# Génération de mélodie de façon aléatoire

In [3]:

```
import music21 as m21
import random
```

## Génération aléatoire basique

**Génération de mélodies aléatoires en do majeur et en 4|4 avec seulement 7 notes et des silences.**

In [ ]:

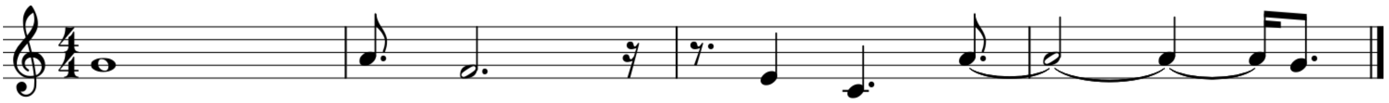
```
NOTE_DURATIONS = [
    0.25,
    0.5,
    0.75,
    1,
    1.5,
    2,
    3,
    4
]

NOTES = ['C','D','E','F','G','A','B']
```

In [16]:

```
from music21 import note, stream

# Générer une mélodie d'une durée spécifiée
def generate_random_melody(total_duration=16, silence_probability=0.2):
    melody = stream.Stream()
    current_duration = 0  # Temps total actuel

    while current_duration < total_duration:
        duration = random.choice(NOTE_DURATIONS)  # Choix d'une durée aléatoire

        # Empêcher de dépasser la durée totale
        if current_duration + duration > total_duration:
            duration = total_duration - current_duration  # Ajuste pour finir pile

        if random.random() < silence_probability:  # Probabilité d'ajouter un silence
            element = note.Rest(quarterLength=duration)  # Créer un silence
        else:
            pitch = random.choice(NOTES)  # Note aléatoire
            element = note.Note(pitch, quarterLength=duration)  # Créer une note

        melody.append(element)  # Ajouter à la mélodie
        current_duration += duration  # Mettre à jour la durée actuelle

    return melody
```

In [17]:

```
melody = generate_random_melody()
melody.show()
```



In [18]:

```
melody.show('midi')
```

## Génération aléatoire avec un choix de notes

**Génération de mélodie aléatoire avec les # en plus.**

In [19]:

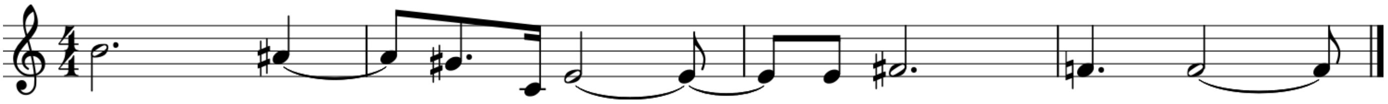
```
NOTES_ALL = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
```

In [30]:

```
def random_melody_with_notes(total_duration=16, silence_probability=0.1, notes=NOTES_ALL)
:
    melody = stream.Stream()
    current_duration = 0   # Temps total actuel

    while current_duration < total_duration:
        duration = random.choice(NOTE_DURATIONS)   # Choix d'une durée aléatoire

        # Empêcher de dépasser la durée totale
        if current_duration + duration > total_duration:
            duration = total_duration - current_duration  # Ajuste pour finir pile

        if random.random() < silence_probability:  # Probabilité d'ajouter un silence
            element = note.Rest(quarterLength=duration)  # Créer un silence
        else:
            pitch = random.choice(notes)  # Note aléatoire
            element = note.Note(pitch, quarterLength=duration)  # Créer une note

        melody.append(element)  # Ajouter à la mélodie
        current_duration += duration  # Mettre à jour la durée actuelle

    return melody
```

In [31]:

```
melody = random_melody_with_notes(notes=NOTES_ALL)
melody.show()
```



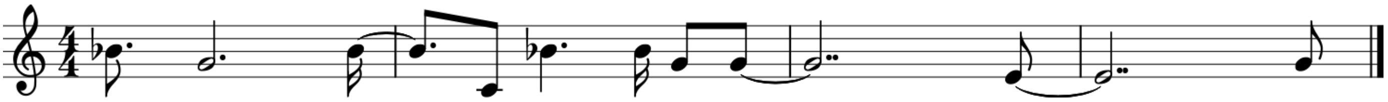
In [32]:

```
melody.show('midi')
```

**Exemple de génération de mélodies avec des accords**

In [37]:

```
# Choix de l'accord
CHORD = ['C','E','G','Bb']

# Génération
melody = random_melody_with_notes(silence_probability=0.05,notes=CHORD)
melody.show()
```



In [38]:

```
melody.show('midi')
```

In [ ]:

