PROJET
"L'harmonie est numérique"

Veuillez lire attentivement le sujet du début jusqu'à la fin avant de commencer.

1 Préambule

La légende raconte que le mathématicien et philosophe Pythagore a été convaincu tout au long de sa vie que la Nature était intégralement régie par des rapports de nombres y compris les sons. Il avait reconnu que l'harmonie musicale exploite des rapports simples entre nombres entiers. Longtemps après, Leibniz soutient que : « La musique est un exercice d'arithmétique secrète où l'esprit ne réalise pas qu'il compte ».
A travers le temps, une multitude de liens entre mathématiques et musique ont été découverts ou introduits, puis étudiés et utilisés par les concepteurs d'instruments ou les compositeurs. D'où l'affirmation excessive que rien ne distingue musique et mathématiques.

Afin d'apprécier ce lien entre ces deux disciplines, nous vous proposons dans ce projet de **créer de la musique en manipulant des mathématiques**.

2 Travail à faire

Le présent projet est à réaliser en langage Python. Son principal objectif est la création de nouvelles partitions musicales. Pour ce faire, une base de données de partitions musicales est fournie dans le fichier "partitions.txt". Elles correspondent aux mélodies des comptines pour enfants. Ce fichier doit être lu, analysé puis servir pour composer de nouveaux rythmes musicaux à offrir aux tous petits :-)

(**A**) **Fonctionnalités**
La réalisation implique :

(a) Le codage des notes musicales ;

(b) L'attribution de fréquences et des durées aux notes ;

(c) L'application de transformations mathématiques telles que la transposition ou l'inversion pour la création de nouvelles partitions musicales ;

(d) L'utilisation des chaînes de Markov (2 versions) pour la création de nouvelles partitions à partir d'une ou plusieurs mélodie (s) ;

(**B**) **Application**
L'application finale doit offrir à l'utilisateur un menu pour choisir parmi :

(a) La lecture d'une partition parmi celles qui lui sont proposées à l'écran.

(b) La lecture d'une nouvelle partition fournie sous forme d'un fichier.

(c) La transformation d'une partition donnée en appliquant au choix la transposition ou l'inversion.

(d) La composition d'une nouvelle partition à partir du style des partitions présentes dans la base de données.

(e) La possibilité d'enrichir la base de donnée en y ajoutant de nouvelles partitions.

(**C**) **Illustration graphique**
De plus, toute lecture musicale doit être accompagnée d'une illustration graphique qui peut être :

(a) Un dessin d'un instrument musical avec animation (Ex. Piano) ;

(b) Un spectrogramme ;

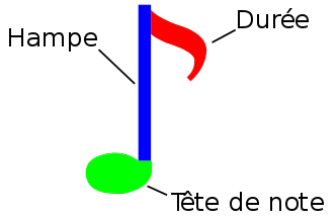
(c) Des notes musicales ;

(d) Autre proposition ...

NB : Il est à noter que toute nouvelle partition générée ou lue par l'utilisateur doit être ajoutée systématiquement à la base de données.

3 Comprendre les notes musicales

3.1 L'anatomie d'une note



Le symbole visuel d'une note de musique sur une partition est constitué de :

— Une tête qui indique la hauteur du son (sa fréquence en hertz) ;

— Un ou plusieurs crochets indiquant sa durée temporelle ou sa longueur ;

— Une hampe représentant un trait vertical attaché à la tête et qui soutient les crochets de la durée.

Selon leurs hauteurs, nous comptons dans l'échelle diatonique 7 notes qui s'énumèrent de la plus grave à la plus aigue : Do, Ré, Mi, Fa, Sol, La, Si puis à nouveau Do.

3.2 Fréquence d'une note

Tout son musical (ou note) possède une fréquence fondamentale (nombre de vibrations par seconde calculé en hertz) correspondant à sa hauteur.
Dans la pratique, pour mesurer la fréquence d'une note par rapport à une note de départ, des **gammes** sont calculées. La désignation de la fréquence de départ dépend de l'instrument utilisé et du type de musique.
Dans ce projet, nous allons adopter **les gammes naturelles** et la distribution des fréquences (Hz) sera comme suit :

