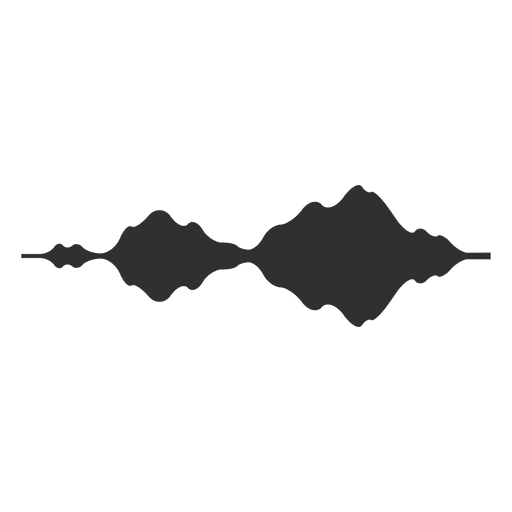


Réalisation d’un outil de reconnaissance musicale

Projet Malece



Cédric Cally-Caballero, Delecourt Léo, Mackenson Greffin

*c.cally-caballero@univ-lyon2.fr, leo.delecourt@univ-lyon2.fr, mackenson.greffin@univ-lyon2.fr*

8 décembre 2020

**Introduction et rappel du cahier des charges**

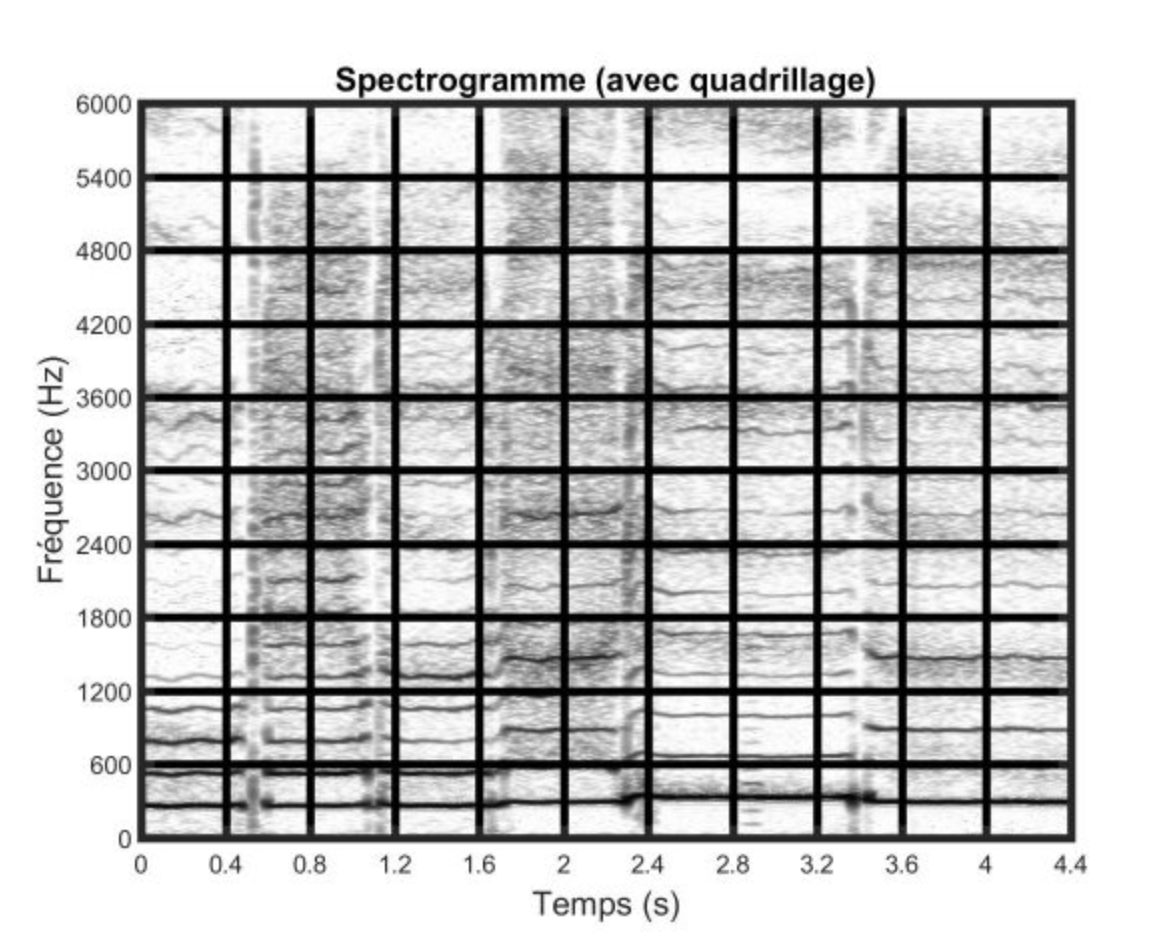
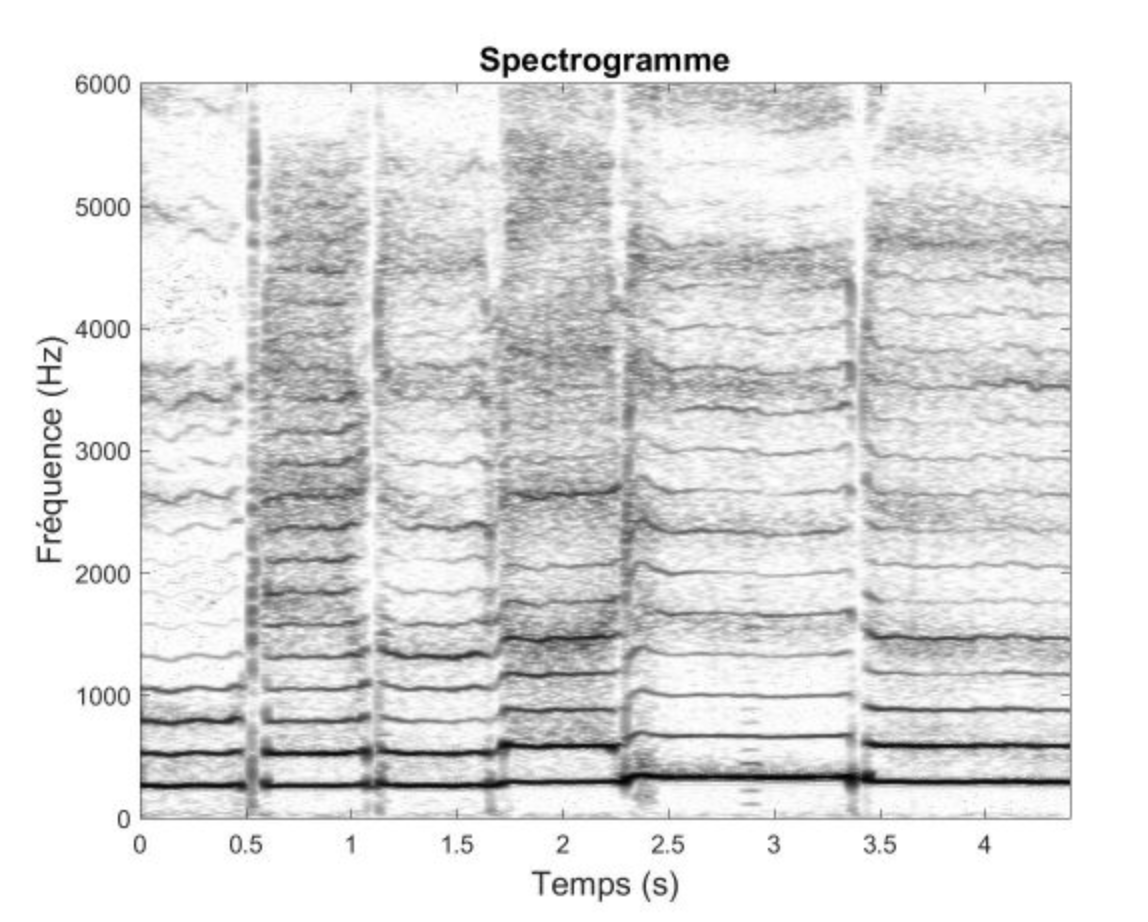
Le projet Malece correspond au projet de reconnaissance de fichier audio qui s’inscrit dans le cadre du cours de Python. Ce projet consiste en la création d’un outil de python capable de reconnaitre un fichier audio à partir d’un enregistrement via un microphone. Quatre axes clés composent le cahier des charges ce projet. Concernant les deux premiers axe, l’outil python doit pouvoir, à partir d’un enregistrement audio via un microphone d’une durée maximale de 15 secondes fournir dans la mesure du possible les informations suivantes titre, nom de l’auteur, titre de l’album et date de sortie. Le troisième axe du cahier des charges autorise l’utilisation de package de traitement de fichier audio, le choix effectué pour ce projet sera détaillé plus loin. Enfin le quatrième axe du cahier des charges concerne la performance de l’outil. C’est-à-dire que la reconnaissance du fichier audio doit être réalisée dans un laps de temps raisonnable.

