Ciberpianola
Un conversor de LilyPond a Wave

Projecte de curs de l'assignatura
INFORMÀTICA
Escola Politècnica Superior d'Enginyeria de Manresa

9 de desembre de 2009

Índex

1 Introducció

Aquest és el guió del projecte de curs de l'assignatura d'Informàtica de l'EPSEM. El projecte de curs és un treball en equip que té com a finalitat el disseny (de parts) i la implementació d'una aplicació informàtica realista en la que s'apliquen els coneixements i habilitats que s'han d'haver adquirit en l'assignatura.

En aquest apartat hi trobareu tot seguit una descripció de l'objectiu d'aquest projecte i les condicions i terminis en que cal dur-lo a terme.

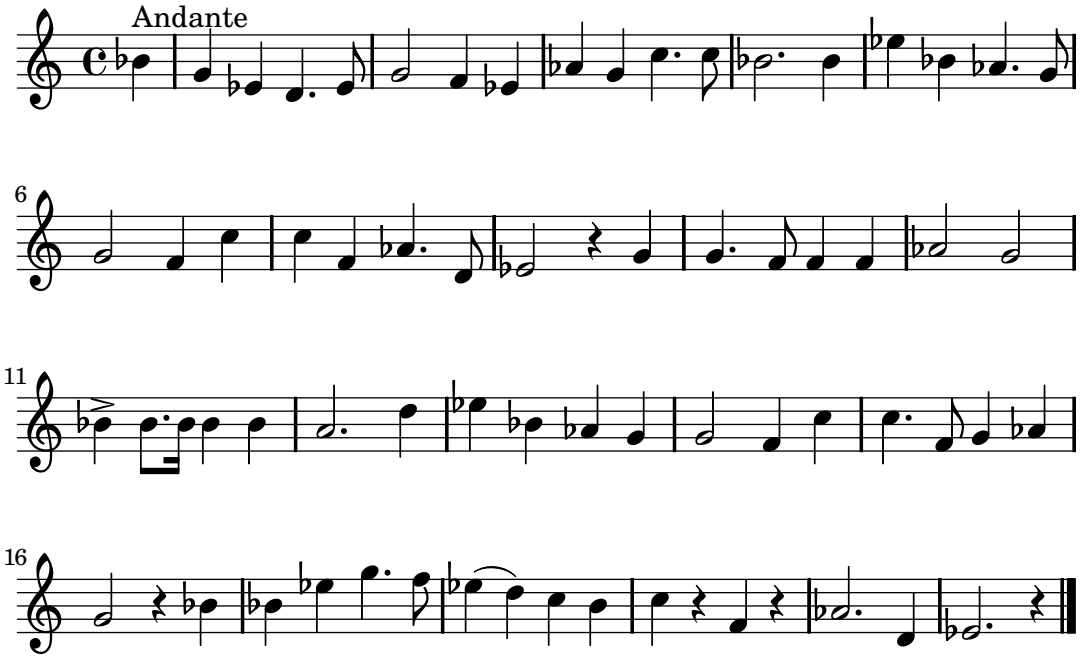
1.1 Objectiu

L'objectiu d'aquest projecte és dissenyar i implementar una aplicació que llegeix una partitura musical d'un fitxer i genera la música corresponent sobre un fitxer resultat. La partitura s'escriu emprant un subconjunt de la sintaxi usada per `LilyPond`. El fitxer resultat és un fitxer de so en format `wave`. Ambdós formats es descriuran més endavant amb el detall suficient.

El projecte treballa amb conceptes principalment informàtics però també requereix certs coneixements matemàtics, físics i, fins a cert punt, musicals.

Per tal que s'entengui bé què fa aquesta aplicació ho il·lustrarem amb un exemple.

Jakob Ludwig Felix Mendelssohn Bartholdy[1] (Hamburg, 1809 – Leipzig, 1847) fou un compositor romàntic que escrigué molta obra coral. Precisament aquest any se'n commemora el 200 aniversari del seu naixement. Una de les moltes obres que va escriure és *Abschied vom Wald: O Thaler weit, o Hohen* op. 59/3. A continuació hi teniu un bocí d'aquesta peça que correspon a un arranjament de la veu de soprano:



Nosaltres representarem aquest bocí de partitura de la següent forma:

[1] `http://en.wikipedia.org/wiki/Felix_Mendelssohn`

```
\include "catalan.ly"
{
sib'4^"Andante"          |
sol' mib' re'4. mib'8    |
sol'2 fa'4 mib'          |
lab' sol' do''4. do''8   |
sib'2. sib'4             |
mib'' sib' lab'4. sol'8  |
sol'2 fa'4 do''          |
do'' fa' lab'4. re'8     |
mib'2 r4 sol'            |
sol'4. fa'8 fa'4 fa'     |
lab'2 sol'               |
sib'4-> sib'8. sib'16 sib'4 sib' |
la'2. re''4              |
mib'' sib' lab' sol'     |
sol'2 fa'4 do''          |
do''4. fa'8 sol'4 lab'   |
sol'2 r4 sib'            |
sib' mib'' sol''4. fa''8 |
mib''4( re'') do'' si'   |
do'' r fa' r             |
lab'2. re'4              |
mib'2. r4
}
```

Com veieu, això és la descripció de la partitura en una mena de "llenguatge de programació". Està escrita en el llenguatge de partitures que usa l'eina `LilyPond`[2]. LilyPond és una eina per escriure partitures. El projecte no utilitza LilyPond, senzillament agafa en préstec la seva manera d'escriure partitures.

Suposeu que aquesta descripció l'emmagatzemeu en un fitxer de text anomenat `wald.ly`. Llavors, amb la nostra aplicació que hem anomenat `toca.py`, podríeu convertir la descripció en so fent el següent:

```
1  user@host: python toca.py wald.ly
```

El resultat seria un nou fitxer anomenat `wald.wav`. Aquest fitxer és un so digital: exactament igual que els fitxers `mp3` però en un altre format de fitxer, el format `wave`[3].

Si ara escoltéssiu aquest fitxer (per exemple fent clic damunt amb el navegador de fitxers i esperant que s'obri el reproductor de música) us trobaríeu amb la música de la cançó de Mendelssohn. Construir el programa que transforma la partitura en so és l'objectiu d'aquest projecte.

