```

## 6.5 Μέθοδοι

Πριν αρχίσουμε να γράφουμε τον κώδικα ενός προγράμματος, μια καλή τακτική είναι να ψάχνουμε στο on-line-help της Python (ή στο εγχειρίδιο αναφοράς το οποίο κατεβάζουμε από το επίσημο web-site) για τυχόν έτοιμες βιβλιοθήκες (modules) ή συναρτήσεις που μπορούμε να χρησιμοποιήσουμε. Οι έτοιμες συναρτήσεις εμφανίζονται με τρεις διαφορετικές συντακτικές μορφές κλήσης:

Κάποιες καλούνται όπως οι συναρτήσεις που ορίζουμε με την εντολή def. Τα ορίσματά τους τα γράφουμε μέσα σε παρένθεση. Μπορεί να επιστρέφουν τιμή ή όχι. Τέτοιες συναρτήσεις είναι για παράδειγμα η sin() και η Beep().

Κάποιες άλλες καλούνται με τη βοήθεια τελεστών. Για παράδειγμα η συνάρτηση που ενώνει δυο λίστες ενεργοποιείται με τον τελεστή + :

```
>>> print [1,2]+[3,4]
[1, 2, 3, 4]
```

Αν δε μας αρέσει αυτός ο τρόπος κλήσης μπορούμε πάντα να ορίσουμε μια δική μας συνάρτηση όπως:

```
>>> def connect(L1,L2):
        return L1+L2
```

Και τώρα κάθε φορά που θέλουμε να ενώσουμε δύο λίστες θα καλούμε τη συνάρτηση connect():

```
>>> print connect([1,2],[3,4])
[1, 2, 3, 4]
```

Τέλος υπάρχει και η τρίτη κατηγορία συναρτήσεων που ονομάζονται **μέθοδοι** και καλούνται ως εξής:

```
>>> L=['a','b','a','c','a','a','d']
>>> print L.count('a')
4
```

Η count() είναι μια μέθοδος που μετράει πόσα από τα στοιχεία της λίστας είναι ίδια με την παράμετρο που δίνουμε. Η ίδια η λίστα δεν μπαίνει ως παράμετρος στην παρένθεση. Γράφουμε απλώς το όνομα της λίστας, μια τελεία και το όνομα της μεθόδου. Αυτός ο τρόπος σύνταξης λέγεται **dot notation** (σημειογραφία με τελεία).

Και πάλι, αν δε μας αρέσει αυτή η σύνταξη κλήσης μπορούμε να ορίσουμε κάτι τέτοιο:

```
>>> def my_count(A,x):
        return A.count(x)
```

... και στη συνέχεια να γράφουμε:

```
>>> print my_count(L,'a')
4
```

Οι λόγοι για τους οποίους υπάρχουν αυτές οι τρεις διαφορετικές κατηγορίες συναρτήσεων με τους διαφορετικούς τρόπους κλήσης αρχίζουν να φαίνονται όταν φτιάχνει κανείς μεγάλα και σύνθετα προγράμματα. Προς το παρόν αρκεί να εξηγήσουμε ότι κάποιες λειτουργίες είναι βολικότερο να συντάσσονται ως τελεστές, άλλες ως μέθοδοι και άλλες ως συναρτήσεις.

Μέθοδοι υπάρχουν όχι μόνο για λίστες αλλά για οποιοδήποτε τύπο δομών δεδομένων που χρησιμοποιεί η Python. Για παράδειγμα η upper() είναι μια μέθοδος που επιδρά σε αλφαριθμητικά (strings) και επιστρέφει το ίδιο αλφαριθμητικό αλλά με κεφαλαία γράμματα.