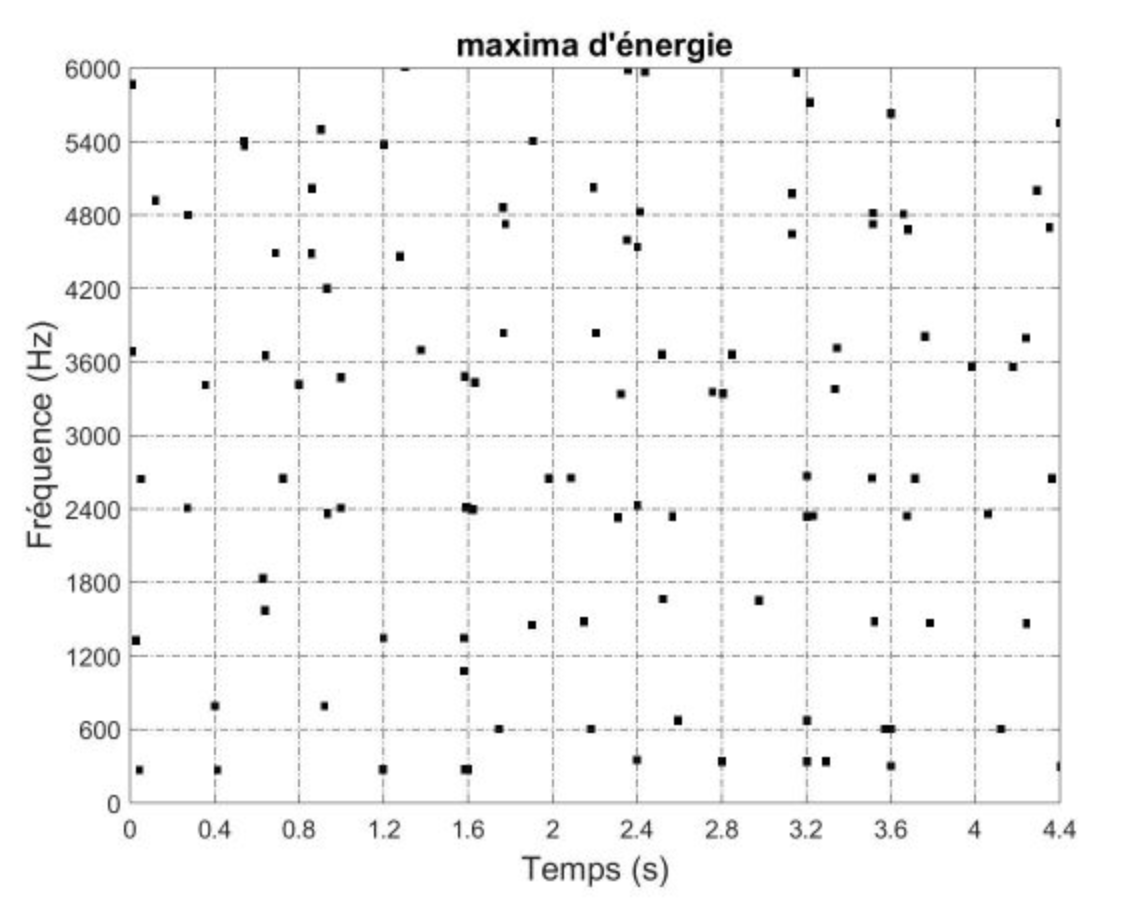
Ce rapport est décomposé en quatre sections, la première aborde une brève revue de la littérature sur les outils de traitement de fichier audio notamment sur python. La deuxième section présente une vue d’ensemble de la méthode utilisée. Dans une troisième section, les parties les plus importantes du code de Malece seront commentées. Enfin la dernière section discutera des difficultés rencontrées, des limites des outils utilisés et les pistes d’amélioration envisageables.