# Génération de mélodies avec un RNN-LSTM - Partie 1 : preprocessing des données

#### ressource: tuto youtube *Melody generation with RNN-LSTM* de *Valerio Velardo*

In [102]:

```
import os
import music21 as m21
import json
```

## Préparation des données

### Charger données

In [69]:

```
def load_songs(data_path, max_songs_nb):
    songs = []
    for path, subdirs, files in os.walk(data_path):
        for file in files:
            if file[-3:] == "krn":
                #print(os.path.join(path, file))
                song = m21.converter.parse(os.path.join(path, file))
                songs.append(song)
                max_songs_nb -= 1
                if max_songs_nb == 0 : return songs
    return songs
```

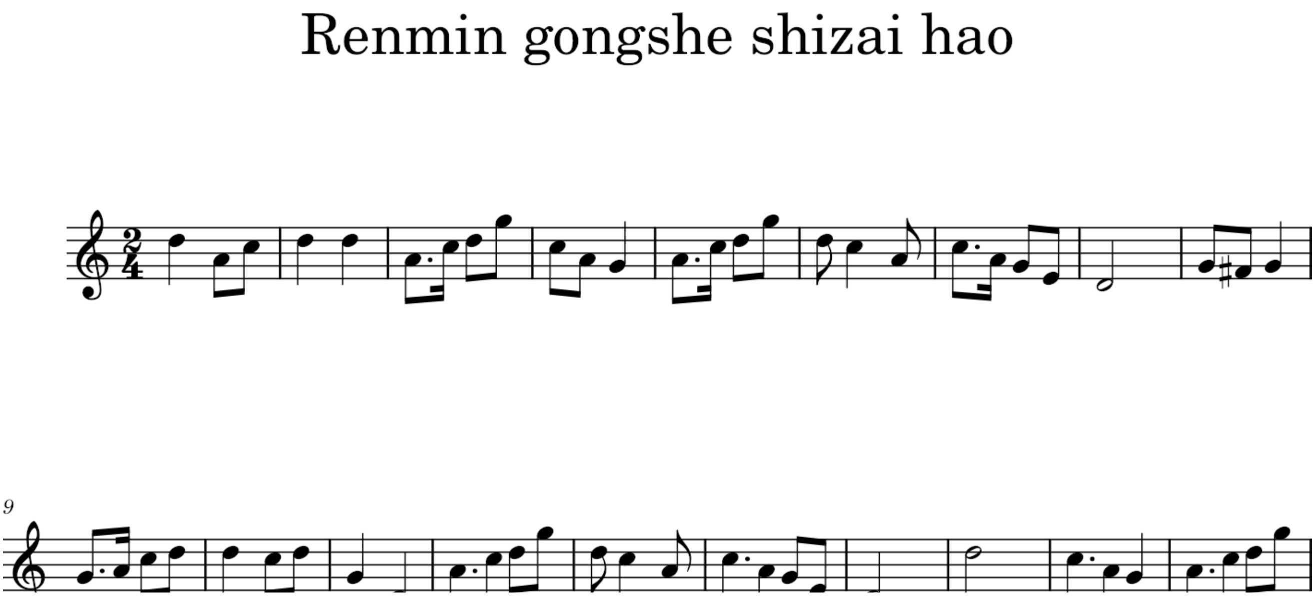
In [70]:

```
DATASET_PATH = "data/han"

print("loading data...")
songs = load_songs(DATASET_PATH, 100)
print("songs loaded")
```

```
loading data...
songs loaded
```

In [71]:

```
songs[0].show('midi')
songs[0].show()
```



**Enlever les rythmes bizarres (garder les noires, croches, ...)**

In [72]:

```
# durations are expressed in quarter length
ACCEPTABLE_DURATIONS = [
    0.25, # 16th note
    0.5, # 8th note
    0.75,
    1.0, # quarter note
    1.5,
    2, # half note
    3,
    4 # whole note
]

def has_acceptable_durations(song, acceptable_durations):
    for note in song.flatten().notesAndRests:
        if note.duration.quarterLength not in acceptable_durations:
            return False
    return True
```

In [73]:

```
print("avant filtrage :", len(songs))
for song in songs:
    if not has_acceptable_durations(song, ACCEPTABLE_DURATIONS):
        # song.show()
        # song.show("midi")
        songs.remove(song)
print("après filtrage :", len(songs))
```

```
avant filtrage : 100
après filtrage : 94
```

**Transposer en do majeur/la mineur (= ne rien avoir à la clé, tout dans la même tonalité)**

In [74]:

