Projet Programmation en Python

L'Harmonie est numérique

Senechal Morgan

Sebbane Ryan

Groupe C

Promo 2025

Introduction

Depuis tout temps l'homme écoute et réalise de la musique pour son confort personnel. Au cours des siècles passés, la musique n'a fait qu'évoluer passant d'un simple instrument manuel à un logiciel informatique. Dans le cadre de l'enseignement de programmation en python nous avons dû réaliser un projet dont l'objectif et de créer un programme permettant de lire une partition de musique. Pour atteindre cet objectif, nous allons voir les différents algorithmes élaborés pour la fonctionnalité et l'application du programme et enfin l'interface graphique qui nous permettra d'ajouter un élément visuel à notre lecture.

I Présentation des Algorithme.

1)Présentation de l'algorithme CodeNote :

Présentation de la fonction :

Fonction Diconote(note :caractère, n :entier)

Paramètre copies : note,n

Paramètre modifier : vide

Variable local :i,j :compteur entier

Début :

```
    Nnote={} #dictionnaire qui va prendre les notes et les nombres associés aux notes.
    Pour i allant de 1 a longueur de note : #boucle qui parcourt la longueur de la liste note.
        Pour j allant de 1 a longueur de n : #Boucle qui parcourt la longueur de la liste de n.
            Si i==j : #Condition qui va permettre d'établir s'il s'agit du chiffre correspondant à la note.
                Nnote[note[i]]=n[j] #prendre la note et le chiffre dans le dictionnaire.
    Retourner Nnote #retourne le dictionnaire au programme.
```

Présentation de l'algorithme :

note=[« Do », «Ré », « Mi », « Fa », « Sol » , « La », « Si »]

n=[1,2,3,4,5,6,7]

afficher(Diconote(note,n)) #Appel à la fonction Diconote.

fin

#La liste note et n sont rangés en fonction de leur occurrence (Do correspond à 1…). Cela nous permet à l'aide de la fonction de pouvoir immédiatement trier les éléments dans le dictionnaire.

2)Présentation de l'algorithme Calc_frequency :

Présentation de la fonction :

Fonction Calc_frequency(note :caractère, frequence:entier)

Paramètre copies : note,frequence

Paramètre modifié : vide

Variable local : i,j:compteur entier

Début :

Fnote={} #Stockage des notes et fréquences dans un dictionnaire.

Pour i allant de 1 à longueur de note : # Boucle qui parcourt la liste de notes.

Pour j allant de 1 à longueur de frequence #Boucle qui parcourt la liste de fréquences

Si i==j #condition qui permet de ranger les notes avec leurs fréquences

Fnote[note[i]]=fréquences[j] #Range les notes avec leur bonnes fréquences.

Retourner Fnote #retourne le dictionnaire Fnote à l'algorithme.

Présentation de l'Algorithme :

Algorithme : freqnote

Var :

note : caractère

frequence : entier

début:

note=["Do","Ré,"Mi","Fa","Sol","La","Si"]

frequence=[264,297,330,352,396,440,495]

#Les notes et les fréquences sont rangés en fonction de leur occurrence (la fréquence 264hz correspond à la note Do…). Cela nous permet de trier immédiatement chacune des note et fréquence dans le dictionnaire à l'aide de la fonction.

afficher(calc_frequency(note,frequence)) #Appelle la fonction Calc_fréquency.

fin

3)Présentation de l'algorithme Calc_duration :

Présentation de la fonction :

Fonction Calc_duration(figures :caractére,d0 :entier)

Paramètre copié : figure,d0

Paramètre modifié : vide

Variable local : i,j :compteur entier

Début :

Fd0={} #Stockage des durées et figures dans un dictionnaire.

pour i allant de 1 a longueur de figures : #boucle qui parcourt la longueur de la figure.

pour j allant de 1 a longueur de d0 #boucle qui parcourt la longueur de la durée.

Si i==j #condition qui associe les bonnes figures aux bonnes durées.

Fd0[figures[i]]=d0[j] #Condition validée donc le dictionnaire prend cette forme.

retourner Fd0 #retourne le dictionnaire à l'algorithme.