# Brève revue de littérature sur les approches de reconnaissance musicales (LiBROSA : un outil de traitement de fichier audio sur Python)

Les applications de reconnaissance musicales sont très nombreuses et la plus célèbre est Shazam. Quelque soit la manière d’implémenter l’outil de reconnaissance musicale, les méthodes semblent toutes convergées. On dispose d’une base de données de fichiers audio auquel sont rattachés des méta données (titre, interprète, nom de l’album…). Les musiques sont stockées sous forme d’empreintes qui sont générées à partir de spectrogrammes. Un spectrogramme donne pour chaque instant donné l’amplitude ou l’énergie sonore et une fréquence. Le spectrogramme est unique à chaque signal sonore, car il montre les variations des fréquences du fichier audio au cours du temps, il est donc possible de créer une signature propre à chaque fichier audio à partir d’un spectrogramme : c’est l’empreinte sonore. Cependant, il arrive souvent que les informations se répètent dans un spectrogramme (à cause des refrains d’une musique par exemple). Par ailleurs, il n’est pas du tout impossible que certaines parties de deux spectrogrammes de deux fichiers audio différents soientt très similaires. Il faut donc générer une empreinte qui synthétisera les informations caractéristiques de chaque fichier audio de manière unique.

Il existe plusieurs manières de générer une empreinte musicale, et la performance de l’outil de reconnaissance dépend évidemment de la manière sont générées les empreintes. L’idée principale des applications de reconnaissance musicale consiste à découper le spectrogramme en intervalle de temps et de fréquence de largeur égale et de ne garder que pour chaque intervalle de temps et chaque plage de fréquences le point correspondant au maximum d’énergie.





Lorsque l’outil doit reconnaitre un fichier enregistré via un microphone, une nouvelle empreinte et générée et celle-ci est comparée à l’ensemble des empreintes de la base donnée. Une fois la comparaison terminée, les métas donnés correspondant au fichier le plus proche sont retournées par l’application.

Il existe plusieurs packages de python permettant de générer des spectrogrammes via un fichier audio. Le package LiBROSA est justement un package de python qui permet d’extraire des spectrogrammes à partir le fichier audio.

