DU SON AVEC PYTHON

TP

Dans ce TP nous allons créer (et écouter) des fichiers sonores ; les résultats produits seront sauvegardés dans le fichier ci-dessous.

```
FICHIER = "M:/mon_son.wav"
```

Une fois qu'il est créé, on peut l'écouter en double-cliquant dessus dans le dossier qui le contient.

Pour arriver à ce fichier, on utilise quelques boîtes noires des modules `scipy` et `numpy`.

```
from scipy.io.wavfile import write
from numpy import array, float32
```

Un fichier son consiste en la discrétisation d'un signal, on prendra

```
RÉS_HORIZONTALE = 44100   # en Hz
```

comme taux d'échantillonage, parce qu'un certain Shannon a dit que pour qu'une fréquence soit perceptible dans un signal discrétisée, il faut que le taux d'échantillonage soit plus deux fois supérieur à cette fréquence. Hors l'oreille humaine perçoit les sons jusqu'à environ 20 kHz.

Enfin, le dernier programme ci-dessous prend en argument une liste de valeurs entre -1 et 1 (le signal discrétisé) et fabrique le fichier son correspondant.

```
def CréerFichierSon(Valeurs) :
    données = array(Valeurs, dtype=float32)
    write(FICHIER, RÉS_HORIZONTALE, données)
```

Partie I — Les notes de la gamme tempérée occidentale

L'intervalle entre une fréquence et la fréquence double s'appelle une *octave*. Traditionnellement on subdivise les octaves en douze sous-intervalles égaux, appelés *demi-tons*, et qui donne douze noms de notes. Rangeons ces douze notes dans un *dictionnaire*

```
NOTES = {
 "La" : 0,
 "La♯" : 1, "Si♭" : 1,
 "Si" : 2,
 "Do" : -9,
 "Do♯" : -8, "Ré♭" : -8,
 "Ré" : -7,
 "Ré♯" : -6, "Mi♭" : -6,
 "Mi" : -5,
 "Fa" : -4,
 "Fa♯" : -3, "Sol♭" : -3,
 "Sol" : -2,
 "Sol♯" : -1, "La♭" : -1
}
```

qui s'utilisera ainsi : si `note` est une chaîne de caractères représentant le nom d'une note, par exemple `"Fa♯"`, alors `NOTES[note]` donne le nombre de demi-tons qui séparent cette note du La (la note de référence) de la même octave.

Dans la suite, on utilisera comme note de référence le La de l'octave numérotée 3 (le quatrième La du piano, le premier La du violon, ou encore la corde aiguë à vide du violoncelle).

```
DIAPASON = 440   # en Hz
```

Pour désigner une note, on utilisera une chaîne de caractères de la forme `"Fa♯:4"`, sans espace : on peut en déduire le nombre d'octaves et le nombre de demi-tons qui séparent cette note de la note de référence, ici une octave $(4-3)$ moins trois demi-tons (car `NOTES["Fa♯"]` vaut -3). Chaque octave correspond à une multiplication (ou une division) de la fréquence par 2, et chaque demi-ton correspond à une multiplication (ou une division) de la fréquence par $2^{1/12}$.

Exercice 1 — Écrire un programme `Fréquence(note)` qui prend en argument une chaîne de caractères de la forme `"Mi♭:5"` et renvoie la fréquence (en Hz) de la note correspondante.

```
>>> Fréquence("La:3")
440.0
>>> Fréquence("La:4")
880.0
>>> Fréquence("Do:3")
261.6255653005986
```

Partie II — Premier test : la sinusoïde

Dans cette partie on va fabriquer et écouter le premier signal sonore, qui sera une sinusoïde. Autrement dit un son « pur », c'est-à-dire constitué d'une seule fréquence. Ce n'est pas très intéressant comme son, mais il faut bien commencer par quelque chose.

```
from numpy import sin, pi as π
```