Présentation de l'Algorithme :

Algorithme : Calc_duration

Var :

figures : caractère

d0 : entier

Début :

figures=[« r », « b », « n », « c »]

d0=[1000,500,250,125]

afficher(calc_duration(figures,d0)) #Appel à la fonction

fin

Les figures et les durations sont rangés en fonction de leur occurrence (ronde a pour fréquence 1000…). Cela nous permet de pouvoir ranger immédiatement dans le dictionnaire les bonnes figures avec leur bonne fréquences grâce à la fonction.

4)Présentation de l'algorithme read_line

Présentation de la fonction :

Fonction read_line(fichier :chaîne de caractère, num) :

Paramètre copié : fichier, num

Paramètre modifié : vide

Variable local : i :compteur entier

Débuts :

c=0 #correspond au compteur de la ligne.

cpt=0 #compteur qui va arrêter le compteur du comptage de ligne.

file=open(fichier, « r ») #ouverture du fichier.

lignes=file.readlines() #lire toutes les lignes du fichier.

pour i allant de 1 a la lignes : #boucle qui permet de parcourir les lignes.

cpt=cpt+1 #compteur qui prend en compte le nombre de ligne.

Si cpt=num #condition qui permet de prendre notre ligne.

c=i #prend la ligne.

Retourner c #retourne la ligne à l'algorithme.

Présentation de l'algorithme :

Algorithme : read_line

Var :

Fichier :caractère

num :entier

afficher(« saisir le nom de votre fichier »)

saisir(fichier)

afficher(« saisir une ligne »)

saisir(num)

afficher(read_line("partitions.txt",num)) #appel de la fonction read_line.

fin

Problème rencontré l'ors de cet algorithme :

Au début nous avons eu du mal à trouver comment ouvrir le fichier texte via pycharm. Après plusieurs minutes de réflexion, nous avons trouvé d'où venait le problème, celui-ci venait du fait que le fichier txt n'était pas incrémenté dans les fichiers de pycharm.

5)Présentation de l'algorithme read_sheet

Présentation de la fonction :

Fonction read_sheet(num :entier)

Paramètre copies : num

Paramètre modifies : vide

Variable local : i :compteur entier

Début :

c=0 #Utilisation de la fonction read_line pour ouvrir le fichier texte et lire la ligne.

cpt=0

file=open(FICHIER, « r »)

lignes=file.readlines()

pour i allant a lignes :

cpt=cpt+1

Si cpt==num :

c=i

```
space=true # verification.
every_note=[] #liste où l'on stocke toute les notes de la ligne.
temp=  « » #pour que ce soit une chaine de caractère.
pour z dans longueur de c : #boucle qui parcoure la longueur de la ligne.
    elem=c[Z]
    if elem =/= '' and elem =/= \n: #condition qui demande que l'élément soit à l'intérieur de la chaine de caractère.
    si space==vrai #condition booléenne.
        temp=elem
    sinon : temp=temp+elem
    space=faux
    evey_note.append(temp)
freq=[] #liste qui prend les fréquences.
fig=[] #liste qui prend les figures.
pour note dans every_noye : #boucle qui prend les notes incluses dans la ligne.
    si « DO » dans  note : #condition des fréquences des notes dans la ligne.
        freq2=264
    sinon « RE » dans note :
        freq2=297
    sinon « MI » dans note :
        freq2=330
    sinon « FA » dans note :
        freq2=352
    sinon « SOL » dans note :
        freq2=396
```

```
        sinon « LA » dans note :
            freq2=440
        sinon « SI » dans note :
            freq2=495
        sinon « Z » dans note :
            freq2=-1
        if freq2=/=0 :
            freq.append(freq2)
        fig2=0
        si ‘c’ dans note :  #condition des figures des notes dans la ligne.
            fig2=125.0
        sinon ‘n’ dans note :
            fig2=125*2.0
        sinon ‘b’ dans note :
            fig2=125*4.0
        sinon ‘r’ dans note :
            fig2=125*8.0
        sinon ‘p’ ==note :
            fig[-1] += (fig[-1])/2
        si ‘Z ‘ dans note :
            fig2=-fig2
        si fig2 =/= 0 :
            fig.append(fig2)
finaly=[] #liste finale qui prend les note, fréquences et figures.
finaly.append(every_note) #finaly prend les notes avec .append.
finaly.append(freq) #finaly prend les frequence avec .append.
finaly.append(fig) #finaly prend les figures avec .append.
```

file.close() #le fichier se ferme.

