Analyse de mélodies en Python 3 avec music21

Cette activité sert à découvrir la libraire Python music21 (`http://mit.edu/music21`) et demande quelques connaissances de base en Python. Si `music21` n'est pas installé, faire, sur le terminal, la commande `pip3 install music21`.

Une mélodie simple

1. Lance `python3`. Après avoir f `from music21 import *`, tu peux rentrer une petite mélodie, le début de *Dansons la capucine*: `capucine = converter.parse("tinyNotation: 4/4 c4 d8 d g g e e")`

Tu peux aussi copier depuis `www.algomus.fr/mediation` d'autres exemples.

2. Affiche le contenu avec `print(capucine)`. Tu peux aussi utiliser `capucine.show('text')`.

Selon l'installation de music21 faite sur ton ordinateur, les commandes `capucine.show()` et `capucine.show('midi')` peuvent aussi être disponibles.

3. Affiche tous les notes de `capucine` une par une, avec une boucle:

```
for n in capucine.flat.notes:
    print(n)
```

4. Que se passe-t-il si on ne met pas le `notes` ?

5. Que se passe-t-il si on ne met pas le `flat` ? Les objets en music21 sont en effet rangés de manière arborescente. Ici la partition contient des *mesures* et les mesures contiennent des notes.

Hauteurs de note et durées

Chaque note a une hauteur et une durée. Étant donné une note `n`, on peut afficher son *code MIDI* par `n.pitch.midi`. Le code 60 est le "Do médium", sous la portée en clé de Sol. La partition jointe te montre les codes de quelques notes dans quelques chansons.

1. Réalise une fonction `hauteurs(c)` qui affiche toutes les hauteurs des notes.

2. Réalise une fonction `plus_haute_note(c)` qui renvoie la note la plus haute d'une chanson `c`.

Teste tes fonctions sur `capucine`.

On peut renvoyer la durée d'une note par `n.duration.quarterLength`. Le 1.0 est une *noire*, le 0.5 une *croche* (deux fois plus courte que la noire) et le 2.0 une *blanche* (deux fois plus longue que la noire).

3. Réalise une fonction `plus_courte_note(c)` qui renvoie la note la plus courte d'une chanson `c`.

Transpositions et transformations

On peut aussi modifier les hauteurs et durées, comme dans `n.duration.quarterLength = 0.5` ou bien `n.pitch.midi = 62`.

1. Crée une fonction `transforme_longueurs(c, ratio)`, qui, étant donné une chanson `c` et un nombre `ratio`, multiplie par `ratio` toutes les longueurs. Teste ta fonction avec `transforme_longueurs(capucine, 2.0)` et `transforme_longueurs(capucine, 0.5)`.

2. Crée aussi une fonction `transpose(c, transposition)` qui modifie toutes les hauteurs par l'entier `transposition` donné en paramètre. Teste ta fonction avec `transpose(capucine, 7)` et `transpose(capucine, 12)`.

Intervalles

Un intervalle est la différence entre deux hauteurs successives. Les intervalles jouent un grand rôle en musique: plutôt que la hauteur absolue des notes, on se souvient généralement du *contour* d'une mélodie, c'est-à-dire de l'ensemble de ses intervalles.

1. Réalise une fonction `cherche_intervalle(c, inter)` qui, étant donné un intervalle (comme `-4`), donne une note dont la suivante est à une hauteur de "4 demi-tons en dessous" (intervalle appelé "une tierce majeure"), comme dans `64 60`.
2. Réalise aussi une fonction `calcule_intervalles(c)` qui renvoie la liste de tous les intervalles d'une chanson.
3. Réalise enfin une fonction `cherche_suite_intervalles(c, inters)` qui, étant donné une liste d'intervalles (comme `[4, -2]`), donne une note suivie de ces intervalles (comme `60` si on la trouve dans `60 64 62`.)

Comparaison de mélodies

Tu peux désormais chercher à faire des analyses plus complexes, comme par exemple comparer deux mélodies. Peux-tu faire une fonction qui cherche des fragments similaires au milieu de plusieurs mélodies ?