On utilisera le programme ci-dessous pour ajouter, à chaque signal sonore, un déphasage aléatoire, parce que lorsqu'on va mélanger plusieurs signaux, s'ils sont tous en phase, ça ne donnera rien de bon.

```
from random import random
def Aléa(a = 0, b = 1) :
    return a + (b - a) * random()
```

Enfin, on va utiliser une variable qui représentera le volume global du son, et qu'on peut ajuster suivant la puissance du moniteur d'écoute dont on dispose.

```
AMPLITUDE = 0.1
```

Voici donc le programme qui renvoie une fonction sinusoïdale de fréquence donnée (et de déphasage aléatoire).

```
def Sinusoïde(f) :
    ω = 2 * π * f
    φ = Aléa(0, 2 * π)
    def s(t) :
        return AMPLITUDE * sin(ω * t + φ)
    return s
```

Enfin, on utilise un programme qui étant donné un signal (donc une fonction) $s(t)$ et une durée d construit le signal discrétisé correspondant.

```
def FabriquerSon(s, d) :
    nb_points = int(d * RÉS_HORIZONTALE)
```

```
    Valeurs = [s(i / RÉS_HORIZONTALE) for i in range(nb_points)]
    return Valeurs
```

Pour écouter, il n'y a plus qu'à exécuter les deux lignes suivantes, et aller double-cliquer sur le fichier son produit. Si l'on entend la tonalité du téléphone, pendant cinq secondes, c'est que tout marche correctement.

```
V = FabriquerSon(Sinusoïde(Fréquence("La:3")), 5.0)
CréerFichierSon(V)
```

Partie III — Mélange de plusieurs sons

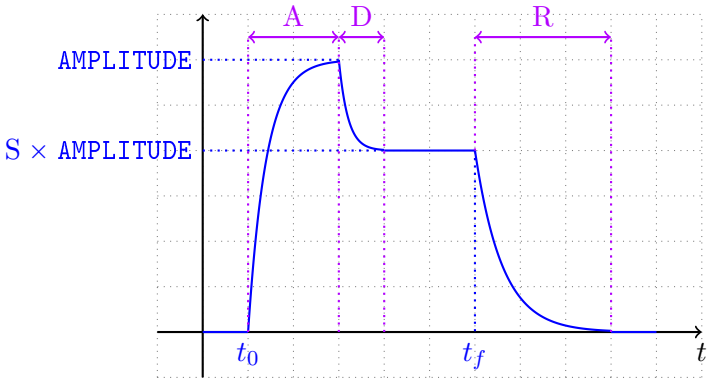
Partie IV — Autres signaux

Partie V — Enveloppes

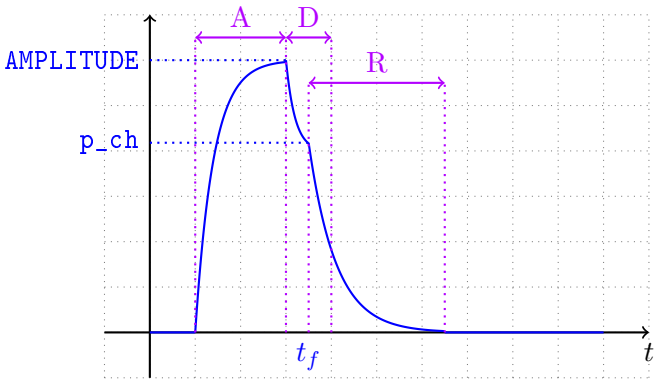
Pour passer d'un son constant au cours du temps à une *note*, c'est-à-dire un son ayant un début et une fin, on va multiplier le signal correspondant à ce son par une fonction d'amplitude `amp(`t`)`.

Cette fonction dépendra de quatre paramètres A, D, S et R, avec A (la durée d'attaque), D (la durée d'atténuation) et R (la durée de relâchement) homogènes à des temps et exprimés en secondes, et S (le niveau de maintien) un réel sans dimension entre 0 et 1. De plus, cette fonction `amp(`t`)` va dépendre de l'instant t_0 à partir duquel la note est jouée et de l'instant t_f (obligatoirement supérieur à t_0) à partir duquel elle n'est plus jouée.