Retourner finaly #retourne la valeur à l'algorithme.

Présentation de l'algorithme :

Algorithme : read_sheet

Var :

num :entier

début :

afficher(« saisir le numéro de la ligne souhaité »)

saisir(num)

afficher(read_sheet(num)) #appel à la fonction read_sheet.

Fin

Problème rencontré lors de l'élaboration de cet algorithme :

Au cours de la création de cet algorithme, nous avons rencontré diverses difficultés. La première était de pouvoir prendre les éléments de la ligne du fichier texte (ce que nous avons réussi à faire au bout de quelque séance). La deuxième était de pouvoir stocker dans une liste principale les trois sous liste (note, fréquence, durée) ce que nous avons réussi à faire en utilisant la commande .append pour rentrer les valeurs de ces trois listes dans la liste finale. Enfin nous nous sommes rendu compte que les notes apparaissent qu'une seule fois dans la liste, alors nous avons passé plusieurs heures pour résoudre ce problème qui venait de notre boucle for.

6)Présentation de l'algorithme play_sheet

Présentation de la fonction :

Fonction play_sheet(num :entier)

Paramètre copies : num

Paramètre modifies : vide

Variable local : i :compteur entier

Débuts :

c=read_sheet(num) #c est la variable qui prend la fonction read_sheet(num).

freq=c[1] #frequence prend la 1er occurrence.

fig=c[2] #fig prend la 2nd occurrence.

pour i dans la longueur de freq : #boucle qui prend la longueur de la freq.

frequency=freq[i] #frequency qui prend la valeur de freq[i].

duration=fig[i] #duration qui prend la valeur de fig[i].

si duration <0 and frequency=-1 : #condition qui permet déterminer si il s'agit d'un silence.

sleep.sleep((-duration/1000) #le silence.

sinon : sound(frequency,duration/1000) #autre condition cette fois ci pour les notes.

afficher(read_sheet(2))

play_sheet(24) #permet de lire la partition à la ligne choisie ici on lit la ligne 24.

Problème rencontré dans l'élaboration de l'algorithme :

Au cours de la création de cet algorithme nous avons rencontré plusieurs problèmes. Le premier était de pouvoir respecter les silences (nous avons réussi à résoudre ce problème grâce à l'utilisation de sleep venant du package time.). Le deuxième était dû à une mauvaise indentation de l'appel à la fonction sound (la fonction sound n'était pas bien incrémenté dans le programme).

7)Présentation de l'algorithme graph :

Présentation de la fonction :

Fonction graph(num :entier) :

Paramètre copies : vide

Paramètre modifies : vide

Variable local : i :compteur entier

Début :

c=read_sheet(num) #c prend la function read_sheet(num).

print(c) #on affiche c.

freq=c[1] #frequence prend la 1 occurrence de c.

fig=c[2] #figure prend la 2eme occurrence de c.

d=0 #d et initialisé a 0.

tr.bgcolor(“black”) #couleur de l’arrière-plan.

colors=[“red”,”purple”,”blue”,”green”,”orange”,”yellow”] #couleur du motif.

x=0 #x et initialisé à 0.

tr.speed(0) #rapidité du motif.

pour i allant a la longueur de freq : #boucle qui parcoure la longueur de freq.

frequency=freq[i] #frequency qui prend la valeur de freq[i].

duration=fig[i] #duration prend la valeur de fig[i].

si duration <0 ou frequency==-1 : #condition sur la duration et la frequence.

si duration <0 #condition sur la duration.

sleep.sleep((-1*duration/1000) #appel au package time.

sinon : sleep.sleep(duration/1000) #appel au package time.

sinon : sound(frequency,duration/1000) #apel au package simpleaudio.

d=d+1 #d prend d+1

tr.up() #position du motif.

tr.color(color[x%6]) #position du motif.

tr.width(x/100+1) #position du motif.

tr.down() #position du motif.

tr.forward(x*5) #position du motif.

tr.left(59) #position du motif.

x=(x+1)%360 #position du motif.

tr.exitonlick() #clické pour stoper.

graph(2) #changer la valeur de graphe pour lire une partition.