[2] `http://lilypond.org/`
[3] `http://en.wikipedia.org/wiki/WAV`

1.2 Condicions

1.2.1 Eines necessàries

Per desenvolupar aquest projecte és necessari tenir les següents eines instal·lades en Gnu/Linux (totes disponibles com paquets Debian):

- `nosetest`, `audacity`, `totem`

I és recomanable tenir també les següents (també disponibles com paquets Debian):

- `lilypond`, `lame`, `sox`

  D'altra banda, us seran molt útils un auriculars per poder sentir so en les aules informàtiques.

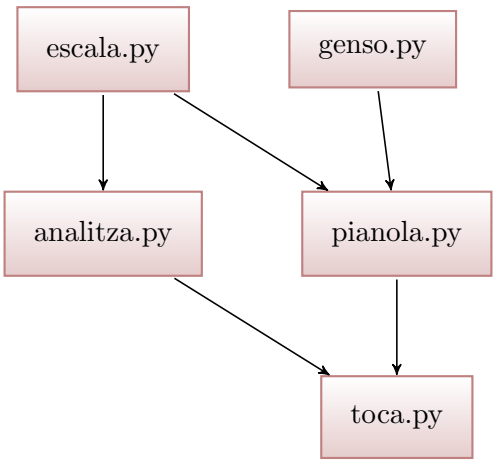
1.3 Manera d'abordar el projecte

El projecte té dues parts, una primera que és la part obligatòria i un conjunt d'ampliacions possibles que són optatives. Us aconsellem que primer feu una lectura superficial de tot l'enunciat. Després aneu treballant un apartat darrera l'altre. Cada apartat té clarament destacades les tasques que s'han d'acomplir. Les tasques de les parts optatives són progressivament menys detallades. És un efecte volgut.

2 Estructura de mòduls

En aquest apartat es mostra amb un diagrama quins mòduls composen el projecte i com estan organitzats. Aquesta informació és important per tenir una visió global de la seva arquitectura.

Cada mòdul correspon a un fitxer Python que conté un conjunt de variables i/o funcions relacionades les unes amb les altres. Una fletxa $A \rightarrow B$ significa que el mòdul B utilitza el mòdul A, per tant el mòdul B té un **import** del mòdul A.



La funcionalitat de cada mòdul es descriu a continuació:

escala És el mòdul responsable de determinar les freqüències de les notes. Donades expressions com ara `la'`, determina la freqüència que li correspon segons un model d'afinació.

genso És el mòdul encarregat de generar els sons de la durada, freqüència i volum necessaris.

analitza És el mòdul encarregat de processar les dades d'entrada, interpretar-les i produir com a resultat una representació interna del so que cal produir.

pianola És el mòdul encarregat de generar el so que correspon a la partitura a partir de la representació interna generada pel mòdul parse.

toca És el mòdul principal. L'encarregat de la interfície de comanda. L'usuari interacciona amb aquest mòdul.

3 Part obligatòria

3.1 Generació de soroll aleatori

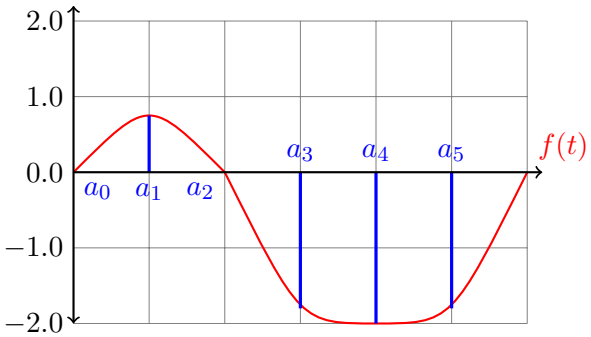
Aquest les el primer pas dels molts que condueixen al final del projecte. En aquest primer pas l'objectiu és implementar un petit programa (`soroll.py`) que simplement generi un fitxer wave (`soroll.wav`) amb 2 segons (2s) de soroll aleatori. Amb això aprendrem com es genera un fitxer wave.

És ben sabut que el so[4] es una successió de canvis de pressió en un medi i que es representa mitjançant una ona.

Un so digitalitzat no és res més que una representació de la ona d'un so plausible per a ser emmagatzemada en un computador. Així doncs, si la ona sonora segueix la funció $s(t)$, on t és el temps, la manera de digitalitzar-la és molt senzilla:

1. Es determina una freqüència de mostreig f_m.
2. Cada $\frac{1}{f_m}$ segons es mesura l'amplitud de la ona sonora i s'enregistra. D'aquest valor en diem *mostra*. Així, en el temps t_0, el valor de la mostra serà de $a_0 = s(t_0)$, en $t_1 = t_0 + \frac{1}{f_m}$, el valor de la mostra serà de $a_1 = s(t_1)$ i així successivament. La seqüència de mostres $\{a_0, \ldots, a_m\}$ és el so digitalitzat corresponent a la ona sonora $s(t)$.

La següent figura il·lustra aquests conceptes. En vermell hi teniu una ona qualsevol $f(t)$. En blau, hi teniu les diferents mostres de la ona. Per exemple, $f(t_0) = a_0$, $f(t_1) = a_1$, etc. Fixeu-vos que la digitalització d'aquesta ona és la seqüència de mostres $\{a_0, \ldots, a_5\}$.



[4] `http://ca.wikipedia.org/wiki/So`

Així doncs, un fitxer que conté un so digital en essència emmagatzema dues informacions:

1. La freqüència de mostreig.
2. La seqüència de mostres.

El cas dels fitxers en format `wave` segueix també aquesta regla i, a banda d'altres informacions menors, emmagatzemen principalment aquestes dues informacions.

Una particularitat de la codificació del so digital en format wave és que l'amplitud cal codificar-la en un byte. Això significa que només pot prendre valors enters en el conjunt $\{0, \ldots, 255\}$. Per tant, quan es digitalitza una ona sonora, cal escalar la mida de la amplitud de manera que l'amplitud màxima no superi el llindar de 255.

Per treballar amb els fitxers en format wave usarem el mòdul `wave`[5] de Python. Aquest mòdul ofereix operacions tant per llegir com per escriure en format wave.