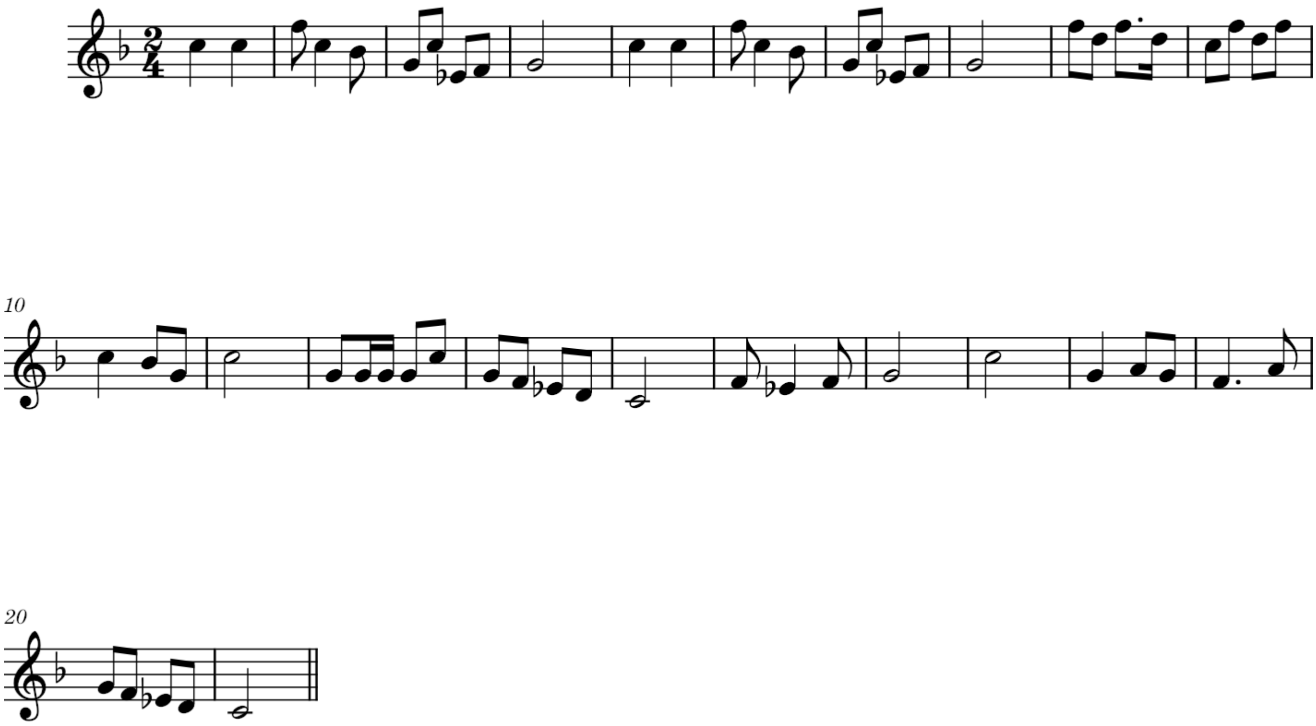
```
def transpose(song, print_enabled=False):
    # transpose song in Cmaj/Amin

    # get key signature
    parts = song.getElementsByClass(m21.stream.Part)
    measures_part0 = parts[0].getElementsByClass(m21.stream.Measure)
    key = measures_part0[0][4]
    if print_enabled : print("old key : ", key)

    # estimate key if not indicated
    if not isinstance(key, m21.key.Key):
        key = song.analyse("key")

    # get interval for transposition
    if key.mode == "major":
        interval = m21.interval.Interval(key.tonic, m21.pitch.Pitch("C"))
    elif key.mode == "minor":
        interval = m21.interval.Interval(key.tonic, m21.pitch.Pitch("A"))
    transposed_song = song.transpose(interval)

    return transposed_song
```

In [87]:

```
# test
song = songs[1]
print("Before transposition into Cmaj")
song.show('midi')
song.show()

song = transpose(song, True)
print("After transposition into Cmaj")
song.show('midi')
song.show()
```