Présentation des programmes du projet :

1)Le codage des notes musicales :

```
import time as sleep
import numpy as np
import simpleaudio as sa
import turtle as tr



FICHIER="partitions.txt"

def Diconote(note,n): #On défini la fonction.
    Nnote = {} #Nnote est notre dictionnaire.
    for i in range(len(note)): #boucle qui parcoure la longueur de liste de
note.
        for j in range(len(n)): #boucle qui parcoure la longueur de la
liste de nombre.
            if i==j: #condition d'égalité des occurences.
                Nnote[note[i]] = n[j] #ce que prend le dictionnaire.
    return Nnote #retourne le dictionnaire Nnote au programme.

note = ["Do", "Ré", "Mi", "Fa", "Sol", "La", "Si"] # liste de note
préranger avec la liste des nombres n par leurs occurences.
n=[1,2,3,4,5,6,7]
print(Diconote(note,n)) #apelle la fonction Diconote.
```

Résultat : Après avoir réalisé le test de notre fonction Diconote, nous avons obtenu exactement ce que nous voulions, c'est-à-dire une fonction qui nous transmet un dictionnaire de note associé à des nombres.

```
{'Do': 1, 'Ré': 2, 'Mi': 3, 'Fa': 4, 'Sol': 5, 'La': 6, 'Si': 7}
```

2)L'attribution des fréquences aux notes :

```
def calc_frequency(note, frequences): #On defini la fonction.
    Fnote = {} #Fnote est notre dictionnaire.
    for i in range(len(note)): #Boucle qui parcoure la longueur de la liste
de note.
        for j in range(len(frequences)): #Boucle qui parcoure la longueur
de la liste de fréquence.
            if i==j:#condition d'égalité des occurences.
                Fnote[note[i]] = frequences[j] #ce que prend le
dictionnaire.
    return Fnote #retourne le dictionnaire Fnote au programme.

note = ["Do", "Ré", "Mi", "Fa", "Sol", "La", "Si"] #liste de note préranger
avec la liste des fréquence par leurs occurences.
frequences = [264, 297, 330, 352, 396, 440, 495]
print(calc_frequency(note, frequences)) #apelle à la fonction
calc_frequency.
```

Résultat : Après avoir réalisé le teste de notre fonction calc_frequency, nous avons obtenue exactement ce que nous voulions, c'est-à-dire une fonction qui nous transmet un dictionnaire de note associé à leur fréquence.

```
{'Do': 264, 'Ré': 297, 'Mi': 330, 'Fa': 352, 'Sol': 396, 'La': 440, 'Si': 495}
```

3)L'attribution des durées aux notes :

```
def calc_duration(figures,d0): #on définie la fonction.
    Fd0={} #Fd0 et notre dictionnaire.
    for i in range(len(figures)): #Boucle qui parcoure la longueur de la
liste des figures.
        for j in range(len(d0)): #Boucle qui parcoure la longueur de la
liste des durées.
            if i==j: #Condition d'égalité des occurences.
                Fd0[figures[i]]=d0[j] #ce que prend le dictionnaire.
    return Fd0 #retourne le dictionaire Fd0 au programme.
figures=["ronde","blanche","noir","croche"] #liste des figure prérangé avec
la liste des duré par leur occurence.
d0=[1000,500,250,125]
print(calc_duration(figures,d0)) #apelle à la fonction calc_duration.
```

Résultat : Après avoir réalisé le teste de notre fonction calc_duration, nous avons obtenue exactement ce que nous voulions, c'est-à-dire une fonction qui nous transmet un dictionnaire de figure associé à leur durée.

```
{'ronde': 1000, 'blanche': 500, 'noir': 250, 'croche': 125}
```