A continuació, per tal d'entendre bé aquests conceptes i eines, comentarem el següent programa. Aquest programa genera una fitxer wave amb soroll aleatori.

```
1  # -*- encoding: utf-8 -*-
3  """
   Programa que genera un fitxer de so en format wave anomenat
5  soroll.wav que conte soroll aleatori
   """
7
   import wave
9  import random
11
   # Frequencia de mostreig
13 SAMPLE_FREQ = 22050
15 def genera_soroll():
       """
17     Genera un so aleatori de un segon de durada.
       El retorna en forma de string.
19     """
       sound = ''
21     for i in range(0,SAMPLE_FREQ):
           r = random.randint(0,255)
23         sound += chr(r)
       return sound
25
   if __name__ == "__main__":
27     f = wave.open('soroll.wav', 'w')
       f.setnchannels(1)
29     f.setsampwidth(1)
       f.setframerate(SAMPLE_FREQ)
31     f.setnframes(0)
       for w in range(0,3):
33         sound = genera_soroll()
           f.writeframes(sound)
35     f.close()
```

Aquest programa té dues parts diferenciades: el programa principal i la funció `genera_soroll`.

[5] `http://docs.python.org/library/wave.html`

La funció `genera_soroll`, genera un soroll d'un segon de durada. Com que la freqüència de mostreig és de 22050 mostres per segon, això vol dir que el resultat d'aquesta funció són exactament 22050 mostres aleatòries. Cada iteració de la sentència de la línia 21 genera una d'aquestes mostres. Fixeu-vos com a la línia 22 es genera una valor enter aleatori del conjunt $\{0, \ldots, 255\}$. Aquest valor es convertit en un caràcter a la línia 23 i concatenat a l'string `sound`. Cada mostra es representa com un caràcter ja que, com sabeu, hi ha una correspondència entre els caràcters i els enters (codificació dels caràcters). Amb aquest truc es pot representar un so com un string!. Per això la funció retorna un string (que en realitat representa un so). Representant un so amb un string aconseguim que la manipulació de sons en el programa sigui molt més simple.

El programa principal té una estructura molt semblant a altres programes que treballen amb fitxers. S'obre un fitxer, s'hi escriu i finalment es tanca. Ara, però, el fitxer és un fitxer en format wave. Si us hi fixeu, la línia 27 es dedica a obrir el fitxer wave en mode escriptura i les línies de la 28 a la 31 determinen la capçalera del fitxer (freqüència de mostreig, amplitud màxima, etc.). En el nostre cas, aquestes crides són sempre les mateixes atès que la freqüència de mostreig altres característiques seran invariants. La iteració de la línia 32 és la que afegeix so al fitxer. Fixeu-vos com ho fa: obté un so cridant a la funció `genera_soroll` i després l'escriu en el fitxer wave usant el mètode `writeframes`. Com la iteració s'executa 3 vegades, el fitxer conté 3s de soroll. La especificació precisa d'aquests mètodes la podeu trobar en el corresponent manual de la llibreria de Python.

Tasca 1
Implementeu el programa `soroll.py`, executeu-lo i comproveu que genera un fitxer wave de nom `soroll.wav`. Escolteu aquest fitxer amb uns auriculars i comproveu que és un soroll aleatori de tres segons de durada.

Tasca 2
Modifiqueu el programa anterior per tal que generi un soroll amb la meitat de volum. Fixeu-vos que el volum està relacionat amb l'amplitud del senyal. Al nou programa digueu-n'hi `soroll2.py`. Feu que el fitxer generat es digui `soroll2.wav`. Comproveu que aquest soroll té la meitat de volum que l'anterior.

3.2 Generació d'una nota

En aquest apartat, avançarem una mica sobre l'anterior i, en comptes de generar un soroll aleatori, generarem so corresponent a la nota la i, posteriorment, un so de qualsevol freqüència. La forma de ona que generarem serà sinusoïdal. Cap instrument real genera un so que correspongui a una ona sinusoïdal perfecta. Per tant el so que generarem necessàriament serà artificial. Generar un so menys artificial faria el projecte excessivament difícil.

La nota la central (LA4) es considera normalment que correspon a una freqüència de 440Hz. La funció que descriu aquesta ona sinusoïdal és doncs:

$$s(t) = A \sin(2\pi 440 t)$$

on t és el temps i A l'amplitud.

Com prenem mostres a 22050 mostres per segon, entre mostra i mostra passen $\frac{1}{22050}$ segons. Per tant, un segon de so es pot calcular de la següent manera:

```
t = 0.0
while t < 1.0:
  m = 127 * math.sin(2.0 * math.pi * 440 * t)
  t += 1.0 / 22050
```

En aquest tros de programa, la variable m va prenent valors a mida que la iteració avança. Cadascun d'aquests valors correspon a una de les mostres del senyal sinusoïdal de la nota la. Com l'amplitud triada és de 127, les mostres prenen valors en l'interval $[-127, 127]$. Si volguéssim que les mostres prenguessin valors en l'interval $[0, 255]$, per tal que fossin representables en un byte, caldria aplicar una translació de $+127$ unitats.

Noteu que aquesta translació de $+127$ fa que la ona quedi centrada en el valor 127. Totes les ones que es generaran hauran de centrar-se en aquest nivell independentment de la seva amplitud.

Considerant la translació i optimitzant un xic la iteració a base de fer factor comú dels càlculs constants, el calcul principal de les mostres de la nota la tindria aquest aspecte:

```
w = 2.0 * math.pi * 440
d = 1.0 / 22050
t = 0.0
while t < 1.0:
  m = 127 + 127 * math.sin(w * t)
  t += d
```

Amb aquesta informació ja es pot abordar aquest apartat.