Do	Ré	Mi	Fa	Sol	La	Si
264 Hz	297 Hz	330 Hz	352 Hz	396 Hz	440 Hz	495 Hz

3.3 Figures / Durée d'une note

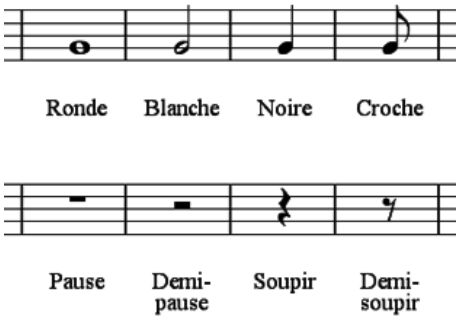
Chacune des 7 notes peut prendre une forme parmi 4 figures : la ronde (r), la blanche (b), la noire (n) et la croche c). Celles-ci déterminent sa durée qui se définit comme étant chaque figure de note vaut toujours le double de celle qui la suit. Ainsi :

— 1 ronde = 2 blanches
— 1 blanche = 2 noires
— 1 noire = 2 croches

Dans ce projet, nous supposons que la durée de la croche est égale à 125 ms.

3.4 Les silences

Dans une partition, nous comptons 4 types de pauses dont les durées sont équivalentes aux figures selon l'ordre suivant :



3.5 Le point de prolongation

Enfin, le point de prolongation est un signe placé après une figure de note ou un silence. Il permet de prolonger la durée de cette figure ou cette pause de la moitié.

4 Codage des notes

Les partitions écrites sur des portées traditionnelles ou sous la forme de noms de notes ne sont pas toujours faciles à lire pour un non connaisseur en musique, encore moins pour un ordinateur. La conversion de cette écriture dans un format numérique peut être une des solutions à ce problème. De plus, une série de notes sous forme numérique permet d'employer facilement des outils arithmétiques pour cacluler de nouvelles séries.
Un exemple de codage qu'on vous propose d'adopter dans ce projet est le suivant :

Do	Ré	Mi	Fa	Sol	La	Si
1	2	3	4	5	6	7

5 Outils de réalisation

5.1 Mathématiques

Les notes musicales étant codées numériquement, les opérations suivantes sont possibles :

1. **La transposition :** Il suffit d'ajouter une même valeur k à chacun des éléments de la série. La règle à suivre ici est la suivante :

Pour une série de m notes réparties sur l'intervalle [0, L-1] de longueur L : $(s_1, s_2, .., s_m)$, $T_k(s_1, s_2, .., s_m) = ((s_1 + k) \mod L, (s_2 + k) \mod L, .., (s_m + k) \mod L)$

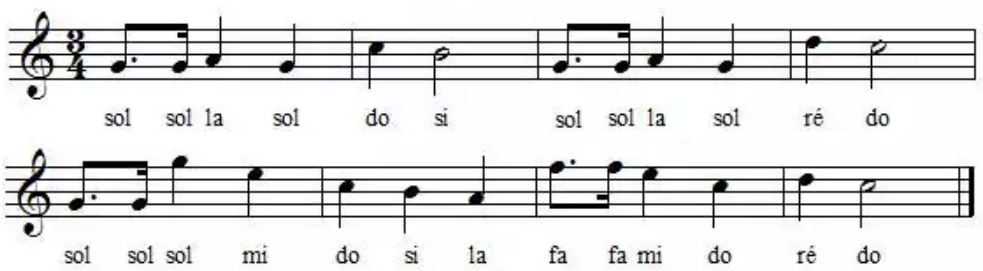
Exemple :
Pour une série de notes dans l'intervalle $[0, 11]$ et $k = 5$:
$T_5(0, 1, 3, 9, 2, 11, 4, 10, 7, 8, 5, 6) \rightarrow (5, 6, 8, 2, 7, 4, 9, 3, 0, 1, 10, 11)$

2. **L'inversion :** En arithmétique modulaire, inverser un nombre appartenant à l'intervalle $[0, L-1]$ revient à lui assigner la différence entre sa valeur et L : $(L - S_i) \mod L$.

   Exemple :
   Pour une série de notes dans l'intervalle $[0, 11]$:
   $I(0, 1, 3, 9, 2, 11, 4, 10, 7, 8, 5, 6) \rightarrow (0, 11, 9, 3, 10, 1, 8, 2, 5, 4, 7, 6)$

3. **Chaînes de Markov**
   L'enseignement de la composition musicale se fait souvent en demandant aux élèves de copier une œuvre d'un grand compositeur. Ceci permettra de l'analyser en déterminant les notes utilisées, le nombres de tons, le nombres de notes graves / aigües, ...etc.