La fonction `amp(`t`)` est constante (et nulle) pour $t \leqslant t_0$ et $t \geqslant t_f + \mathrm{R}$, elle est constante pour $t_0 + \mathrm{A} + \mathrm{D} \leqslant t < t_f$, et exponentielle par morceaux pour $t_0 \leqslant t < t_0 + \mathrm{A}$, pour $t_0 + \mathrm{A} \leqslant t < t_0 + \mathrm{A} + \mathrm{D}$ et pour $t_f \leqslant t < t_f + \mathrm{R}$. Voici sa courbe.



Une subtilité : si on arrête de jouer la note avant la fin de l'attaque ou de l'atténuation, c'est-à-dire si $t_f < t_0 + \mathrm{A} + \mathrm{D}$, le dernier morceau exponentielle de la courbe par de la valeur « où l'on en était », qui peut être plus haut (ou plus bas) que S × `AMPLITUDE`.



Exercice ? — Écrire le programme `Enveloppe(A, D, S, R, `t_0`, `t_f`)` qui fabrique et renvoie le programme `amp(`t`)`. Puis en déduire un programme `GénérateurEnveloppe(A, D, S, R)` qui fabrique et renvoie le programme `E(`t_0`, `t_f`)` renvoyant pour tous t_0 et t_f la fonction `amp(`t`)` correspondante, à A, D, S et R fixés.

Partie VI — Notes

Lorsque plusieurs instruments jouent ensemble la même ligne mélodique (par exemple les violons de l'orcheste) ils ne jouent pas rigoureusement tous la même fréquence. Le son obtenu, caractéristique, s'obtient artificiellement en superposant différent signaux de fréquences proches les uns des autres.

Exercice ? — Écrire le programme `BougerFréquence(`f`)` qui renvoie une fréquence aléatoire sensiblement égale à f.

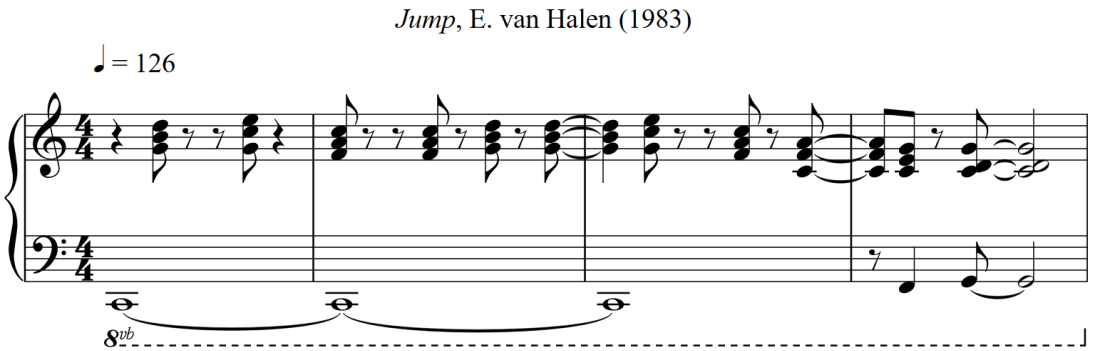
On veut maintenant produire des notes, donc. Une note, ce sera un couple `(V, pt_dép)` où `V` est le signal discrétisé de la note et `pt_dép` est un entier indiquant le numéro du point (dans le fichier son final) sur lequel la note commence. Si l'on a plusieurs *notes* (V_k, p_k), le signal discrétisé regroupant toutes ces notes sera la grande liste `V` définie par

$$\texttt{V[}i\texttt{]} = \sum_k \mathrm{V}_k[i - p_k],$$

mais il faudra évidemment prendre garde au fait que $i - p_k$ doit être compris entre 0 et $\mathrm{len}(\mathrm{V}_k) - 1$ lorsqu'on calculera cette somme.

