

Instituto Tecnológico y de Estudios Superiores de Monterrey
Escuela de Ingeniería y Ciencias
Agosto - Diciembre 2022
Campus Monterrey



Implementación de Métodos Computacionales

Avance de Situación Problema 1

Implementación básica de un DSL para ejecutar música en Python



Profesor:

Roman Martinez Martinez

Grupo: 850

Integrantes:

Leonardo González Guerra

| A01721434

8 de septiembre de 2022

El mundo profesional de los ITC está lleno de retos interesantísimos y uno relativamente común es el de aprender nuevos lenguajes de programación o incluso diseñar nuevos lenguajes. En algunas ocasiones, hay necesidades tan específicas, que demandan un lenguaje de dominio específico o DSL (Domain Specific Language). Un DSL es un lenguaje de programación con un nivel superior de abstracción optimizado para una clase específica de problemas. Un DSL usa los conceptos y reglas de su campo o dominio y normalmente es menos complejo que un lenguaje de propósito general como los que típicamente utilizamos.

Aunque en la actualidad existen muchas tecnologías que permiten hacer música digital, es interesante conocer un poco de cómo el lenguaje de la música ha sido representado para su digitalización. Para esta problemática usaremos el lenguaje MUTRAN. El programa además de analizar el uso correcto del léxico de lenguaje musical, ejecuta el código de entrada con los sonidos correspondientes a las notas musicales identificadas. El análisis del léxico fue implementado en Python.

Empezaremos por definir las librerías que ocuparemos de Python para que nos ayuden a generar los sonidos que queremos. Usaremos la librería `re` para leer las expresiones regulares, la librería `numpy` para que nos ayude en las fórmulas matemáticas y la librería `sound device` que nos ayudará a generar los sonidos.

```
import re
import numpy as np
import sounddevice as sd
```

Con estas librerías podremos pasar a nuestro segundo paso el cual será definir nuestra fórmula para obtener la frecuencia y onda para generar el sonido. Dándole un modelo matemático de una onda sonora va poder reproducirla y transformarla en una onda acústica. La fórmula de la frecuencia es la cantidad de veces que se produce un suceso en determinada cantidad de tiempo. Como hablamos de ondas, será la cantidad de oscilaciones que van a producirse en un segundo que se mide en hertz y mientras más oscilaciones hay más aguda será la nota y al contrario mientras menos oscilaciones tenga por segundo nuestra nota será más grave.

$$440 * ((\sqrt[12]{2})^{(o-3)*12+n-10})$$

```
def get_freq(note, octave):
    expo = (octave - 4) * 12 + (note - 10)
    return 440 * ((2 ** (1/12)) ** expo)
```

Después queda modelar nuestra onda la cual es la amplitud de la frecuencia. Pero necesitamos pasarle la amplitud a distintos instantes para que vaya formando la onda. Si partimos un segundo en dos, y calcularemos la altura de la onda cada medio segundo y unimos los puntos, obtendremos una onda deforme, lo que menos parecería es una onda. Pero podemos seguir dividiendo un segundo en 4, 8, 50, 100 y así hasta obtener mejores resultados.

Por eso podemos usar como framerate o frecuencia de muestreo 44100 hz, o 48000 hz si ustedes lo prefieren, que son convenciones que se utilizan en audio y son inputs que admite sounddevice

```
framerate = 44100
```

Ahora queda calcular las muestras para el total de la duración de la nota. Suponiendo que dicha duración es un segundo vamos a partir ese segundo en 44100 (hz) partes iguales y como nuestro segundo se expresa en milisegundo. 1 segundo = 1000 milisegundos. Esto nos devolverá un arreglo con 44100 valores por segundo el cual debemos después pasar la fórmula de la onda.

$$y(t) = A \sin(2\pi ft + \varphi)$$

```
time = 1000
```

```
def get_wave(note, octave, duration):
    framerate = 44100
    freq = get_freq(note, octave)
```

```
t = np.linspace(0, duration/time, int( framerate*duration/time))
wave = np.sin(2 * np.pi * freq * t)
```

Para reproducir nuestra onda solo queda usar nuestra librería sounddevice para obtener el sonido acústico.

- La onda que ya modelamos
- La frecuencia de muestreo

```
sd.play(data, frameRate)
sd.wait()
```

El lenguaje MUTRAN utiliza una sintaxis muy sencilla y fácil de leer aunque no tengas mucha experiencia leyendo música utilizas nada mas lo básico para poder generar un arreglo de notas musicales la cual será después interpretado por nuestro código, lexemas, fórmulas y funciones.

```
#Ejemplo MUTRAN que leera la escala musical
```

```
#DO    RE    MI FA  SOL  LA    SI
#C C# D D# E F F# G G# A A# B
```

```
instrumento C3h C#3h D3h D#3h E3h F3h F#3h G3h G#3h A3h A#3h B3h
```

En este ejemplo tenemos un archivo .txt el cual será leído por nuestro código en python que interpretará de la siguiente forma el lenguaje MUTRAN:

- Los comentarios empiezan con # y serán ignorados ya que son para que un usuario sepa interpretar y no una máquina.
- El “instrumento” colocado antes de la escala es el sonido que escogerá para interpretar.
- Empezamos con la escala musical y empezamos con el Do el cual es también llamado C. Tenemos nuestra escala musical la cual es C, D, E, F, G, A, B pero en esta ocasión usamos sostenidos por eso es C, C#, D, D#, E, F, F#, G, G#, A, A#, B.
- El 3 que le sigue es en la octava que estará del piano la cual es la posición que estuviéramos en un piano. Mientras más grave queramos el sonido tendremos que

bajar el número y si lo queremos más agudo obviamente tendremos que subir el número.