   Exemple :
   Analysons la partition de "Joyeux anniversaire" ci-dessous et essayons de faire quelques statistiques dessus :



   Le nombre d'apparitions de chaque note est donné dans le tableau suivant :

Sol	La	Si	Do	Ré	Mi	Fa
9	3	2	5	2	2	2

   Si nous souhaitons créer une partition similaire, cette information ne nous aidera pas beaucoup. Elle permet de recréer l'ambiance générale mais elle n'est pas sensible au contexte. En effet, avec cette méthode, une partition similaire serait une partition où la distribution des notes et le chaînage entre elles ne sont pas pris en considération (Ex : une partition qui commence par 5 *Do*) .
   Les "Chaînes de Markov" sont un outil mathématique qui propose une solution approchée à ce problème. La technique consiste à étudier la succession des "états" différents que peut prendre un système puis de les reproduire statistiquement. En d'autres termes, en l'appliquant à la composition musicale, elle permet d'indiquer les choix des notes qui suivent l'apparition d'une certaine note.

   Exemple :
   En reprenant la partition précédente, nous représentons la transition de chaque type de note dans le Tableau 1.
   L'anniversaire de Markov ici va se créer comme suit :

		Note suivante							Total
		Sol	Fa	Mi	Ré	Do	Si	La	
Note	Sol	4	0	1	1	1	0	2	**9**
	Fa	0	1	1	0	0	0	0	**2**
	Mi	0	0	0	0	2	0	0	**2**
	Ré	0	0	0	0	2	0	0	**2**
	Do	2	0	0	1	0	2	0	**5**
	Si	1	0	0	0	0	0	1	**2**
	La	2	1	0	0	0	0	0	**3**

TABLE 1 – Matrice des notes successeurs

Version 1 : Sans tenir compte du nombre d'occurrence des notes

(a) Choisir la première note de manière aléatoire ou considérer la première note de la partition originale (Ex : Sol) ;

(b) Voir dans le tableau les notes qui lui succèdent (Ex : pour la note Sol, les successeurs sont : $Sol, Mi, Ré, Do, La$) ;

(c) Choisir une note aléatoire parmi ces successeurs ;

(d) Réitérer ce processus, autant de fois que nécessaire, à partir de l'étape b) en considérant à chaque fois les successeurs de la dernière note sélectionnée.

Version 2 : En tenant compte du nombre d'occurrence des notes

(a) Choisir comme première note celle ayant le nombre d'occurrence le plus élevé. Dans l'exemple c'est la note Sol car elle apparait $\frac{9}{25}$ où 25 est le nombre total de notes dans la partition choisie ;

(b) Voir dans le tableau les notes qui succèdent à la note sélectionnée (Ex : pour la note Sol, les successeurs sont : $4Sol, 1Mi, 1Ré, 1Do, 2La$) ;

(c) Stocker ces notes successeurs dans un tableau en considérant leurs nombres d'occurrences. Ex : Pour la note Sol, le tableau des successeurs est :
$succ = \{Sol, Sol, Sol, Sol, Mi, Ré, Do, La, La\}$

(d) Choisir aléatoirement une note du tableau des successeurs ;

(e) Réitérer ce processus, autant de fois que nécessaire, à partir de l'étape b) en considérant à chaque fois les successeurs de la dernière note sélectionnée.

Pour ce projet :

Bien distinguer deux situations :

— **Application de Markov sur une partition :** Cela signifie que l'analyse statistique ne concerne qu'une seule mélodie pour créer une partition similaire.

— **Application de Markov sur la base de données :** Cela signifie que l'analyse statistique concerne l'ensemble de la base de données. Il s'agit de déterminer les successeurs d'une note données en parcourant l'ensemble des partitions du fichiers. Le résultat sera une nouvelle partition du même style musical que les mélodies de la base de données.