Exercice ? — Écrire le programme `Instrument(timbre, A, D, S, R, unisson = 1)` qui étant donné un signal de référence `timbre` (la sinusoïde, le carré, etc.) fabrique et renvoie le programme `FabriquerNote(note, `t_0`, `t_f`)` qui à chaque chaîne de la forme `"Do:4"`, t_0 et t_f associe le couple `(V, pt_dép)` correspondant. Pour chaque note, on utilisera le générateur d'enveloppe défini par les paramètres A, D, S et R, et on superposera plusieurs signaux identiques (autant qu'indiqués par la variable `unisson`) ayant chacun des fréquences voisines de la fréquence théorique de la note ciblée.

Exercice ? —



```
Notes = [("Do:0", 0, 12), ("Fa:0", 12.5, 1), ("Sol:0", 13.5, 2.5), ("Sol:3
", 1, 0.5), ("Si:3", 1, 0.5), ("Ré:4", 1, 0.5), ("Sol:3", 2.5, 0.5), ("Do:
4", 2.5, 0.5), ("Mi:4", 2.5, 0.5), ("Fa:3", 4, 0.5), ("La:3", 4, 0.5), ("D
o:4", 4, 0.5), ("Fa:3", 5.5, 0.5), ("La:3", 5.5, 0.5), ("Do:4", 5.5, 0.5),
 ("Sol:3", 6.5, 0.5), ("Si:3", 6.5, 0.5), ("Ré:4", 6.5, 0.5), ("Sol:3", 7.
5, 1.5), ("Si:3", 7.5, 1.5), ("Ré:4", 7.5, 1.5), ("Sol:3", 9, 0.5), ("Do:4
", 9, 0.5), ("Mi:4", 9, 0.5), ("Fa:3", 10.5, 0.5), ("La:3", 10.5, 0.5), ("
Do:4", 10.5, 0.5), ("Do:3", 11.5, 1), ("Fa:3", 11.5, 1), ("La:3", 11.5, 1)
, ("Do:3", 12.5, 0.5), ("Fa:3", 12.5, 0.5), ("Sol:3", 12.5, 0.5), ("Do:3",
 13.5, 2.5), ("Ré:3", 13.5, 2.5), ("Sol:3", 13.5, 2.5)]

instrument = Instrument(DdS, 0.01, 0, 1, 0.2, 4)
tempo = 126 ; nb_mesures = 4 ; battements_par_mesure = 4
d = 60 * nb_mesures * battements_par_mesure / tempo

V = CréerMorceau(Notes, instrument, tempo, d) ; CréerFichierSon(V)
```