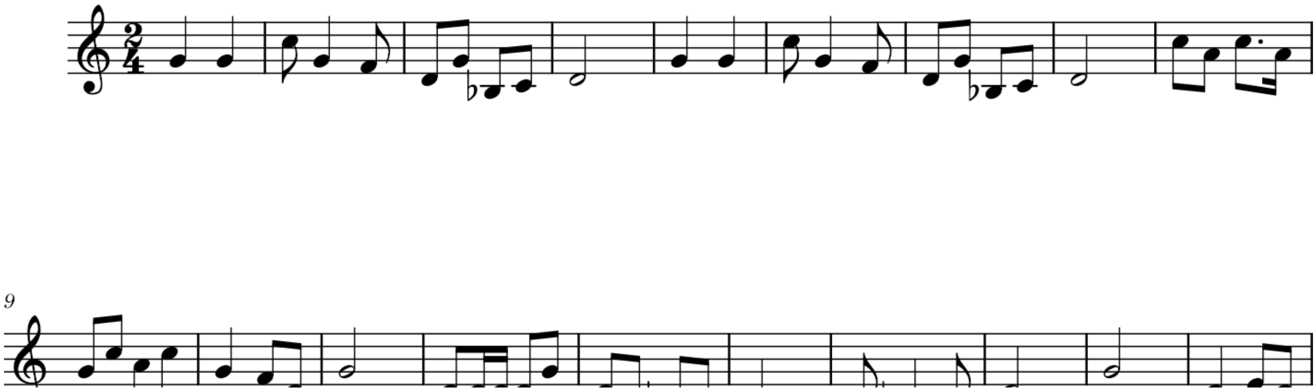
```
Before transposition into Cmaj
```

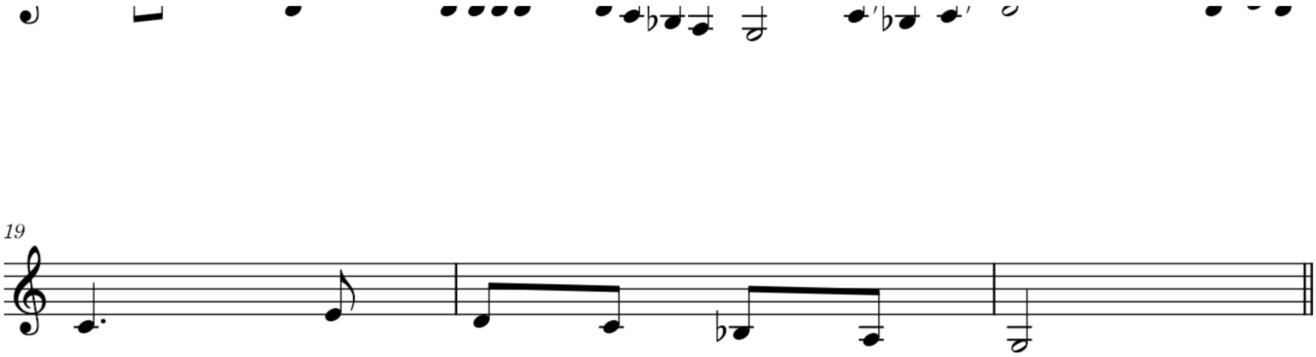
# Zanmen de ling xiu Mao Zedong



```
old key :  F major
After transposition into Cmaj
```

# Zanmen de ling xiu Mao Zedong





In [78]:

```
transposed_songs = []
for song in songs:
    transposed_songs.append(transpose(song))
```

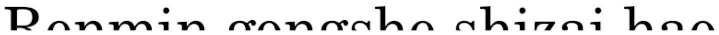
**Encoder les musiques dans un format qui ira dans un fichier texte**