4) Lecture de la partition : fonction read_line :

```
def read_line(fichier,num): #on defini la fonction.
    c=0 #variable c qui prend la ligne.
```

```
    cpt=0 #compteur cpt qui compte les lignes.
    file=open(fichier,"r") #permet ouvrire le fichier.
    lignes=file.readlines() #lire les lignes du fichier.
    for i in lignes: #boucle qui parcoure les lignes.
        cpt+=1 #compteur qui prend le nombre de ligne.
        if cpt==num: #condition qui permet de prendre la ligne demandé.
            c=i #prend la ligne demandé.
    return c #retourne la ligne demandé c au programme.
fichier=input("saisir votre fichier") #saisir un fichier de type caractére.
num=int(input("saisir une ligne")) #saisir un numéro de ligne.
print(read_line("partitions.txt",num)) #apelle à la fonction read_line.
```

Résultat : Après avoir réalisé le teste de notre fonction read_line, nous avons obtenue exactement ce que nous voulions, c'est-à-dire une fonction qui nous fournie la ligne du fichier que l'on souhaite.

```
saisir votre fichier partitions.txt
saisir une ligne 2
SOLc p Zc SOLn LAn SOLn DOn Zc SIb SOLc p Zc SOLn LAn SOLn REn Zc DOb SOLc p Zc SOLn SOLn MIn DOn Zc SIn LAn FAc p Zc FAn MIn DOn REn DOr
```

5)Lecture de la partition avec liste de note,frequence et durée : fonction read_sheet :

```
def read_sheet(num):
    c = 0
    cpt = 0
    file = open(FICHIER, "r")


    lignes = file.readlines()      #utilisation de la fonction read_line.
    for i in lignes:
        cpt += 1
        if cpt == num:
            c = i

    space = True #verification.
    every_note = [] #stockage dans une liste.
    temp="" #pour que ce soit  une chaine de caractère.
    for z in range(len(c)): #boucle qui parcoure la longueur de la ligne
choisie.
        elem = c[z]
        if elem != " " and elem !="\n": #condition qui demande que
l'élément soit à l'intérieur de la chaine de caractére.
            if space == True : #condition booléene.
                temp = elem
            else:
                temp = temp+elem
            space = False
        else:
            space = True
            every_note.append(temp)


    freq = [] #liste de stockage des fréquences.
    fig = [] #liste de sotackage des figures.

    for note in every_note : #boucle qui prend note dans toute les notes de
de la ligne.
```

```
        # FREQUENCE
        freq2 = 0
        if "DO" in note: #condition des fréquence des notes dans la ligne.
            freq2 = 264
        elif "RE" in note:
            freq2 = 297
        elif "MI" in note:
            freq2 = 330
        elif "FA" in note:
            freq2 = 352
        elif "SOL" in note:
            freq2 = 396
        elif "LA" in note:
            freq2 = 440
        elif "SI" in note:
            freq2 = 495
        elif "Z" in note:
            freq2 = -1
        if freq2 != 0:
            freq.append(freq2)

        # FIGURE
        fig2 = 0
        if 'c' in note: #condition des figures des notes dans la liste.
            fig2 = 125.0
        elif 'n' in note:
            fig2 = 125*2.0
        elif 'b' in note:
            fig2 = 125*4.0
        elif 'r' in note:
            fig2 = 125*8.0
        elif 'p' == note:
            fig[-1] += (fig[-1])/2

        if 'Z' in note:
            fig2 = -fig2

        if fig2 !=0:
            fig.append(fig2)

    finaly = [] #liste qui prend toute les listes.
    finaly.append(every_note) #.append permet d'ajouter les élément a
finaly.
    finaly.append(freq)
    finaly.append(fig)
    file.close()
    return finaly #retourne la liste finaly au programme.

num=int(input("saisir une ligne")) #saisir un numéro de ligne.
print("voici: (notes,frequence, durée")
print(read_sheet(num)) #apelle la fonction read_sheet
```