Tasca 3
Modifiqueu el programa `soroll.py` *tot creant un nou programa que anomenareu* `nota.py`*. Aquest nou programa generarà un fitxer wave amb 3 segons d'una nota la anomenat* `nota.wav`*. A tal efecte cal que substituïu la funció* `genera_soroll` *per una nova funció* `genera_nota` *que retorna un string que codifica un segon de la nota la.*

Tasca 4
Modifiqueu el programa anterior afegint un paràmetre a la funció `genera_nota` *que correspongui a la freqüència del so que es genera. Proveu ara, amb aquesta nova funció, de generar un fitxer que contingui tres segons de so: un segon d'un so a 440Hz, un segon a 880Hz i un segon a 1760Hz.*

Tasca 5
Modifiqueu de nou el programa anterior. Ara afegiu un segon paràmetre a la funció `genera_nota` *que correspongui a la durada del so generat mesurada en segons. Amb aquesta nova funció, genereu un fitxer que contingui: un segon d'un so a 440Hz, mig segon de so a 880Hz i un un quart de segon de so a 1760Hz enganxats un darrera l'altre.*

Tasca 6
És interessant i sovint una bona manera de depurar els programes que generen so poder observar gràficament el so generat i que s'ha emmagatzemat en el fitxer wave. A tal efecte una aplicació com `audacity`[6] *és molt útil. Per exemple, per veure el contingut del fitxer* `nota.wav` *que hem generat abans podem usar audacity fent:*

```
user@host: audacity nota.wav
```

El programa obrirà una finestra i us mostrarà el senyal sonor. Amb el zoom (la icona és una lupa amb un símbol de sumar) i l'slider podeu ampliar i moure-us pel senyal. Veieu amb claredat les durades i les diferents freqüències?

3.3 El mòdul genso

Amb els progressos de l'apartat anterior és immediat implementar el mòdul `genso.py`. Aquest mòdul és l'encarregat d'encapsular la generació de so.

Tasca 7
Implementeu el mòdul `genso.py`*. Aquest mòdul únicament ha de contenir la funció* `genso(f,t,v)`*. En aquesta funció, els paràmetres són:*

f *Freqüència (Hz) del so generat.*

t *Durada (s) del so generat.*

v *Volum (real entre 0.0 i 1.0) del so generat. 0 indica volum mínim i 1 indica volum màxim.*

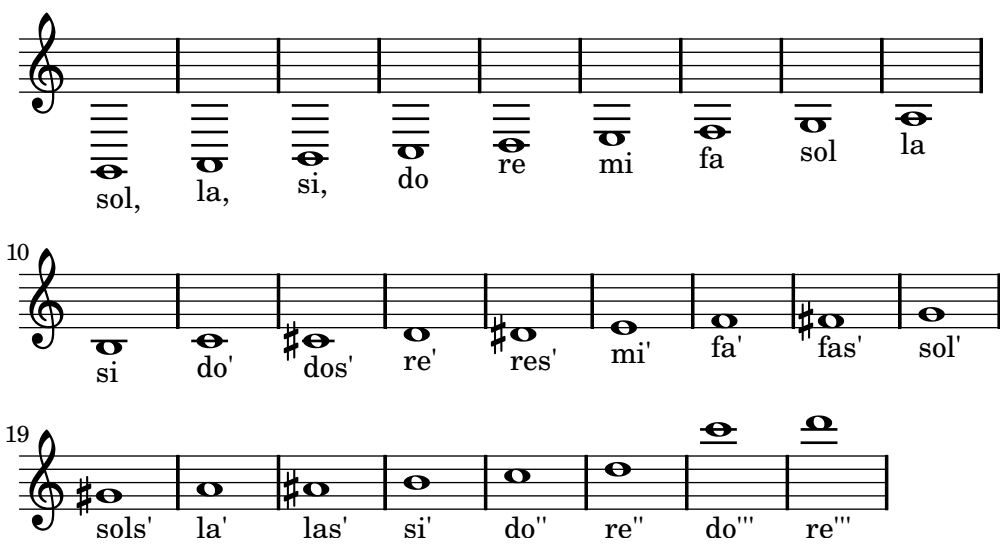
La funció retorna el so sinusoïdal mostrejat a 22050Hz de les característiques indicades pels paràmetres.

3.4 El modul escala

Per generar notes cal saber a cada nota quina freqüència li correspon a cadascuna. Aquest mòdul és el que té el coneixement sobre les notes i les freqüències que els i corresponen.

Per entendre bé aquest mòdul cal fer una mica de cultura prèvia. Tothom sap que les notes són: do, do sostingut, re, re sostingut, mi, fa, fa sostingut, sol, sol sostingut, la, la sostingut, si, i que es van repetint cada vegada en una octava diferent creant una seqüència de sons cada vegada més aguts. Per tant, en realitat hi ha més d'un do, d'un re, etc. Per tal de distingir les notes de cada octava hi ha diversos sistemes. Aquí usarem una notació derivada de la que es coneix com "notació de Helmholtz"[7]. Aquesta notació representa la octava a que pertany una nota afegint apòstrofs o comes darrera el nom de la nota. La següent figura il·lustra la notació:

[7] `http://en.wikipedia.org/wiki/Helmholtz_pitch_notation`



Així, emprant aquesta notació, el do central s'escriu `do'` i el rang d'un piano convencional va de `la,,` a `c'''''`.

Sabem que al `la'` (la central) li correspon habitualment la freqüència de 440Hz. La freqüència de la resta de les notes depèn del sistema d'afinació o temperament[8] que s'aplica. En la història de la Música s'han succeït diversos temperaments, cadascun d'ells amb avantatges i inconvenients. En aquest projecte usarem el temperament igual de dotze tons[9] (12-TET). En aquest sistema, cada la freqüència d'una nota correspon a la freqüència de la nota anterior multiplicada per $\sqrt[12]{2}$. D'aquesta manera, la freqüència d'una nota x coneguda la freqüència d'una nota w es pot calcular amb la següent fórmula:

$$f_x = f_w \sqrt[12]{2^{x-w}}$$

L'expressió $x - w$ és la distància que hi ha entre la nota w i x.