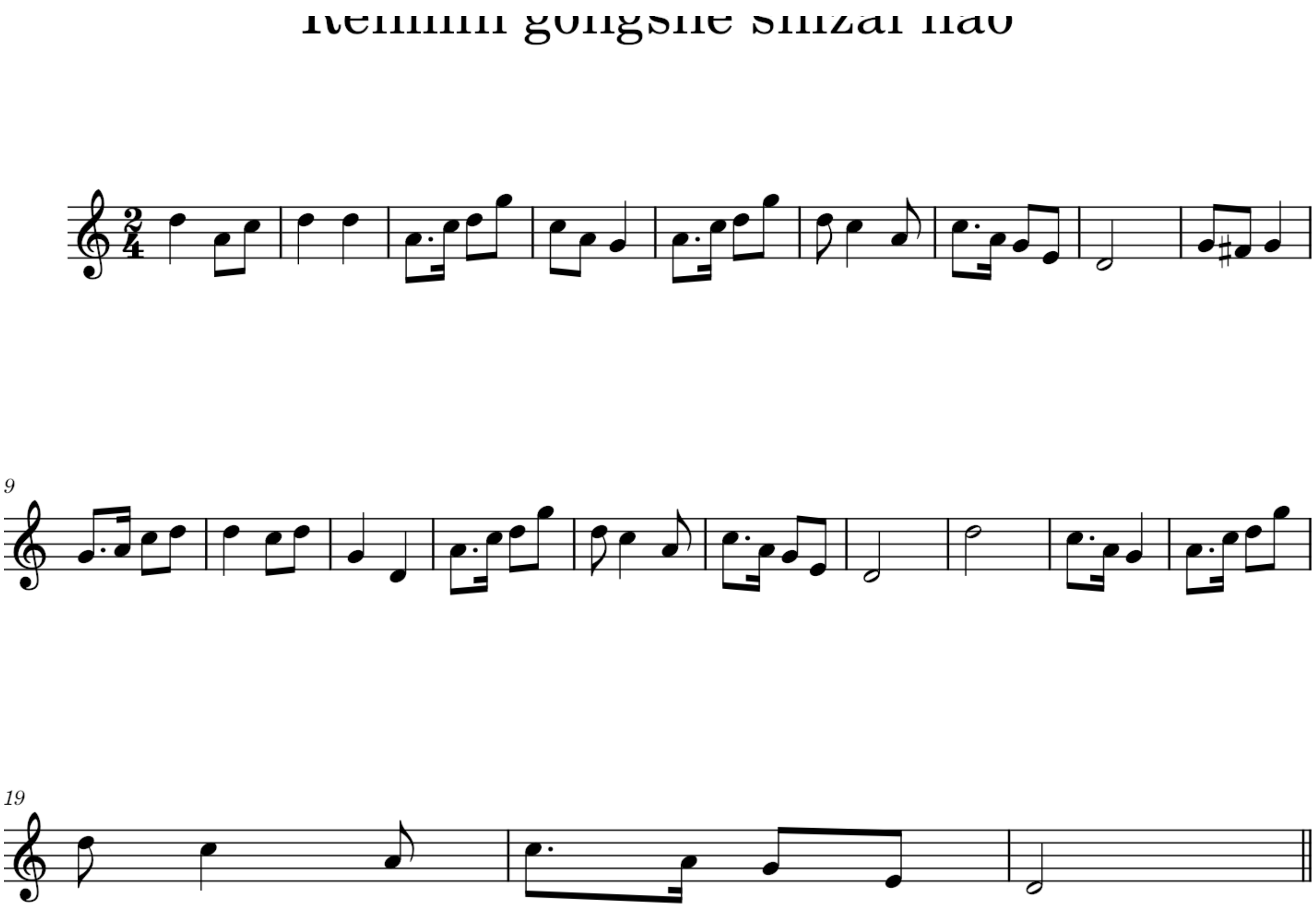
In [82]:

```
def encode_song(song, time_step=0.25):
    """Converts a score into a time-series-like music representation. Each item in the encoded list represents 'min_duration'
    quarter lengths. The symbols used at each step are: integers for MIDI notes, 'r' for representing a rest, and '_'
    for representing notes/rests that are carried over into a new time step. Here's a sample encoding:

        ["r", "_", "60", "_", "_", "_", "72" "_"]

    :param song (m21 stream): Piece to encode
    :param time_step (float): Duration of each time step in quarter length
    :return:
    """

    encoded_song = []

    for event in song.flatten().notesAndRests:

        # handle notes
        if isinstance(event, m21.note.Note):
            symbol = event.pitch.midi # 60
        # handle rests
        elif isinstance(event, m21.note.Rest):
            symbol = "r"

        # convert the note/rest into time series notation
        steps = int(event.duration.quarterLength / time_step)
        for step in range(steps):

            # if it's the first time we see a note/rest, let's encode it. Otherwise, it means we're carrying the same
            # symbol in a new time step
            if step == 0:
                encoded_song.append(symbol)
            else:
                encoded_song.append("_")

    # cast encoded song to str
    encoded_song = " ".join(map(str, encoded_song))

    return encoded_song
```

In [85]:

```
print(transposed_songs[0].show())
print(encode_song(transposed_songs[0]))
```





```
None
74 _ _ _ 69 _ 72 _ 74 _ _ _ 74 _ _ _ 69 _ _ 72 74 _ 79 _ 72 _ 69 _ 67 _ _ _ 69 _ _ 72 74
_ 79 _ 74 _ 72 _ _ _ 69 _ 72 _ _ 69 67 _ 64 _ 62 _ _ _ _ _ _ _ 67 _ 66 _ 67 _ _ _ 67 _ _
69 72 _ 74 _ 74 _ _ _ 72 _ 74 _ 67 _ _ _ 62 _ _ _ 69 _ _ 72 74 _ 79 _ 74 _ 72 _ _ _ 69 _
72 _ _ 69 67 _ 64 _ 62 _ _ _ _ _ _ _ 74 _ _ _ _ _ _ _ 72 _ _ 69 67 _ _ _ 69 _ _ 72 74 _
79 _ 74 _ 72 _ _ _ 69 _ 72 _ _ 69 67 _ 64 _ 62 _ _ _ _ _ _ _
```