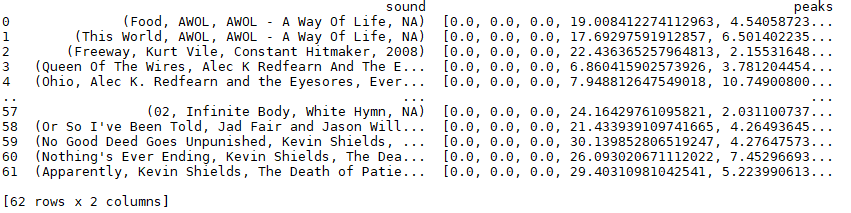
# Vision globale de la méthode et des outils utilisés

Pour qu’elle soit complète, la méthodologie utilisée doit répondre à plusieurs problématiques : créer une base de données des sons à reconnaître ; construire des spectrogrammes représentatifs de chaque musique ; être en mesure de comparer plusieurs spectrogrammes ; intégrer une interface graphique pour les utilisateurs. Dans cette section, nous allons détailler les solutions implémentées pour répondre à ces différents points.

**1. Création et chargement d’une base de données**

Afin de pouvoir détecter de quelle musique est extraite un échantillon, ce dernier est comparé à une « base » de musique. Celle-ci est initialement générée à partir d’un répertoire contenant des fichiers au format .mp3 - ce format a été retenu car il embarque des métadonnées telles que le nom d’artiste, le titre et l’année de la musique ainsi que l’album. Pour chacun de ces fichiers audio du répertoire, les métadonnées sont extraites sous forme de tuple, puis le spectrogramme est calculé (voir section suivante). Ces données sont alors chargées dans un DataFrame Pandas de deux colonnes (« sound » contenant le tuple de metadonnées et « peaks » contenant l’empreinte), chaque ligne correspondant à une musique. Ce DataFrame est également enregistré dans un fichier *pickle* (.pikl) afin de pouvoir être chargé rapidement au lancement du programme, et non reconstruit à chaque fois.

Enfin, de manière à pouvoir faire évoluer le programme dans le temps, la fonction de génération de la base de données peut-être utilisée pour ajouter des lignes à un fichier pickle précédemment créé.

Figure 1: Exemple de DataFrame utilisé pour la reconnaissance des musiques

**2. Création des empreintes**

Il s’agit du cœur du programme car c’est à partir de ces éléments que la reconnaissance va pouvoir opérer. Les empreintes sont des « signatures » calculées à partir des fichiers sons. Elles doivent être :

* **uniques** pour pouvoir différencier les différentes musiques
* **prédictibles et robustes** afin que deux musiques (ou échantillons) identiques donnent des empreintes identiques
* de taille **modeste** afin de pouvoir être stockées en grande quantité dans la base
* **simple** afin de pouvoir être calculé et comparé rapidement

Après de multiples essais, nous avons abouti à la méthode de construction des empreintes suivante. Les empreintes sont construites en conservant les maximas d’énergies détectés dans la bande son : la fonction librosa.onset.onset\_strength permet d’obtenir ces maximas dans un spectrogramme. Le spectrogramme en question est obtenu à partir d’une transformation CQT (constant-Q transform), alternative à la transformation de Fourrier, qui nous a donné de meilleurs résultats.