Per exemple, per saber la freqüència que correspon a la nota `do'`, considerant coneguda la freqüència de la nota `la'`, en fixarem en la taula següent que representa un segment de la escala:

...	0	1	2	3	4	5	6	7	8	9	...
	do'	dos'	re	res'	mi'	fa'	fas'	sol'	sols'	la'	

el terme $x - w$ és la distància entre la nota `la'` i la nota `do'`. En aquest cas és $0 - 9 = -9$. Noteu que els semitons (do sostingut, res sostingut,...) també computen. Per tant la freqüència de `do'` és:

$$f_{do'} = f_{la'} \sqrt[12]{2^{x-w}} = 440 \sqrt[12]{2^{-9}} = 261.626Hz$$

Finalment, si s'usa el temperament igual, com és el cas, les freqüències que corresponen als bemolls i als sostinguts són equivalents segons aquesta taula:

Bemoll	reb	mib	solb	lab	sib
Sostingut equivalent	dos	res	fas	sols	las

[8] `http://en.wikipedia.org/wiki/Musical_temperament`
[9] `http://ca.wikipedia.org/wiki/Temperament_igual#Temperament_igual_de_dotze_tons`

Amb aquesta informació podeu resoldre les següents tasques:

Tasca 8
Implementeu el mòdul `escala.py`*. Aquest mòdul ha de tenir les següents funcions:*

get_freq(n) *Donat el nom d'una nota segons la notació explicada abans entre* `la,,` *i* `do'''''` *incloent sostinguts i bemolls, retorna la freqüència corresponent en Hz o bé* `None` *en cas que la nota no sigui sintàcticament correcta.*

es_nota(n) *Retorna* `True` *si i només si* `n` *és el nom d'una nota sintàcticament correcta. Per a una nota* `n` *donada, si* `es_nota(n)` *retorna* `cert`*, llavors* `get_freq(n)` *ha de tornar una freqüència.*

3.5 El mòdul pianola

Aquest mòdul està dedicat a generar el so corresponent a una seqüència de notes donada en un format específic. La seva implementació es fa en dues fases. Aquest apartat correspon a la primera fase.

Una pianola[10] és un piano mecànic que toca de manera automàtica a partir d'una cinta de paper perforada. Aquest mòdul és quelcom semblant. Només conté una funció, `interpreta(f,c)`, que enregistra en un fitxer wave `f` la cançó enregistrada en la cinta `c`.

La cinta `c` no és altra cosa que una llista de tuples. Cada tuple correspon amb una acció de la pianola. En aquesta primera fase, la única acció possible serà tocar una nota o un silenci:

- Les tuples que corresponen a tocar una nota tenen la següent estructura:

  `(0, <nota>, <durada>)`

  L'enter zero indica que la tuple correspon a l'acció de tocar una nota, el camp `<nota>` és un string que indica la nota a tocar i el camp durada és:

  - Un real que diu la durada de la nota.
  - El valor `None`, que indica que aquesta nota té la mateixa durada que la nota o silenci anterior.

  La durada de les notes la expressarem de manera relativa. Així, una durada de valor δ en realitat s'allarga $\delta \cdot \tau$ segons. τ és una constant que determina la "velocitat" de la interpretació i l'anomenarem *tempo*. Així doncs, les tuples que formen una cinta codifiquen el temps com un factor del tempo. En aquesta versió del mòdul, el tempo el cosiderarem constant i amb valor $\tau = 1.5s$.

- Les tuples que corresponen a un silenci tenen la següent forma:

  `(1, <durada>)`

  i tenen el significat obvi: un silenci amb aquesta `<durada>`. Com en el cas de les notes, la durada pot ser `None`.

Així doncs, una possible "cinta" per a aquesta funció podria ser la següent llista:

[10] `http://www.pianola.org`

```
[(0, "mi'", 0.25), (0, "sol'", 0.125), (0, "do''", None),
 (1, None),
 (0, "mi'", 0.25), (0, "sol'", 0.125), (0, "do''", None)]
```

Amb aquesta informació podeu implementar fàcilment aquesta primera fase del mòdul.

Tasca 9
Implementeu una primera fase del mòdul `pianola.py`. *En aquesta fase implementareu la funció* `interpreta(f,c)`. *Aquesta funció té dos paràmetres:*

f *És un fitxer en format wave obert en mode escriptura i amb les capçaleres convenientment inicialitzades.*

c *És una llista de tuples que representa una melodia seguint el format que s'ha explicat prèviament.*

La funció no retorna res i modifica el fitxer `f` *afegint el so que correspon a la melodia codificada en* `c`. *Recordeu que el tempo és de* $\tau = 1.5s$.

Per implementar aquest mòdul utilitzeu el mòdul `genso`. *Noteu que un silenci es pot veure com un so de volum 0.*

Tal i com teniu el mòdul implementat ara, si en una "cinta" us apareixen dues o més notes iguals seguides, el so produït serà un continuum. Per exemple, si la cinta és

```
[(0, "sol", 0.25), (0, "sol", None)]
```

El so generat seria un `sol` de $0.5\tau s$ de durada. Aquesta forma de encadenar un so amb el següent sense separació es coneix com a *lligat*. En el nostre cas, i si no es diu en contrari, no volem generar un so tant lligat. Volem que una nota estigui lleugerament separada de la següent. Això vol dir que, si una nota té una durada de $\delta = 0.25\tau s$, nosaltres generarem un so de α segons i després un silenci de β segons de manera que $\alpha + \beta = \delta = 0.25\tau s$. Una manera senzilla d'implementar això és considerar que la durada del so i del silenci és proporcional a la durada de la nota. Per exemple, suposeu que $l = 0.95$, llavors, si una nota s'ha de generar amb una durada δ, el que fem en realitat és generar el so corresponent a la nota durant $\alpha = l\delta$ segons i a continuació generar un silenci de $\beta = (1 - l)\delta$ segons. l, doncs, és una constant que regula el "nivell de lligat" de la interpretació.

Tasca 10
Afegiu a la funció `interpreta` *el mecanisme per controlar el lligat que s'ha acabat d'explicar. Considereu un* l *de* 0.95.

3.6 El mòdul toca (I)

Aquest mòdul és, de fet, el programa principal. La seva feina és gestionar la interfície amb l'usuari. En aquesta primera versió farem una implementació que no llegirà fitxers en format `Lilypond` sinó que llegirà directament una "cinta", o sigui una llista de tuples adequada per a la pianola.