Résultat : Après avoir réalisé le teste de notre fonction read_sheet, nous avons obtenue exactement ce que nous voulions, c'est-à-dire une fonction qui nous une liste contenant toutes les notes de la ligne suivit de leur fréquence et de leur durée.

```
saisir une ligne 2
voici: (notes,frequence, durée
[['SOLc', 'p', 'Zc', 'SOLn', 'LAn', 'SOLn', 'DOn', 'Zc', 'SIb', 'SOLc', 'p', 'Zc', 'SOLn', 'LAn', 'SOLn', 'REn', 'Zc', 'DOb', 'SOLc', 'p', 'Zc', 'SOLn', 'SOLn', 'MIn', 'DOn', 'Zc',
```

```
'Zc', 'SIn', 'LAn', 'FAc', 'p', 'Zc', 'FAn', 'MIn', 'DOn', 'REn', 'DOr'], [396, -1, 396, 440, 396, 264, -1, 495, 396, -1, 396, 440, 396, 297, -1, 264, 396, -1, 396, 396, 330, 264,
```

```
264, 396, -1, 396, 396, 330, 264, -1, 495, 440, 352, -1, 352, 330, 264, 297, 264], [187.5, -125.0, 250.0, 250.0, 250.0, 250.0, -125.0, 500.0, 187.5, -125.0, 250.0, 250.0, 250.0,
```

```
-125.0, 500.0, 187.5, -125.0, 250.0, 250.0, 250.0, 250.0, -125.0, 250.0, 250.0, 187.5, -125.0, 250.0, 250.0, 250.0, 250.0, 1000.0]]
```

6) Lecture du son : fonction sound :

```python
def sound (freq,duration) : # la fonction sound qui permet d'émettre un son à partir d'une frequence et d'un temps²
    sample_rate = 44100
    t = (np.linspace(0.0, duration, num=int(duration*sample_rate), endpoint=False))
    tone = np.sin(freq*t*(6)*np.pi)
    tone *= 8388607/np.max(np.abs(tone))
    tone = tone.astype(np.int32)
    i = 0
    byte_array = []
    for b in tone.tobytes():
        if i % 4 != 3:
            byte_array.append(b)
        i += 1
    audio = bytearray(byte_array)
    play_obj = sa.play_buffer(audio, 1, 3, sample_rate)
    play_obj.wait_done()
```

Résultat : Cette fonction nous à était donner et nous permet de lire la partition avec du son.

7) Lecture de la partition avec son : fonction play_sheet :

```python
def play_sheet(num):
    c = read_sheet(num) #c est la variable qui va prendre notre fonction read_sheet.
    freq = c[1] #freq vas prendre le 1er élément comme fréquence.
    fig = c[2] #fig vas prendre le second élément comme figure.
    for i in range(len(freq)): #boucle qui parcoure la longeur de la frequence.
        frequency = freq[i]
        duration = fig[i]
        if duration < 0 and frequency == -1: #condition qui permet établire le silence.
            sleep.sleep((-duration)/1000)
        else: #sinon si ce n'est pas un silence alors ont prend la note.
            sound(frequency, duration/1000)
```

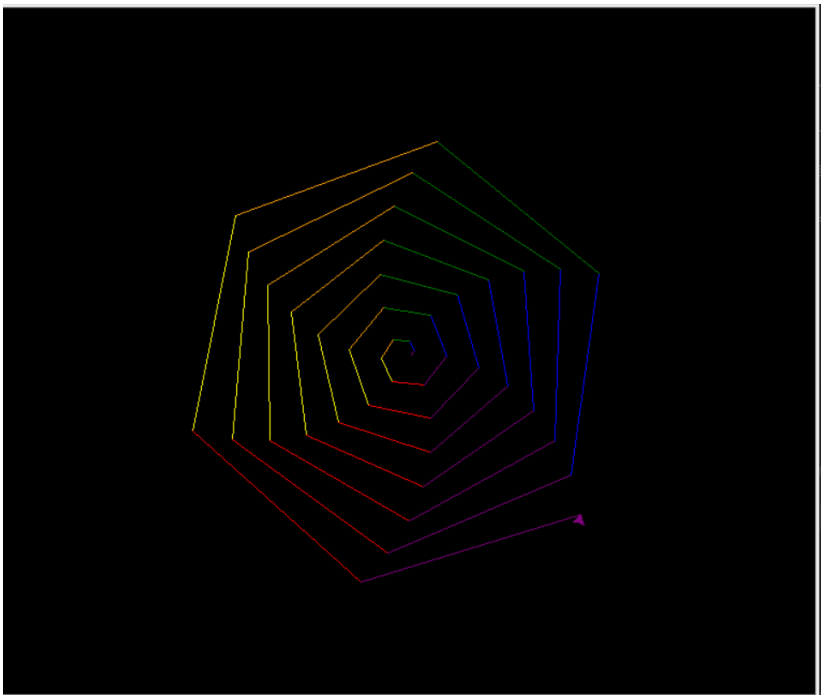
```
print(read_sheet(2))

#sound(396, 0.1875)
play_sheet(24) #commande qui permet de choisir la ligne a jouer.
```

Résultat : Après avoir réalisé le teste de notre fonction play_sheet, nous avons obtenue exactement ce que nous voulions, c'est-à-dire une fonction qui lit notre partition avec du son.