**3. Comparaison d’empreintes**

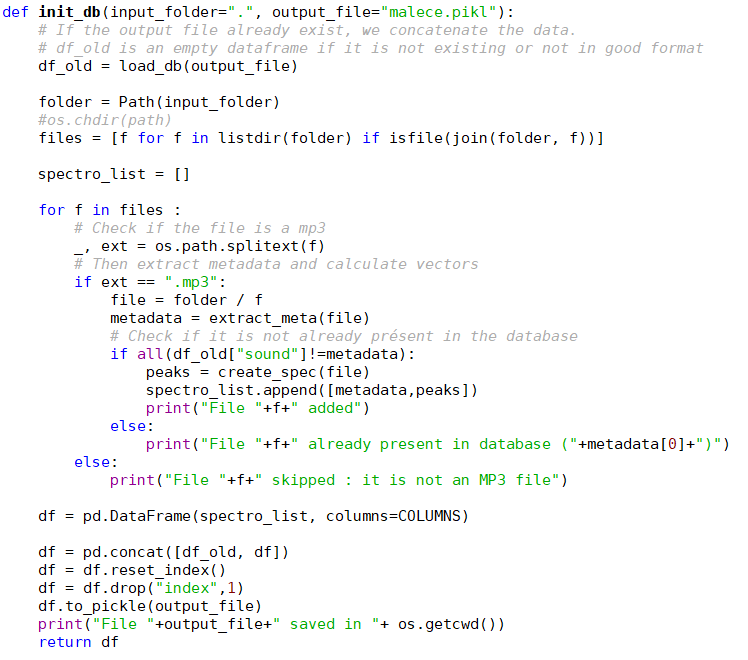
Lorsqu’un extrait est enregistré via le microphone, son empreinte est calculée de la même manière que pour la création de la base de données (voir section précédente) ; il devient alors possible de reconnaître un enregistrement via le microphone en comparant son empreinte à toutes les empreintes musicales de la base de données. Pour comparer l’empreinte « a » de l’enregistrement à une empreinte « b » de la base de données, on a eu recours à un indicateur simple : le coefficient de corrélation de Pearson. Or, « a » et « b » sont des vecteurs de taille différente, pour comparer « a » et « b » on extrait à chaque fois un sous vecteur « b ‘ » de « b » de même taille que « a » et on calcule le coefficient de corrélation entre « a » et « b’ ». Par la suite, on fait glisser le vecteur « b’ » tout au long de « b » en décalant à chaque fois d’un nombre d’éléments fixe (slide\_paddins) vers la droite jusqu’à ce qu’on arrive à la fin du vecteur « b ». Pour chaque vecteur « b’ » généré le coefficient de corrélation entre « a » et « b’ » est calculé. L’indice de similarité entre « a » et « b » est le coefficient de corrélation la plus grande entre le vecteur « a » et l’ensemble des vecteur « b’ ». On réitère le processus précédent en comparant l’empreinte « a » à toutes les empreintes de la base de données. L’empreinte de la base de données qui donnera le coefficient de corrélation la plus élevée est considérée comme le fichier audio le plus proche de l’enregistrement. Les métadonnées (Titre, auteur, album) correspondant à ce fichier peuvent être alors retournés.

La taille de la translation vers la droite, que nous avons nommé slide\_padding, est un hyper paramètre important de notre algorithme. En effet, si nous le fixons à 1 (valeure minimale pour que la translation soit fonctionnelle), cela signifie que nous allons tester tous les sous-vecteurs b’ possibles : pour chaque musique testée, nous allons donc obtenir le score de corrélation le plus correct, mais cela rend la recherche dans la base de données assez longue. Après des tests empirique, nous avons fixé cette valeurs à 4 : le temps de calcule de corrélation par musique a donc été divisé par 4, tandis que le taux de reconnaissance n’en soit affecté. Au delà de cette valeur, le score de corrélation est trop biaisé et empêche parfois la reconnaissance du bon titre.

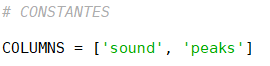
# Commentaires du code

1. Initialisation de la base de données : init\_db

La fonction init\_db prends deux paramètres en entrée : le chemin vers le répertoire contenant les musiques (répertoire courant par défaut) et le nom du fichier .pikl qui sera créé pour sauvegarder la base générée (si celui-ci existe déjà, les nouvelles pistes seront ajoutées à la base existante).