```
>>> v='αβγδε'
>>> print v.upper()
ΑΒΓΓΔ
```

Οι keys() και values() είναι μέθοδοι που επιδρούν πάνε σε λεξικά και επιστρέφουν λίστες με όλα τα κλειδιά και όλες τις τιμές των στοιχείων του λεξικού αντίστοιχα. Πχ,:

```
>>> a={'Do':261, 'Do#':277, 'Re':293, 'Re#':311, 'Mi':329, 'Fa':349, 'Fa#':370, 'Sol':392, 'Sol#':415, 'La':440,
'La#':466, 'Si':494}
>>> print a.keys()
['Do', 'Re', 'Do#', 'La', 'Mi', 'Sol', 'Sol#', 'Fa', 'Si', 'Fa#', 'Re#', 'La#']
>>> print a.values()
[261, 293, 277, 440, 329, 392, 415, 349, 494, 370, 311, 466]
```

Προσοχή χρειάζεται μόνο στην περίπτωση που κάποια μέθοδος (ή συνάρτηση) συμβαίνει να αλλάζει το στοιχείο στο οποίο επιδρά. Για παράδειγμα η reverse() είναι μια μέθοδος που αντιστρέφει τη σειρά των στοιχείων μιας λίστας. Η ίδια η μέθοδος δεν επιστρέφει κάποια τιμή. Απλώς κάθε φορά που την καλούμε κάνει αντιστροφή:

```
>>> L=[1,2,3,4]
>>> L.reverse()
>>> print L
[4, 3, 2, 1]
>>> L.reverse()
>>> print L
[1, 2, 3, 4]
```

Και τώρα δείτε το εξής (εκ πρώτης όψεως παράξενο):

```
>>> L=[1,2,3,4]
>>> M=L
>>> print L,M
[1, 2, 3, 4] [1, 2, 3, 4]
>>> M.reverse()
>>> print L,M
[4, 3, 2, 1] [4, 3, 2, 1]
```

Εμείς είπαμε να αντιστρέψει μόνο τη M. Γιατί αντέστρεψε και τις δυο;

Η απάντηση έγκειται στον τρόπο με τον οποίο η Python χειρίζεται τον τελεστή ανάθεσης τιμής '='. Όταν γράψαμε L=[1,2,3,4] η Python πρώτα δημιούργησε τη λίστα [1,2,3,4] και στη συνέχεια της έδωσε το όνομα L. Με τη δεύτερη εντολή M=L δεν είπαμε να δημιουργήσει μια δεύτερη λίστα ίδια με την L, είπαμε απλώς ότι η L θα έχει και ένα δεύτερο όνομα M. Τελικά δημιουργήσαμε μία λίστα με δύο ονόματα. Αν θέλαμε να δημιουργήσουμε δυο λίστες θα έπρεπε να γράψουμε κάτι τέτοιο:

```
>>> L=[1,2,3,4]
>>> M=[1,2,3,4]
```

... ή κάτι τέτοιο για να μη ξαναγράφουμε τα στοιχεία της λίστας:

```
>>> L=[1,2,3,4]
>>> M=L[:]
```

Το L[:] είναι συντομογραφία του L[0:len(L)] και δηλώνει το τμήμα της λίστας από την αρχή μέχρι το τέλος της. Είναι ένας τρόπος για να πούμε στην Python ότι θέλουμε να δημιουργηθεί μια νέα λίστα ίδια με την L. Όχι όμως και ο μοναδικός. Θα μπορούσαμε να γράψουμε και κάτι τέτοιο:

```
>>> L=[1,2,3,4]
>>> M=L+[]
```

... που σημαίνει "δημιούργησε μια νέα λίστα που θα είναι η σύνδεση της L με την κενή λίστα ([ ]) και δώσε της το όνομα M".

Με όποιον τρόπο κι αν το κάνουμε, δημιουργούμε δυο ξεχωριστές ίδιες λίστες και τώρα η reverse() αλλάζει μόνο τη μία:

```
>>> print L,M
[1, 2, 3, 4] [1, 2, 3, 4]
>>> M.reverse()
>>> print L,M
[1, 2, 3, 4] [4, 3, 2, 1]
```

Σημειωτέον ότι με τον ίδιο τρόπο συμπεριφέρεται η Python με όλους τους τύπους δεδομένων, όμως δεν έχουν όλοι οι τύποι μεθόδους που τους αλλάζουν κι έτσι δεν γίνεται αντιληπτό πάντα αυτό το ζήτημα. Για να γίνει κατανοητό αυτό ας δούμε το ίδιο παράδειγμα με ακέραιους αντί για λίστες:

```
>>> L=5
>> M=L
>>> print L,M
55
```

Στην πραγματικότητα δεν υπάρχουν δυο αντίγραφα του αριθμού 5 στη μνήμη του υπολογιστή. Υπάρχει μόνο ένα 5 με δύο "ετικέτες" L και M. Αν μπορούσαμε να αλλάξουμε την τιμή αυτή απευθείας στη μνήμη τότε θα άλλαζαν και οι δυο μεταβλητές. Όμως ο συνήθης τρόπος αλλαγής τιμής είναι γράφοντας κάτι τέτοιο:

```
>>> M=M+1
```

Αυτό δεν αλλάζει το 5 σε 6. Δημιουργεί ένα 6, σε κάποιο άλλο σημείο στη μνήμη του υπολογιστή και μεταφέρει την "ετικέτα" M εκεί. Η L παραμένει στο 5 το οποίο δεν έχει αλλάξει. Κι έτσι:

```
>>> print L,M
56
```

Κάποιος λιγότερος "υποψιασμένος" προγραμματιστής θα μπορούσε να υποθέσει ότι είχαμε δυο μεταβλητές, στην αρχή βάλαμε την ίδια τιμή και στις δυο και μετά αλλάξαμε την τιμή στη μία. Το αποτέλεσμα (φαινομενικά) είναι το ίδιο. Όμως εμείς τώρα ξέρουμε τί έγινε πραγματικά.

Και πάλι, αν δε μας βολεύει ο τρόπος με τον οποίο λειτουργεί μια συνάρτηση ή μια μέθοδος μπορούμε να φτιάξουμε μια δική μας παραλλαγή. Για παράδειγμα, να είναι μια συνάρτηση που δέχεται μια λίστα και επιστρέφει την αντίστροφή της ως τιμή, χωρίς να αλλάζει την αρχική:

```
>>> def rev(L):
        M=L+[]
        M.reverse()
        return M
```

```
>>> J=[1,2,3,4]
>>> print J, rev(J), J
[1, 2, 3, 4] [4, 3, 2, 1] [1, 2, 3, 4]
```

Να και μια παραλλαγή που κάνει το ίδιο χωρίς να χρησιμοποιεί τη μέθοδο reverse():

```
>>> def rev2(L):

        if L==[]:
                return L
        H=L[0]
        T=L[1:]
        RT=rev2(T)
        return RT+[H]