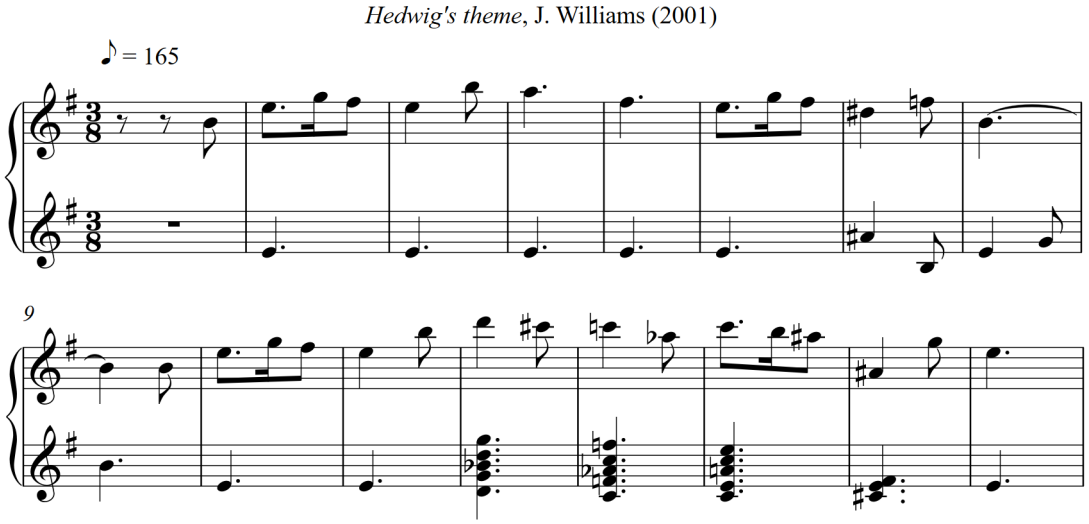
Partie VII — Synthèse FM

Qui sera plutôt de la modulation de phase, et non pas de fréquence, mais passons sur cette subtilité. Fixons deux paramètres $\mathrm{I}_m = 1$ et $r = 27/12$ (par exemple). On obtient des sons intéressants en prenant comme signal la fonction

$$s(t) = \texttt{AMPLITUDE} \times \sin(\omega t + \mathrm{I}_m \sin(r \times \omega t)).$$

Exercice ? — Écrire le programme `Cloche(`f`, I_m = 1, r = 27/12)` qui fabrique et renvoie le programme $s(t)$ calculant la fonction ci-dessus.

Exercice ? — Tester avec le morceau ci-dessous. On pourra essayer de faire varier I_m et r pour voir ce que cela change.




```
Notes = [("Si:5", 2, 1), ("Mi:6", 3, 1.5), ("Sol:6", 4.5, 0.5), ("Fa♯:6",
5, 1), ("Mi:6", 6, 2), ("Si:6", 8, 1), ("La:6", 9, 3), ("Fa♯:6", 12, 3), (
"Mi:6", 15, 1.5), ("Sol:6", 16.5, 0.5), ("Fa♯:6", 17, 1), ("Ré♯:6", 18, 2)
, ("Fa:6", 20, 1), ("Si:5", 21, 5), ("Si:5", 26, 1), ("Mi:6", 27, 1.5), ("
Sol:6", 28.5, 0.5), ("Fa♯:6", 29, 1), ("Mi:6", 30, 2), ("Si:6", 32, 1), ("
```

```
Ré:7", 33, 2), ("Do♯:7", 35, 1), ("Do:7", 36, 2), ("La♭:6", 38, 1), ("Do:
7", 39, 1.5), ("Si:6", 40.5, 0.5), ("La♯:6", 41, 1), ("La♯:5", 42, 2), ("S
ol:6", 44, 1), ("Mi:6", 45, 3),
("Mi:5", 3, 3), ("Mi:5", 6, 3), ("Mi:5", 9, 3), ("Mi:5", 12, 3), ("Mi:5",
15, 3), ("La♯:5", 18, 2), ("Si:4", 20, 1), ("Mi:5", 21, 2), ("Sol:5", 23,
1), ("Si:5", 24, 3), ("Mi:5", 27, 3), ("Mi:5", 30, 3), ("Ré:5", 33, 3), (
"Sol:5", 33.05, 3), ("Si♭:5", 33.1, 3), ("Ré:6", 33.15, 3), ("Do:5", 36,
3), ("Fa:5", 36.05, 3), ("La♭:5", 36.1, 3), ("Do:6", 36.15, 3), ("Do:5",
39, 3), ("Mi:5", 39.05, 3), ("La:5", 39.1, 3), ("Do:6", 39.15, 3), ("Do♯:
5", 42, 3), ("Mi:5", 42, 3), ("Fa♯:5", 42, 3), ("Mi:5", 45, 3)]

instrument = Instrument(Cloche, 0.001, 2, 0, 2)
tempo = 165 ; nb_mesures = 16 ; battements_par_mesure = 3
d = 60 * nb_mesures * battements_par_mesure / tempo

V = CréerMorceau(Notes, instrument, tempo, d) ; CréerFichierSon(V)
```