On notera que le nom des colonnes est utilisé afin de vérifier que le fichier chargé (s’il existe déjà) est bien un DataFrame ayant la même structure que ceux utilisés par le programme. Les colonnes en question sont définies comme suit :

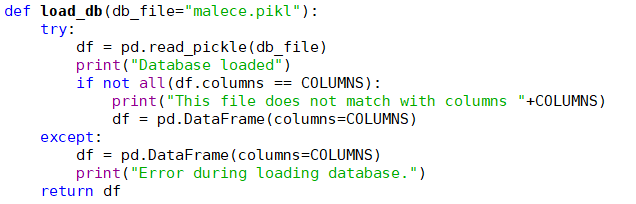


Les métadonnées sont extraites des fichiers MP3 grâce à la librairie *EasyID3* utilisée dans la fonction *extract\_metadata* suivante :



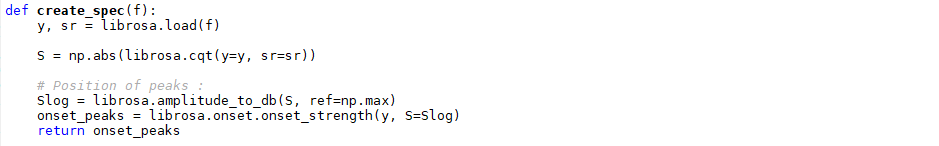
2. Chargement de la base de données : load\_db

Lorsque l’utilisateur utilise l’application, il n’a pas besoin de généré la base de données car celle-ci est déjà présente au format *peakle* (malece.pikl). La seule opération nécessaire est donc de charger cette base dans un DataFrame Pandas. C’est ce que permet la fonction load\_db, qui prends en paramètre le chemin vers le fichier peakle à charger :

Ici encore, on utilise le nom des colonnes pour vérifier qu’il s’agit bien d’une base initialement générée par Malece (init\_db). Dans le cas contraire, le DataFrame chargé sera vide.

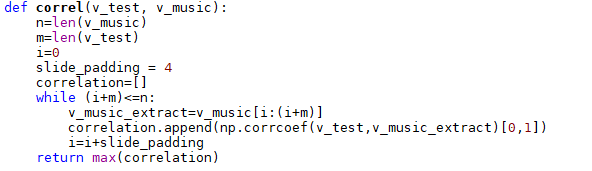
3. Création des empreintes : create\_spec

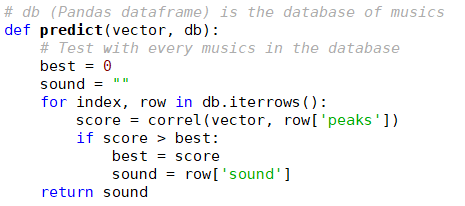
Cette fonction permet de calculer l’empreinte d’un fichier audio. Le fichier est passé en paramètre. Elle est utilisée à la fois pour initialiser la base de données et pour créer l’empreinte de l’échantillon enregistré à partir du microphone. En effet, dans ce dernier cas, l’échantillon est dans un premier temps enregistré localement sur le poste (dans un fichier .wav) puis soumis à cette fonction.



4. Comparaison des empreintes : predict & correl

La fonction correl permet de calculer la ressemblance entre deux empreintes. Comme expliqué dans la section précédente, celle-ci est calculée par corrélation avec un ensemble des « sous-empreintes » possibles (une sous-empreinte étant une portion de l’empreinte de même taille que l’échantillon testé) via une translation. On gardera, pour chaque musique comparée, la meilleure corrélation obtenue sur l’ensemble des sous-empreintes.

Enfin, la fonction predict permet de réaliser ce test sur l’ensemble des titres présent dans la base de données. On gardera donc le titre dont l’empreinte est la plus fortement corrélée avec l’échantillon enregistré par microphone.



5. Interface graphique

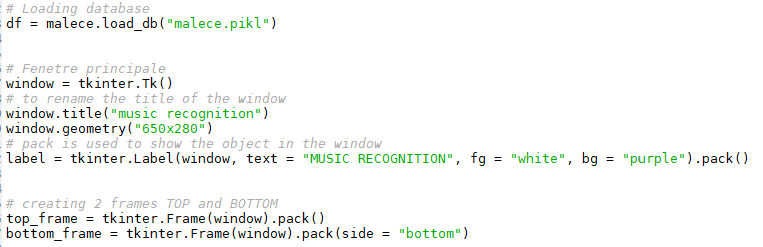
La mise en place d’une interface graphique simple nous a paru judicieuse notamment pour faciliter le contact entre l’utilisateur et l’algorithme. Nous avons donc utilisé le package ‘tkinter’ sous python qui permet de réaliser des interfaces basiques mais fonctionnelles.