5.2 Programmation

Et pour passer à la réalisation, voici quelques indications qui peuvent vous aider pour démarrer votre projet :

5.2.1 Quelques fonctions utiles

Le code de votre projet doit nécessairement être segmenté en fonctions. Pour vous aider, voici ci-dessous une liste non exhaustive de fonctions à envisager :

- Une fonction **calc_frequency(notes, frequences)** qui prend en entrée une liste de notes et une liste de fréquences et qui retourne une variable de type dictionnaire associant à chaque note une fréquence ;
- Une fonction **calc_duration(figures, d0)** qui prend en entrée la liste des 4 figures et la durée de la croche $d0$. Elle calcule les durées des autres figures et retourne un dictionnaire associant à chaque figure une durée.
- Une fonction **read_line_file(f, num)** qui prend en paramètres un nom de fichier et un numéro de ligne et qui retourne le contenu de la ligne en question.
- Une fonction **read_sheet** qui à partir d'une ligne du fichier extrait les notes, les figures, les silences et les points de prolongation et construit une séquence de fréquences et de durée qu'elle retourne en sortie.

  Exemple :
  ligne = "SOLc p Zc SOLn LAn SOLn DOn"
  $freq\_seq = [396, -1, 396, 440, 396, 264]$
  $duration\_seq = [187.5, -125, 250, 250, 250, 250]$

  NB : Dans l'exemple, l'on attribut la fréquence -1 à un silence et on marque sa durée par une valeur négative.

- Une fonction **play_sheet** qui à partir d'une séquence de fréquences et de durées, appelle les fonctions **sound** et **sleep** pour lire la partition musicale.

Cependant, plusieurs fonctions additionnelles sont nécessaire pour répondre aux objectifs **A**, **B**, **C** décrits dans la section 2.

5.2.2 Manipuler les fichiers en Python

Le fichier "partitions.txt" fourni est formaté comme suit :

- Il s'agit d'un fichier texte ;
- Il est composé de plusieurs partitions ;
- Chaque partition occupe deux lignes ;
  - Une première ligne débutant par le caractère # suivi d'un numéro séquentiel et le titre de la partition ;
  - Une deuxième ligne listant les notes , leurs figures et les silences en utilisant les caractères suivants :

Voici les fonctions usuelles pour manipuler les fichiers en Python :

Pour ouvrir un fichier en lecture

```
file = open("partitions.txt", "r") # le "r" ici indique lecture seule
```

Les notes	DO, RE, MI, FA, SOL, LA, SI
Les figures	r (ronde) , b (blanche), n (noire), c (croche)
Le silence	Z
Le point	p
Exemples	- DOr : Note Do de la durée d'une ronde - Zc : Silence de la durée d'une croche - p : Prolonger la durée de la note précédente

Lire toutes les lignes d'un fichier dans une liste

```
lignes = file.readlines()
for ligne in lignes:
    print(ligne)
```

Lire le fichier ligne par ligne

```
ligne = file.readline()
while ligne != "":
    print(ligne)
    ligne = filein.readline()
```

Ecrire dans un fichier

```
lignes = ["DO RE MI", "FA SOL", "LA SI"]
file = open("partitions.txt", "w") # "w" =  ouvert en ecriture
for ligne in lignes:
    file.write(ligne)
```

Fermer un fichier

```
file.close()
```

5.2.3 Lecture des notes musicales en Python

Afin de jouer vos partitions musicales, nous vous fournissons la fonction **sound (freq, duration)** ci-dessous. Elle prend en entrée une fréquence (en Hz) et sa durée (en secondes) et retourne le son joué par la note. L'utilisation de cette fonction requiert l'installation des deux bibliothèques : **simpleaudio** et **numpy**.
Par ailleurs, les silences sont marqués par la fonction **sleep(duration)** de la bibliothèque **time**.

Exemple d'utilisation :
Le code suivant permet la lecture de deux notes séparées par un silence :

```
from time import sleep
import numpy as np
import simpleaudio as sa

def sound(freq,duration):
    # get timesteps for each sample, "duration" is note duration in seconds
    sample_rate = 44100
    t = np.linspace(0, duration, int(duration*sample_rate), False)
    # generate sine wave tone
    tone = np.sin(freq * t * (6) * np.pi)
    # normalize to 24-bit range
    tone *= 8388607 / np.max(np.abs(tone))
    # convert to 32-bit data
    tone = tone.astype(np.int32)
    # convert from 32-bit to 24-bit by building a new byte buffer,
    # skipping every fourth bit
    # note: this also works for 2-channel audio
    i = 0
    byte_array = []
    for b in tone.tobytes():
        if i % 4 != 3:
```

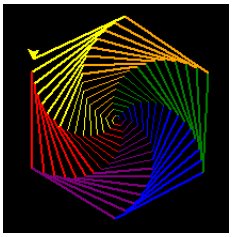
```
                byte_array.append(b)
            i += 1
        audio = bytearray(byte_array)
        # start playback
        play_obj = sa.play_buffer(audio,1, 3, sample_rate)
        # wait for playback to finish before exiting
        play_obj.wait_done()

sound(440, 0.5) # jouer la note LA sur 500 ms
sleep(0.5) # marquer un silence de 500 ms
sound(264, 0.25) # jouer la note DO sur 250 ms
```

5.2.4 Interface graphique en Python

Afin d'utiliser des figures graphiques qui accompagnent la lecture des partitions musicales, la librairie **Turtle** est vivement recommandée.