>>> J=[1,2,3,4]
>>> print J, rev2(J), J
[1, 2, 3, 4] [4, 3, 2, 1] [1, 2, 3, 4]
```

Διαβάζοντας τον κώδικα της rev2() βλέπουμε ότι καταρχήν ελέγχει αν η λίστα που δίνουμε ως παράμετρο είναι κενή και τότε την επιστρέφει ως τιμή και σταματά. Αν η λίστα δεν είναι κενή, τότε διαβάζει το πρώτο στοιχείο της L[0], δημιουργεί μια νέα υπολίστα με όλα τα υπόλοιπα στοιχεία πλην του πρώτου (L[1:]) και τέλος επιστρέφει μια λίστα που είναι η σύνδεση της αντεστραμμένης υπολίστας με το L[0] στο τέλος. Το ενδιαφέρον είναι ότι για να υπολογίσει την αντεστραμμένη υπολίστα η rev2() καλεί τον εαυτό της! Αυτό είναι κάτι απόλυτα επιτρεπτό στις περισσότερες γλώσσες προγραμματισμού και δεν είναι περισσότερο παράξενο από μια έκφραση όπως η M=M+1 όπου μια μεταβλητή αναφέρεται στον εαυτό της. Πρέπει μόνο να προσέχουμε ώστε οι διαδοχικές κλήσεις κάποτε να σταματούν και μα συνεχίζουν επ' αόριστον. Εδώ είμαστε σίγουροι ότι οι κλήσεις κάποια στιγμή θα σταματήσουν επειδή (α) σε κάθε κλήση το μήκος της υπολίστας μικραίνει, (β) σε περίπτωση κενής λίστας η εκτέλεση σταματά και (γ) ο έλεγχος για κενή λίστα γίνεται πριν την κλήση στον εαυτό της. Αν κάποιο από τα (α), (β) και (γ) δεν ίσχυε τότε η rev2() δε θα σταματούσε ποτέ. Συναρτήσεις που καλούν τον εαυτό τους λέγονται αναδρομικές.

## 6.6 Ασκήσεις

Αφού τρέξετε τα παραδείγματα των ενοτήτων που προηγήθηκα, εκπονήστε τις παρακάτω ασκήσεις. Αναρτήστε τα αρχεία σας ως ένα συμπιεσμένο αρχείο.

**Άσκηση #1**

Φτιάξτε μια λίστα που θα περιέχει όλες τις συχνότητες των νοτών στο ακουστικό φάσμα 20-22000 Hz καθώς και τα ονόματά τους. Οι συχνότητες να είναι στρογγυλοποιημένες στον πλησιέστερο ακέραιο.

**Άσκηση #2**

Δίνεται η εξής εντολή εκχώρησης τιμής:

```
mysterysong=[['Do',3,'Σ1'],['Fa',2,'Σ2'],['Fa',1,'Σ3'],['Fa',4,'Σ4'],['Sol',3,'Σ5'],['La',2,'Σ6'],['La',1,'Σ7'],['La',4,'Σ8'],['La',1,'Σ9'],['Sol',1,'Σ10'],['La',1,'Σ11'],['La#',2,'Σ12'],['Mi',2,'Σ13'],['Sol',2,'Σ14'],['Fa',4,'Σ15']]
```

Κάθε στοιχείο της παραπάνω λίστας αποτελείται από μια τριάδα (πάλι λίστα) που ως 1ο στοιχείο έχει το όνομα μιας νότας, ως 2ο στοιχείο μια χρονική διάρκεια και ως 3ο στοιχείο μια συλλαβή (ή λέξη) η οποία θα πρέπει να τυπώνεται σε συγχρονισμό με τη νότα. Η χρονική διάρκεια είναι εκφρασμένη ως πολλαπλάσιο των 240 msec. Δηλαδή:

1 → 1x240 = 240 msec
2 → 2x240 = 480 msec
3 → 3x240 = 720 msec, κλπ

Να κατασκευάσετε πρόγραμμα σε Python που περιέχει τις κατάλληλες δομές dictionary και εντολές εκτέλεσης, οι οποίες με χρήση της winsound θα παίζουν το mysterysong. Μόλις αναγνωρίσετε το τραγούδι αλλάξτε τις συλλαβές Σ1-Σ15 ώστε να εμφανίζουν συγχρονισμένα τα λόγια του τραγουδιού.

**Άσκηση #3**

Φτιάξτε κώδικα Python που θα δημιουργεί μια τυχαία μουσική φράση με μήκος 5-10 νότες και στη συνέχεια θα παίζει τη φράση και την αντίστροφή της.

# Σημειώματα

## Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.

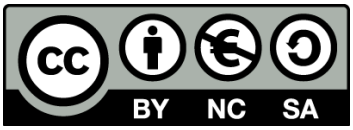
- Έκδοση 1.0 διαθέσιμη εδώ.

## 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

Εικόνες: Προέρχονται από Python IDLE.

Πίνακες

# 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 τη 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.