In [86]:

```
encoded_songs = []
for song in transposed_songs:
    encoded_songs.append(encode_song(song))
```

### sauvegarde dans un fichier texte

In [89]:

```
SAVE_DIR = "data/han/encoded_songs"
for i, encoded_song in enumerate(encoded_songs):
    save_path = os.path.join(SAVE_DIR, str(i))
    with open(save_path, "w") as fp:
        fp.write(encoded_song)
```

In [97]:

```
#test
with open("data/han/encoded_songs/0", "r") as fp:
    song = fp.read()
    print(song)
```

```
74 _ _ _ 69 _ 72 _ 74 _ _ _ 74 _ _ _ 69 _ _ 72 74 _ 79 _ 72 _ 69 _ 67 _ _ _ 69 _ _ 72 74
_ 79 _ 74 _ 72 _ _ _ 69 _ 72 _ _ 69 67 _ 64 _ 62 _ _ _ _ _ _ _ 67 _ 66 _ 67 _ _ _ 67 _ _
69 72 _ 74 _ 74 _ _ _ 72 _ 74 _ 67 _ _ _ 62 _ _ _ 69 _ _ 72 74 _ 79 _ 74 _ 72 _ _ _ 69 _
72 _ _ 69 67 _ 64 _ 62 _ _ _ _ _ _ _ 74 _ _ _ _ _ _ _ 72 _ _ 69 67 _ _ _ 69 _ _ 72 74 _
79 _ 74 _ 72 _ _ _ 69 _ 72 _ _ 69 67 _ 64 _ 62 _ _ _ _ _ _ _
```

### tout mettre dans un fichier

In [98]:

```
def load(file_path):
```

```python
def load(file_path):
    with open(file_path, "r") as fp:
        song = fp.read()
    return song
```

In [99]:

```python
def create_single_file_dataset(dataset_path, file_dataset_path, sequence_length):
    """Generates a file collating all the encoded songs and adding new piece delimiters.

    :param dataset_path (str): Path to folder containing the encoded songs
    :param file_dataset_path (str): Path to file for saving songs in single file
    :param sequence_length (int): # of time steps to be considered for training
    :return songs (str): String containing all songs in dataset + delimiters
    """

    new_song_delimiter = "/ " * sequence_length
    songs = ""

    # load encoded songs and add delimiters
    for path, _, files in os.walk(dataset_path):
        for file in files:
            file_path = os.path.join(path, file)
            song = load(file_path)
            songs = songs + song + " " + new_song_delimiter

    # remove empty space from last character of string
    songs = songs[:-1]

    # save string that contains all the dataset
    with open(file_dataset_path, "w") as fp:
        fp.write(songs)

    return songs
```

In [100]:

```python
create_single_file_dataset(dataset_path=SAVE_DIR, file_dataset_path="data/han/file_datase
t", sequence_length=64)
```