Exemple d'utilisation :

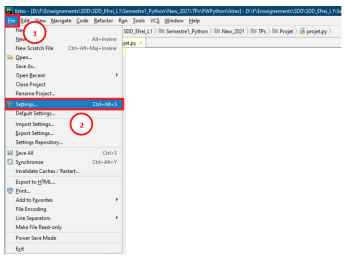
```
from time import sleep
import numpy as np
import simpleaudio as sa
import turtle as tr
notes=[262,294,330,262,262,294,330,262,330,349]
duration=[0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5]
d=0
tr.bgcolor("black")
colors=["red", "purple", "blue", "green", "orange", "yellow"]
x=0
for note in notes:
    sound(note, (int(duration[d])+3)*68)
    d+=1
    tr.up()
    tr.color(colors[x%6])
    tr.width(x/100+1)
    tr.down()
    tr.forward(x)
    tr.left(59)
    x=(x+1)%360
tr.exitonclick()
```

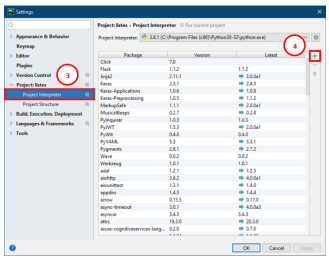


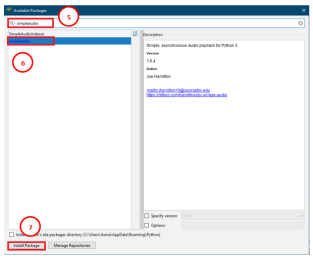
5.3 Installation d'une librairie sur PyCharm

Avant d'importer certaines libraries standard en python, il faut d'abord les installer. Voici les étapes à suivre pour installer une librairie sur PyCharm.

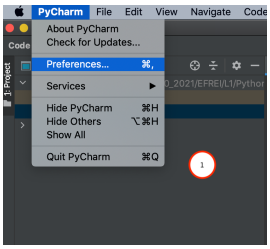
5.3.1 Sous Windows

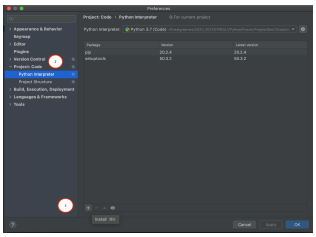


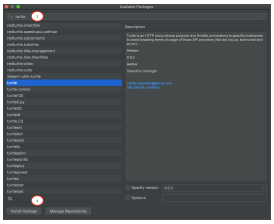




5.3.2 Sous McOS







6 Consignes générales

1. Le projet est à réaliser en binôme (monôme possible). Un seul groupe à trois étudiants pourrait être accepté dans le cas d'une imparité dans le groupe.
2. La remise de votre travail doit inclure :
   - Un fichier .zip contenant l'ensemble des fichiers sources.
   - Un rapport de 10 à 15 pages permettant de présenter :
     - vos solutions (en langage algorithmique) et vos choix de structures de données.
     - les difficultés rencontrées
     - les enseignements de ce projet
     - perspectives d'amélioration de votre rendu
3. Tout dossier remis doit être renommé comme suit : NOM1_NOM2 (avec NOM1 et NOM2 : noms des étudiants qui ont réalisé le projet).
4. Le projet est à rendre sur Moodle.
5. Les travaux sont à rendre au plus tard le 07/01/2020 à 23 :59.
6. La note finale du projet sera composée de :
   (a) La note du code
   (b) La note du rapport
   (c) La note de soutenance.

7 Planning

- Samedi 28/11/2020 : Lancement du projet (publication sur moodle)
- Semaine du 7/12/2020 : Suivi du projet
- Semaine du 04/01/2021 : Soutenances du projet