Tasca 11
Feu un petit programa per provar el mòdul `pianola.py` *que acabeu d'implementar. Anomeneu aquest programa* `toca.py`*. En aquest programa simplement heu de:*

1. *Llegir una llista de tuples. Ho podeu fer amb un únic* `input()`*.*
2. *Obrir un fitxer wave en mode escriptura.*
3. *Cridar a la funció* `pianola.interpreta` *passant-li com a paràmetres el fitxer i la llista que hem llegit.*
4. *Tancar el fitxer wave.*

Executeu el programa. Useu com a dades la llista de tuples que fa d'exemple a l'apartat anterior. Escolteu el fitxer resultant i comproveu que sona una melodia. La reconeixeu?

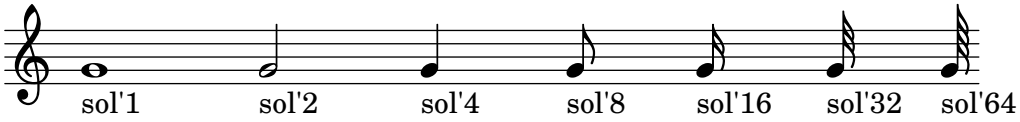
Ja teniu un programa que genera música!

3.7 El mòdul analitza

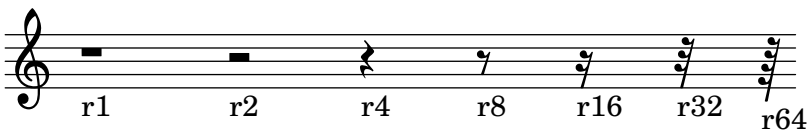
Aquest mòdul té com a objectiu analitzar un fitxer amb una partitura `Lilypond` i generar una "cinta" per la pianola. És un mòdul que fa un gran ús del tractament d'strings.

El mòdul `analitza.py` té una única funció `analitza(l)` que té un sol paràmetre `l`, un string que conté la partitura `LilyPond`, i retorna una "cinta", una llista de tuples per la pianola.

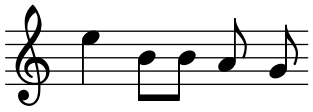
En aquesta secció s'explicarà com anar implementant una primera fase d'aquest mòdul. Per entendre alguns aspectes d'aquest mòdul és necessari entendre el concepte de durada d'una nota. Com segurament sabeu, les notes en un pentagrama es representen de diverses maneres segons la durada que han de tenir. Així és possible anar traçant el patró rítmic de la melodia. Les durades s'expressen de manera relativa. Així, si el tempo (recordeu el concepte del mòdul `pianola.py`) és de τ segons, una nota rodona dura un temps $\frac{\tau}{1}$ segons, una blanca dura $\frac{\tau}{2}$ i una negra $\frac{\tau}{4}$. Fixeu-vos que cada tipus de nota dura la meitat de temps que l'anterior. Per simplificar, sovint s'escriu la durada usant només el denominador de les fraccions anteriors: una rodona dura 1, una blanca 2, una negra 4 i així successivament. La notació de `LilyPond` fa servir aquest truc. D'aquesta manera `sol'4` representa un sol amb temps de negra i `si'8` un si amb temps de corxera. La següent figura mostra les durades més habituals:



Ara també és un bon moment per parlar dels silencis, que son símbols que denoten l'absència de so durant un cert temps. En la notació de `LilyPond` els silencis s'escriuen com `r`, de l'anglès *rest*. Les durades dels silencis funcionen exactament igual que les de les notes. Sent així, els silencis més habituals són els de la següent figura:



Finalment, la notació de `LilyPond` permet abreviar una mica les seqüències de notes de la mateixa durada amb la següent regla: si una nota dura el mateix que la anterior, llavors no cal escriure la durada. Seguint aquesta regla la seqüència `mi''4 si'8 si' la' sol'` representa una negra seguida de 4 corxeres com es veu a la figura següent:



Amb tota aquesta informació ja podeu entendre el primer exemple d'aquest document, el de Mendelssohn. Veieu la relació entre la partitura i la versió `LilyPond`?

Tasca 12
Comenceu per implementar la funció `error(m)`*. Aquesta funció és privada del mòdul. Això només vol dir que aquesta funció es cridarà únicament des dins del mateix modul* `analitza.py`*. La funció* `error` *té un paràmetre de tipus string. Quan es crida, escriu per la pantalla aquest missatge i avorta immediatament la execució del programa. Per avortar useu la funció* `sys.exit`*.*

Tasca 13
Implementeu la funció `analitza_nota(s)`*. Aquesta funció és privada del mòdul. La funció* `analitza_nota` *té com a paràmetre un string que és una nota (o un silenci) acompanyada, opcionalment, de la seva durada. La funció retorna una tuple que correspon a la nota entrada segons la especificació que s'ha fet en l'apartat 3.6.*

Per exemple, una crida amb argument `"mi4"` *hauria de tornar el tuple* `(0, "mi", 0.25)`*; una crida amb argument* `mi'` *hauria de tornar* `(0, "mi'", None)` *i una crida amb argument* `r8` *hauria de tornar* `(1, 0.125)`*. En cas que l'argument de la funció no tingui la sintaxi prevista, com per exemple* `mmi'4`*, cal que activeu un error usant la funció prevista a la tasca anterior.*

Fixeu-vos que, com hem dit en el mòdul `pianola.py`*, en les tuples hi codifiquem, la durada relativa.*

Per implementar aquesta funció useu els mètodes del tipus string. Per comprovar si el nom de la nota és correcte, useu les funcions del mòdul `escala.py`*.*

Tasca 14
Implementeu la funció `analitza_partitura(s)`*. Aquesta funció té com a paràmetre un string que representa una partitura en* `LilyPond` *i retorna la "cinta" (llista de tuples) que li correspon.*

L'argument, per a aquesta primera fase del mòdul, sempre té la forma següent:

```
\include "catalan.ly" { sol8 la si do re ... }
```

Les dues primeres paraules (`\include` *i* `"catalan.ly"`*) són obligades. De la mateixa manera, també és obligatori que les notes estiguin envoltades per unes claus. Entre cadascun dels elements sempre hi ha un o més espais i/o salts de línia.*

En cas que, en analitzar l'string, l'estructura no sigui la que s'espera, cal acabar l'execució usant la funció `error`*. Això pot passar, per exemple, si falta una clau o l'include.*

3.8 El mòdul toca (II)

El programa principal, com sabeu, és el mòdul `toca.py`. Ara cal modificar-lo per incorporar-hi les noves implementacions. A tal efecte seguiu les instruccions de les següents tasques.