Out[100]:

```
'74 _ _ _ 69 _ 72 _ 74 _ _ _ 74 _ _ _ 69 _ _ 72 74 _ 79 _ 72 _ 69 _ 67 _ _ _ 69 _ _ 72 74
_ 79 _ 74 _ 72 _ _ _ 69 _ 72 _ _ 69 67 _ 64 _ 62 _ _ _ _ _ _ _ 67 _ 66 _ 67 _ _ _ 67 _ _
69 72 _ 74 _ 74 _ _ _ 72 _ 74 _ 67 _ _ _ 62 _ _ _ 69 _ _ 72 74 _ 79 _ 74 _ 72 _ _ _ 69 _
72 _ _ 69 67 _ 64 _ 62 _ _ _ _ _ _ _ 74 _ _ _ _ _ _ _ 72 _ _ 69 67 _ _ _ 69 _ _ 72 74 _
79 _ 74 _ 72 _ _ _ 69 _ 72 _ _ 69 67 _ 64 _ 62 _ _ _ _ _ _ _ / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / 67 _ _ _ 67 _ _ _ 72 _ 67 _ _ _ 65 _ 62 _ 67 _ 58 _ 60 _ 62 _ _ _ _ _ _ _ 67
_ _ _ 67 _ _ _ 72 _ 67 _ _ _ 65 _ 62 _ 67 _ 58 _ 60 _ 62 _ _ _ _ _ _ _ 72 _ 69 _ 72 _ _
69 67 _ 72 _ 69 _ 72 _ 67 _ _ _ 65 _ 62 _ 67 _ _ _ _ _ _ _ 62 _ 62 62 62 _ 67 _ 62 _ 60 _
58 _ 57 _ 55 _ _ _ _ _ _ _ 60 _ 58 _ _ _ 60 _ 62 _ _ _ _ _ _ _ 67 _ _ _ _ _ _ _ 62 _ _ _
64 _ 62 _ 60 _ _ _ _ _ 64 _ 62 _ 60 _ 58 _ 57 _ 55 _ _ _ _ _ _ _ / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / / 67 _ 67 _ 62 _ 62 64 67 _ 67 _ 62 _ 62 64 65 _ 65 67 72 _ 69 _ 67 _ _ _ _
_ _ _ 65 _ 65 67 72 _ 69 _ 67 _ _ _ _ _ _ _ 62 _ 67 _ 64 _ 62 _ 60 _ 60 _ 57 _ 55 _ 60 _
60 _ 57 _ 55 _ 60 _ _ 62 67 _ 64 _ 62 _ _ _ _ _ 64 _ 60 _ 62 64 60 _ 57 _ 55 _ _ _ _ _ _
_ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / / / 67 _ 67 _ 69 _ 72 64 62 60 62 _ _ _ 72 69 67 _
67 64 62 64 62 55 60 _ _ _ _ _ _ _ 62 _ 67 _ 65 _ 64 62 60 _ 59 57 55 _ 67 _ 60 _ 59 57 5
5 57 55 50 55 _ _ _ _ _ _ _ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / 74 _ 72 69 74 _ 79
76 74 _ 76 74 72 _ 69 _ 74 _ 76 79 74 72 69 _ 69 67 65 62 67 _ _ _ 69 74 67 69 74 _ _ 72 6
9 72 69 67 65 _ _ 67 69 _ 69 72 69 67 65 64 62 _ _ 67 62 _ _ _ / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / 64 _ 67 _ 69 _ _ _ 72 _ 72 _ _ _ 69 _ 67 _ 64 _ 67 _ 72 _ 67 _ _ _ _ _ _ _
64 _ 67 _ 69 _ _ 69 72 _ 69 _ 67 _ 64 _ 62 _ 60 _ 62 _ 64 _ 62 _ _ _ _ _ _ _ 67 _ _ _ 58
_ _ _ 67 _ _ _ 58 _ _ _ 67 _ 64 _ 67 _ 69 _ 60 _ _ _ _ _ _ _ 57 _ 60 _ 62 _ _ 62 64 _ 62
_ 60 _ 57 _ 55 _ 52 _ 55 _ 57 _ 55 _ _ _ _ _ _ _ 62 _ _ 60 62 _ 64 _ 62 _ _ _ _ _ 64 _ 67
_ _ 64 67 _ 69 _ 60 _ _ _ _ _ _ _ 57 _ 60 _ 62 _ _ 62 64 _ 62 _ 60 _ 57 _ 55 _ 52 _ 55 _
57 _ 55 _ _ _ _ _ _ _ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / 55 _ 60 _ 60 _ 60 60 62 _
67           67 72 67       62   60           r   55   60   60   60   62   67   62 60 62
```

```
_ 60 _ 60 _ 57 _ _ _ 60 _ 62 _ 60 _ 62 _ 64 _ 57 _ _ _ 62 _ 60 _ 60 _ 57 _ r _ _ _ / / /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / 69 _ 72 _ 69 67 64 67 69 _ 69 _ _ _ 67 _ 72 _ 69 67 64
_ 67 _ 69 _ 67 _ 64 _ _ _ 69 _ 64 67 69 _ 67 _ 64 _ 67 _ _ _ 64 _ 64 _ _ 67 60 _ 62 _ 67
_ 64 _ _ _ _ _ 62 64 62 60 57 _ 60 _ 64 _ 62 _ r _ 69 69 69 72 69 67 67 _ 64 _ 67 _ 64 62
60 _ 62 _ r _ 64 _ 60 _ 62 _ 67 _ 64 _ 67 _ _ _ 62 _ 64 67 62 _ 60 _ 57 _ _ _ _ _ _ _ /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / 64 _ 60 _ 62 _ _ _ 62 _ 60 57 57 _ _ _ 62 _ 60 62
60 _ 60 _ 60 _ 57 55 55 _ _ _ 60 57 _ 57 62 _ 62 _ 60 57 _ 60 64 62 _ _ 62 60 _ _ 57 _ _
_ 62 60 _ 62 62 60 _ 57 64 55 _ 57 55 57 60 _ 64 _ 55 57 60 _ _ 64 60 _ 57 _ 55 _ _ _ 64
_ 60 _ 62 _ _ _ 62 _ 60 57 57 _ _ _ 62 60 _ 62 62 _ 60 _ 60 _ 57 55 55 _ _ _ 62 _ _ 62 64
62 _ _ 60 60 _ 60 64 62 _ _ 62 60 60 57 57 _ _ _ 55 _ _ 57 62 60 57 _ 62 64 60 62 64 62 _
_ 64 _ 55 57 60 _ _ 64 60 _ 57 _ 55 _ _ _ 64 _ 67 64 62 _ _ _ 62 _ 60 57 57 _ _ _ 62 _ 60
62 62 _ 60 _ 60 _ 57 55 55 _ _ _ 62 _ _ 60 64 _ 62 _ 62 62 62 60 64 62 _ _ 62 60 60 57 57
_ _ _ 55 _ _ 57 62 60 57 _ 62 62 60 57 55 57 60 _ 64 _ 55 57 60 _ _ 64 60 _ 57 _ 55 _ _ _
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / / 67 _ 67 _ 69 _ 72 _ 74 _ _ _ _ _ 72 _ 69 _ 74 _
72 _ _ _ 69 _ _ _ 67 _ _ _ 72 _ 69 67 72 _ 72 _ 69 72 74 _ 72 _ _ _ 72 _ 69 67 65 67 69 _
67 _ _ _ _ _ _ _ 67 _ 67 _ 67 _ 69 _ 74 _ _ _ _ _ 72 _ 69 _ 74 _ 72 _ _ _ 69 _ _ _ 67 _
_ _ 69 72 69 67 72 _ 72 _ 69 72 74 _ 72 _ _ _ 67 _ 69 72 69 _ _ _ 67 _ _ _ _ _ _ _ / / /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / /'
```