L’idée est la suivante : l’utilisateur run le code, une fenêtre s’ouvre alors et lui demande combien de temps veut il enregistrer son morceau. Ce dernier rentre alors dans la case adéquat le nombre (entier) de secondes, il joue sur une source externe (smartphone, enceinte…) le son qu’il veut reconnaître et clique sur le bouton « start recognition ». A ce moment-là, le gif incrémenté dans la fenêtre devrait s’arrêter de tourner et ainsi permettre à l’utilisateur de vérifier que l’algorithme est bien en train de répondre à sa demande. Une fois la reconnaissance terminée, le gif s’anime de nouveau et une seconde fenêtre s’ouvre en proposant différentes informations sur la musique reconnu (artiste, titre, album, année).

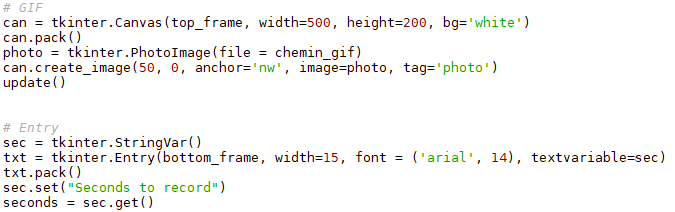
Dans le cas où l’utilisateur ne rentre pas un nombre entier ou si celui-ci est trop élevée, une fenêtre s’ouvre l’avertissant de ces informations.

Nous allons maintenant présenter des briques de codes constituant la majeure partie du script.

Lors du lancement du code, avant l’ouverture de la première fenêtre, la base de données contenant tous les spectrogrammes des musiques est chargée via la fonction ‘load\_db’ expliquée dans à la partie III.1) de ce papier. C’est dans un second temps que la fenêtre d’accueil est générée (‘tkinter.Tk()’) avec sa taille et son titre propre. Nous créons aussi deux frames que nous utiliserons comme architecture pour y mettre les différents éléments interactifs.



La création des éléments présent sur la fenêtre principale commence par celle du gif à l’aide de la commande ‘tkinter.Canvas’, puis celle de la zone où l’utilisateur va pouvoir inscrire le nombre de secondes qu’il veut enregistrer avec la commande ‘tkinter.Entry’.



La dernière partie de ce code correspond à la création du bouton (‘tkinter.Button’) qui va, lorsque l’utilisateur cliquera dessus, lancer une fonction (get\_seconds()) qui appelera l’algorithme de reconnaissance malace.py et enfin ouvrir une seconde page avec les information extraits du son reconnu.



**Pour aller plus loins**

**idées pouvant permettre d’améliorer le code :**

* **clustering des musiques, par type par exemple (rock, electro, ….). Cela permetrait de ne pas à avoir à parcourir toute la base mais uniquement ceux du même type**
* **Parallelisation des calculs de correlations**

**→ on a test, mais pas de résultats satisfaisants.**

biblio

<https://blogrecherche.wp.imt.fr/2015/09/07/une-application-de-reconnaissance-musicale-pour-les-morceaux-live/>

<https://interstices.info/de-fourier-a-la-reconnaissance-musicale/>

<http://myinspirationinformation.com/uncategorized/audio-signals-in-python/>

<https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/>

<https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf>

<https://pastel.archives-ouvertes.fr/tel-01307915/document>

<http://service.tsi.telecom-paristech.fr/cgi-bin/valipub_download.cgi?dId=185>

https://github.com/jiaaro/pydub#installation

https://github.com/dpwe/audfprint