Tasca 15
Modifiqueu el mòdul `toca.py` *per tal que faci la següent feina:*

1. *Obtenir un nom de fitxer de la línia de comandes usant* `sys.argv`*. Cal que controleu que l'usuari dona aquest nom quan invoca el programa* `toca.py`*.*
2. *Llegir del fitxer que l'usuari ha indicat tota la informació a un string.*
3. *Usant el mòdul* `analitza.py`*, analitzeu l'string obtingut abans i genereu una "cinta".*
4. *Obriu un fitxer wave en mode escriptura.*
5. *Crideu a la funció* `pianola.interpreta` *passant-li com a paràmetres el fitxer i la "cinta".*
6. *Tanqueu el fitxer wave.*

Tasca 16
Comproveu que el programa anterior funciona correctament. Feu-ho amb el fitxer `gegant.ly`*. Reconeixeu el so que genera? Genereu vosaltres mateixos altres partitures, escriviu-les en un fitxer i proveu a generar els fitxers wave corresponents.*

Potser voleu endur-vos-els en el vostre mp3. Possiblement el vostre reproductor digital no reconeix el format wave. Per endur-vos-els necessiteu convertir el fitxer wave a un altre format. Hi ha dos candidats possibles: `ogg` *i* `mp3`*.* `ogg` *és una format lliure*[11] *i per tant es preferible. Amb tot, és possible que malauradament el vostre reproductor no sàpiga reproduir* `ogg`*, llavors no teniu altre solució que convertir-ho a* `mp3`*.*

Per convertir a `ogg` *el fitxer* `gegant.wav` *feu:*

```
user@host: sox gegant.wav gegant.ogg
```

Per convertir a `mp3` *el fitxer* `gegant.wav` *feu:*

```
user@host: lame gegant.wav gegant.mp3
```

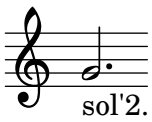
Si heu arribat aquí ja heu cobert la part bàsica del projecte. A partir d'ara i segons la vostra disponibilitat aneu afegint ampliacions.

4 Part d'ampliació

4.1 Durades amb punt

L'objectiu d'aquesta ampliació és suportar durades de notes amb punt. Si una nota de durada δ té punt, s'entén que la seva durada és $\delta + \frac{\delta}{2}$. Considereu doncs, una nota com la següent (una blanca

amb punt):



La durada d'una blanca és $\frac{1}{2}$ de la durada d'una rodona. Per tant, la blanca amb punt dura $\frac{1}{2} + \frac{1}{4} = \frac{3}{4}$ de la durada d'una rodona.

En la notació de `LilyPond`, una blanca amb punt s'escriu com `"sol'2."`, i una negra amb punt, per exemple, s'escriu com `"sol'4."` i silenci de blanca amb punt com `"r2."`.

Amb això podeu afrontar aquesta ampliació fàcilment.

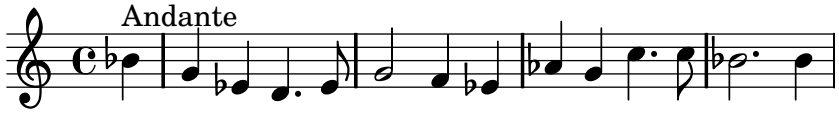
Tasca 17
Amplieu el projecte per tal que admeti durades amb punt. A tal efecte heu de modificar el mòdul `analitza.py` *per tal que reconegui les notes amb punt tal i com s'escriuen en* `LilyPond` *i assigni la durada adequada en la "cinta" de la pianola.*

Per fer aquesta modificació definiu una nova funció privada del mòdul. La funció `separa_durada(f)` *rep com a paràmetre la descripció d'una nota en format* `LilyPond` *i retorna un tuple amb la forma* `(resta_nota, durada)` *en la que* `durada` *és la durada de la nota en segons o* `None` *si no té durada, i* `resta_nota` *és un string que conté la resta de la nota una vegada eliminada la part de la durada.*

Amb la funció `separa_durada` *implementa de nou la funció* `analitza_nota` *de manera que tingui en compte les durades amb punt.*

4.2 Tempo

L'objectiu d'aquesta ampliació és fer que es tinguin en compte algunes anotacions de tempo en la partitura. Les anotacions de tempo indiquen lo ràpida o lenta que és la interpretació de la peça. A continuació teniu un exemple:



En el nostre context, el que fan és fixar la durada en segons d'una nota rodona. En la següent taula hi teniu les indicacions de tempo que admetrem en aquest projecte juntament amb la durada de rodona que representen (en segons):

tempo	durada (s)
Lento	3
Largo	2.4
Larghetto	1.88
Adagio	1.7
Adagietto	1.6
Andante	1.4
Moderato	1.04
Vivace	0.86
Allegro	0.83
Presto	0.67
Prestissimo	0.57

En notació `LilyPond`, les anotacions de tempo tenen aquesta forma: `sib'4^"Andante"`.

Tasca 18
Afegiu un nou mòdul de nom `tempo.py`*. En aquest mòdul heu de definir la funció* `get_durada(t)` *en que* `t` *és un string que conté una indicació de tempo i retorna la durada d'una rodona per aquest tempo o bé* `None` *si el tempo no era conegut. També heu de definir la funció* `es_tempo(t)` *que retorna* `True` *només si* `t` *és un tempo.*

Tasca 19
Amplieu el projecte per tal que admeti notes amb indicació de tempo. A tal efecte heu de modificar els mòduls `analitza.py` *i* `pianola.py` *per tal que reconeguin les indicacions de tempo tal com s'escriuen amb* `LilyPond`*.*

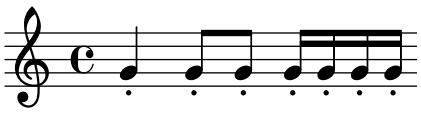
Al mòdul `pianola.py` *hi afegiu un nou tipus de tupla* `(2, tempo)`*. Aquesta tupla indica un canvi de tempo. El valor* `tempo` *és un real que indica la nova durada del tempo en segons.*