#### mapping dans un fichier json

In [101]:

```
def create_mapping(songs, mapping_path):
    """Creates a json file that maps the symbols in the song dataset onto integers

    :param songs (str): String with all songs
    :param mapping_path (str): Path where to save mapping
    :return:
    """
    mappings = {}

    # identify the vocabulary
    songs = songs.split()
    vocabulary = list(set(songs))

    # create mappings
    for i, symbol in enumerate(vocabulary):
        mappings[symbol] = i

    # save voabulary to a json file
    with open(mapping_path, "w") as fp:
        json.dump(mappings, fp, indent=4)
```

In [104]:

```
songs=load("data/han/file_dataset")
create_mapping(songs, "data/han/mapping.json")
```

# Génération de mélodies avec un RNN-LSTM - Partie 2 : training

### ressource: tuto youtube *Melody generation with RNN-LSTM* de *Valerio Velardo*

In [14]:

```
import os
import music21 as m21
import json
import numpy as np
import tensorflow.keras as keras
```

#### charger fichier de données et fichier de mapping

In [15]:

```
def load(file_path):
    with open(file_path, "r") as fp:
        song = fp.read()
    return song

songs = load("data/han/file_dataset")

def load_json(file_path):
    with open(file_path, "r") as fp:
        mappings = json.load(fp)
    return mappings

mappings = load_json("data/han/mapping.json")
```

#### générer séquences d'entrainements

In [16]:

```
def convert_songs_to_int(songs, mappings):
    int_songs = []

    # transform songs string to list
    songs = songs.split()

    # map songs to int
    for symbol in songs:
        int_songs.append(mappings[symbol])

    return int_songs
```

In [17]:

```
def generate_training_sequences(sequence_length):
    """Create input and output data samples for training. Each sample is a sequence.

    :param sequence_length (int): Length of each sequence. With a quantisation at 16th notes, 64 notes equates to 4 bars

    :return inputs (ndarray): Training inputs
    :return targets (ndarray): Training targets
    """

    # map songs to int
    int_songs = convert_songs_to_int(songs, mappings)

    inputs = []
    targets = []
    # generate the training sequences
    num_sequences = len(int_songs) - sequence_length
    for i in range(num_sequences):
        inputs.append(int_songs[i:i+sequence_length])
        targets.append(int_songs[i+sequence_length])

    # one-hot encode the sequences
    vocabulary_size = len(set(int_songs))
    # inputs size: (# of sequences, sequence length, vocabulary size)
    inputs = keras.utils.to_categorical(inputs, num_classes=vocabulary_size)
    targets = np.array(targets)

    return inputs, targets
```

In [19]:

```
inputs, targets = generate_training_sequences(64)
```