8) Lecture de la partition avec image : fonction graph :

```
def graph(num):
    c = read_sheet(num) #variable qui prend la fonction read_sheet(num).
    print(c) #afficher c.
    freq = c[1] #freq prend la premier occurence de c.
    fig = c[2] #fig prend la 2eme occurence de c.
    d = 0 #d est initialisé à 0.
    tr.bgcolor("black") #couleur de l'arrière-plan.
    colors = ["red", "purple", "blue", "green", "orange", "yellow"]
#couleur du motif.
    x = 0 #x est initialisé à 0.
    tr.speed(0) #vitesse du motif.
    for i in range(len(freq)): #boucle qui parcoure la longueur de freq.
        frequency = freq[i] #frequency prend la valeur de freq[i].
        duration = fig[i] #duration prend la valeur de fig[i].
        if duration < 0 or frequency == -1: #condition sur la durée et la
frequence.
            if duration < 0: #condition sur la durée.
                sleep.sleep((-1*duration)/1000) #appel au package time.
            else:
                sleep.sleep(duration/1000)#appel au package time.
        else:
            sound(frequency, duration/1000) #appel au package simpleaudio.
            d += 1 #d prend d+1.
            tr.up() #position du motf.
            tr.color(colors[x % 6]) #position du motif.
            tr.width(x / 100 + 1) #position du motif.
            tr.down() #position du motif.
            tr.forward(x*5)#position du motif.
            tr.left(59)#position du motif.
            x = (x + 1) % 360#position du motif.
    tr.exitonclick()#clicker sur le motif pour stoper.

graph(2) #changer la valeur de graph pour changer de partition.
```

Résultat : Après avoir réalisé le teste de notre fonction graph, nous avons obtenue exactement ce que nous voulions, c'est-à-dire une fonction qui nous lit la partition suivit d'un élément graphique.