El mòdul `analitza.py` *el modifiqueu definint una nova funció privada. La funció* `separa_tempo(f)` *rep com a paràmetre la descripció d'una nota en format* `LilyPond` *i retorna un tuple amb la forma* `(tempo, resta_nota)` *en la que* `tempo` *és la durada del tempo en segons o* `None` *si la nota* `f` *no tenia indicació de tempo. El camp* `resta_nota` *és un string que conté la resta de la nota una vegada eliminada la part del tempo.*

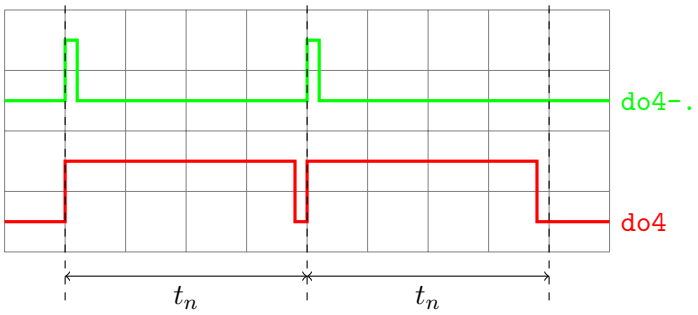
Amb la funció `separa_tempo` *implementa de nou la funció* `analitza_nota` *de manera que tingui en compte els canvis de tempo i generi una nova tupla*

4.3 Picats

L'objectiu d'aquesta ampliació és admetre notes amb execució picada en comptes de lligada. Una nota amb execució picada té la mateixa durada que la mateixa nota lligada però només sona durant una petita fracció de temps. En les partitures les notes picades es distingeixen afegint un punt just sobre o sota de la nota com en aquest fragment:



En el següent diagrama es veu el diagrama d'execució de dues notes `do4`. El diagrama superior (verd) és la execució d'una nota picada i el diagrama inferior (vermell) el d'una nota lligada. Si us fixeu, el temps que s'assigna a ambdues és el mateix, t_n, malgrat en l'execució picada només es produeix so durant un instant.



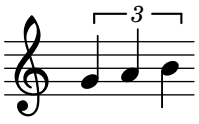
Les notes picades es representen en `LilyPond` amb el sufix `-.` com per exemple `do4-.`.

Tasca 20
Per implementar l'execució picada caldrà modificar el mòdul `pianola.py` *i també al mòdul* `analitza.py`*. En primer lloc modificarem el mòdul* `pianola` *afegint un nou tipus de tuple a la "cinta". Aquesta tuple indicarà "nota amb execució picada" i tindrà aquesta estructura:* `(3, nota, durada)`*. Si us fixeu és la mateixa estructura que per una nota convencional. La funció* `analitza.interpreta`*, per generar el so corresponent a aquesta nota, ha de generar un so de durada* $\frac{1}{32}$ *del tempo i completar la resta del temps amb un silenci.*

Tasca 21
Modifiqueu el mòdul `analitza.py` *de manera homòloga a com heu fet en altres ampliacions per tal que analitzi si una nota té punt o no. Feu-ho afegint una nova funció* `separa_picat(f)` *i refent la funció* `analitza` *per tal que, en cas de trobar una nota amb picat, genera la tuple de la "cinta" que codifica els picats.*

4.4 Tresets

Els tresets són una conjunts de tres notes de la mateixa mena (negres, corxeres, etc.) la durada de les quals té un tractament especial. Així per exemple, aquest treset de negres que hi ha a continuació:



significa que la durada de cadascuna de les tres notes no és de $\frac{1}{4}$ de rodona com és habitual sinó que és de $\frac{1}{6}$. En general, la durada d'una nota que forma part d'un treset és de $\frac{2}{3}$ de la seva durada convencional. En el cas de l'exemple, fixeu-vos que $\frac{1}{4} \cdot \frac{2}{3} = \frac{1}{6}$.

En `LilyPond` la notació emprada per a un treset és la següent:

```
\times 2/3 { nota1 nota2 nota3 }
```

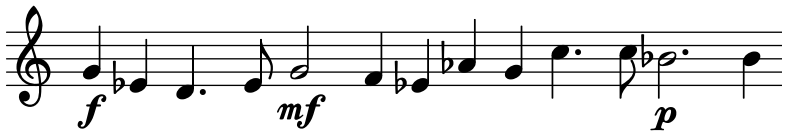
el cas de l'exemple anterior s'escriu doncs com:

```
\times 2/3 { sol'4 la' si' }
```

Tasca 22
Afegiu al mòdul `pianola.py` un nou tipus de tuple que indiqui començament i final de treset. Modifiqueu el mòdul `analitza.py` per tal que interpreti correctament l'estructura dels tresets i generi les tuples de la cinta que s'escaiguin.

4.5 Dinàmica

L'objecte d'aquesta ampliació és admetre algunes indicacions de dinàmica en la partitura. Les indicacions de dinàmica expliquen amb quina intensitat cal interpretar la melodia. Per exemple, una indicació de *fortissimo* (**ff**) indica que cal interpretar amb molta intensitat. En les partitures, s'entén de la següent manera:



En la notació `LilyPond`, aquestes indicacions s'escriuen annexades a la nota com en `do'4\ff`. Fixeu-vos que es poden donar combinacions com aquesta: `do''8.-.\f` o bé `do^"Allegro"\pp`.

En el nostre cas entendrem que aquestes indicacions regulen el volum (és a dir l'amplitud del senyal generat). Les indicacions i el volum (mesurat enter 0.0 i 1.0) corresponent seguiran aquesta taula:

indicació	ff	f	mf	mp	p	pp
volum	1	0.6	0.3	0.2	0.1	0.05

Tasca 23
Afegiu el necessari per interpretar indicacions de dinàmica. Una possibilitat passa per afegir un nou mòdul `dinamica.py` que transforma les indicacions de dinàmica en valors de volum. Posteriorment caldrà modificar el mòdul `pianola.py` per afegir un nou tipus de tuple per a la "cinta", que indiqui canvis de volum a la pianola. Finalment també caldrà introduir canvis al mòdul `analitza.py` per tal d'analitzar les indicacions de dinàmica.