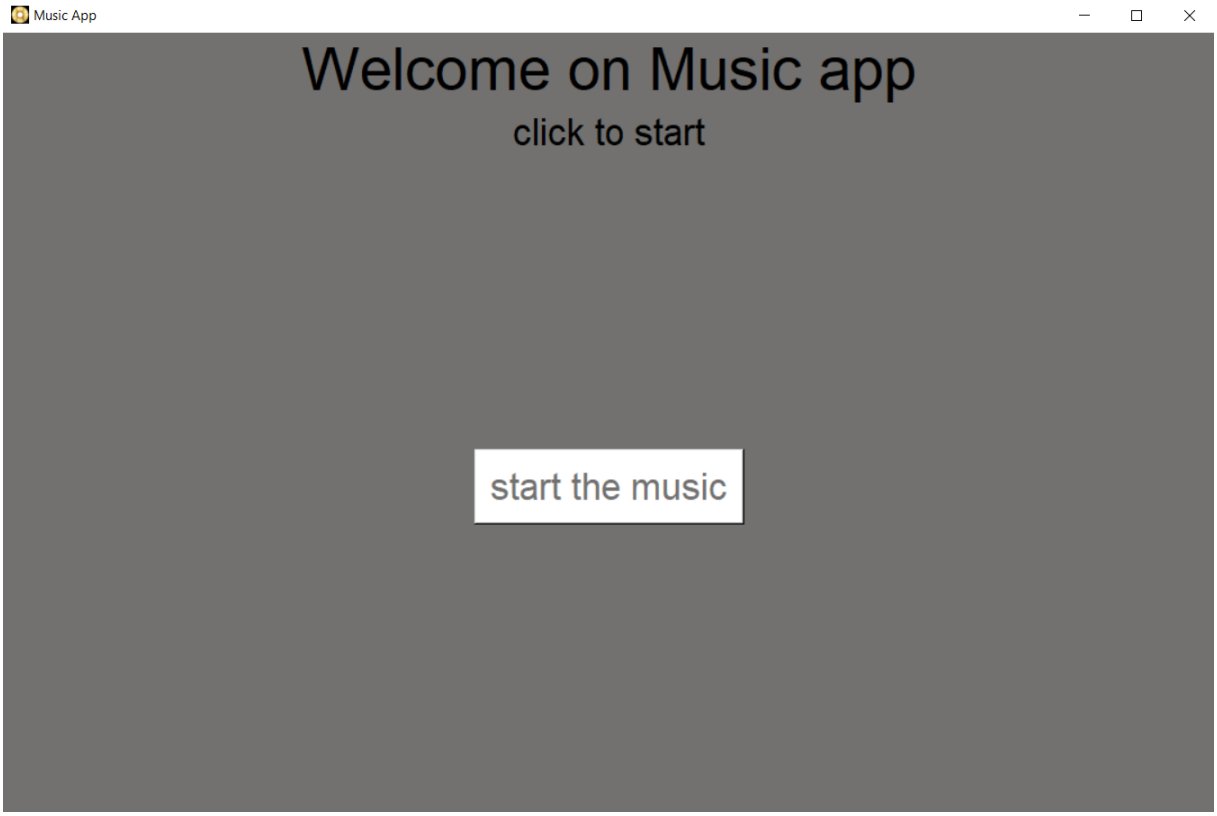


Interface Tkinter non finie :

```
#INTERFACE GRAPHIQUE PAR TKINTER

from tkinter import*
window = Tk()
window.title("Music App")
window.geometry("1080x720")
window.minsize(480, 360)
window.iconbitmap("3378993_87aa7.ico")
window.config(background='#727170')

# frame pour centré les textes.

frame = Frame(window, bg='#727170')

# ajouter un  Principal texte
label_title = Label(window, text="Welcome on Music app", font=("Helvetica",
40), bg='#727170', fg='black')
label_title.pack()  # permet de mettre au millieu le texte meme en
modifient la fenetre.

# ajouter un texte secondaire
label_subtitle = Label(window, text="click to start", font=("Helvetica",
25), bg='#727170', fg='black')
label_subtitle.pack()

# ajoute de bouton

play_button = Button(frame, text="start the music", font=("Helvetica", 25),
bg='white', fg='#727170')
play_button.pack()
frame.pack(expand=YES)
```

```
window.mainloop()
```

Résultat : Ce programme nous permet d'ouvrir une fenêtre d'application pour l'utilisateur et lui propose un bouton start (nous n'avons pas réussi à terminer l'interface).



Problème rencontré :

L'ors de la réalisation de notre projet, nous avons pu rencontrer diverses difficultés. En effet nous avons rencontré certaine difficulté sur la création de certain de nos algorithmes tel que (read_line,read_sheet,play_sheet). Malgré cela, nous avons toute de même réussi à aboutir aux l'algorithmes finaux voulue. Cependant il y à certain algorithme que nous n'avons pas réussi à élaborer tel que la transposition, l'inversion, chaine de Markov et la base de données. Nous avons passé pas mal de temps du ces 4 algorithmes mais nous n'avons pas réussi à y aboutir. Enfin nous avons essayé de créer une interface graphique pour l'utilisateur en utilisant le package Tkinter mais nous n'avons pas réussi à faire incrémenter la partition dans le programme.

Conclusion

Ce projet nous a été très utile car il nous a plongé dans le vif du travail en groupe. En effet grâce à cela, nous avons appris à planifier notre travaille chaque semaine, a se partagé diverses tâches pour optimiser au mieux notre travail tout en se communiquant nos avancé par des meetings réguliers. De plus ce projet nous a permis d'améliorer nos compétences sur le langage python tel que l'organisation de notre code en commençant en premier lieu par la création de l'algorithme jusqu'au programme. Par ailleurs cela nous a aussi apporté certaine connaissance sur la musique que nous n'avions pas. Enfin se projet nous a était très bénéfique sur plusieurs points tel que l'organisation de notre travail en équipe ou encore l'application de nos connaissances via le langage python pour répondre au mieux à la demande du projet.