- El “w”, “h”, “q”, “e”, “s”, “t” y “f” al final de cada nota es el tiempo que durará la nota. Ya sea una blanca, negra, corchea etc. Esto nos regresará un valor del 16.6 al 1000 que es el máximo tiempo que pusimos como variable el cual es un segundo.

Con esto ya aprendido hay que ver como nuestro código leerá este léxico. Deberemos hacer funciones dentro de nuestro código para que puedan leer nuestros archivos .txt y puedan reconocer según la nota que muestra el archivo el valor que regresaran para después usar dentro de nuestras fórmulas seguidas por el modelo matemático de la frecuencia en una onda. Para la función que leerá nuestra nota y nos regresara el sonido de la nota en cuestión será la siguiente.

```
def get_letter(note):  
    if note == "A":  
        return 10  
    elif note == "B":  
        return 12  
    elif note == "C":  
        return 1  
    elif note == "D":  
        return 3  
    elif note == "E":  
        return 5  
    elif note == "F":  
        return 6  
    elif note == "G":  
        return 8  
    return 0
```

Después viene reconocer en qué octava tenemos posicionadas nuestras notas musicales. Tendremos un rango de 0 a 8 siendo 0 el más grave y 8 el más agudo.

```
def get_octave(octave):  
    if octave == "0":  
        return 0  
    elif octave == "1":  
        return 1
```

```

elif octave == "2":
    return 2
elif octave == "3":
    return 3
elif octave == "4":
    return 4
elif octave == "5":
    return 5
elif octave == "6":
    return 6
elif octave == "7":
    return 7
elif octave == "8":
    return 8
return 0

```

Para nuestros sostenido o bemoles definiremos una pequeña función que llamaremos `accidental` la cual nos regresara el valor indicado según la nota a la cual está concatenada.

```

def get_accidental(accidental):
    if accidental == "#":
        return 1
    elif accidental == "b":
        return -1
    return 0

```

En el caso de nuestro tempo tendremos que diferenciar nuestra letra minúscula después de nuestra nota para ser interpretada como cierto valor numérico que será nuestra x en esta fórmula $1000/x$.

```

def get_temp(value):
    if value == "w":
        return 1000
    elif value == "h":
        return 500
    elif value == "q":
        return 250

```

```

elif value == "e":
    return 125
elif value == "s":
    return 62.5
elif value == "t":
    return 33.3
elif value == "f":
    return 16.6
return 0

def get_modtemp(valueTemp):
    if valueTemp == "t":
        return 2/3
    elif valueTemp == "3":
        return 2/3
    elif valueTemp == "5":
        return 4/5
    elif valueTemp == "7":
        return 6/7
    elif valueTemp == "9":
        return 8/9
    elif valueTemp == ".":
        return .25
    elif valueTemp == "tt" or valueTemp == "33":
        return 4/9
    return 1

```

Teniendo en cuenta estas funciones continuaremos creando la función que crea la nota tomando en cuenta la nota, la octava y la duración después de obtener los valores los almacena en su respectivo arreglo.

```

def create_note(conc):
    get_note = 0
    octave_result = 0
    get_duration = 0
    dot = 1.25
    dot_mod = False

```

```

# concatenación de notas, octavas y duración
for i in range (0, len(conc)):
    get_note += get_letter(conc[i])
    get_note += get_accidental(conc[i])
    if conc[i] == "-":
        octave_result += get_octave(conc[i+1]) * -1
        i += 1
    else:
        if octave_result == 0:
            octave_result += get_octave(conc[i])

    get_duration += get_temp(conc[i])

    if conc[i] == ".":
        dot_mod = True
        dot += get_modtemp(conc[i])

    if dot_mod == True and len(conc) == i+1:
        get_duration *= dot
    else:
        if (conc[i] == '3' or conc[i] == 't') and len(conc) !=
i+1:
            get_duration *= get_modtemp(conc[i] + conc[i+1])
            i+=1
        else:
            get_duration *= get_modtemp(conc[i])

# concatenacion (note|octave|duration)
note_arr.append(get_note)
octave_arr.append(octave_result)
duration_arr.append(get_duration)

```

Para extraer la información de nuestro archivo txt crearemos una función que lea el archivo y extraiga toda la información con ayuda de las funciones strip() y split().

```
def read_txt(file):
```



```

if not txt_file.fullmatch(file):
    # para que el usuario no tenga que escribir .txt
    file += '.txt'

# extractor de información del archivo
with open(file, 'r') as file_input:
    # variable para guardar la info
    file_info = file_input.readline()
    info = True
    list = []
    for line in file_input:
        if info == True:
            list.append(file_info)
            info = False
        list.append(line)

    list_data = ""

    # convertidor de info a string
    for element in list:
        new_element = element.strip().split(",")
        list_data += new_element[0] + chr(13)

    return list_data

```

Por último tenemos que implementar nuestro “for” donde tendremos todas las funciones `fullmatch()` que nos ayudarán a validar la información dependiendo si siguen las expresiones regulares que les corresponde ya sea nota, descanso u octava.

```

for i in range(0, len(user_input)):
    if i <= flag and i != 0:
        continue

    user_input_a = ord(user_input[i])

    if user_input_a != 13:
        conc += user_input[i]

```

```

else:
    token_arr.append(5)
    line_error += 1

    # detector de comentarios y palabras reservadas
if conc == '#':
    comment_a = comments(i, user_input)
    flag = comment_a[0]
    conc += comment_a[1]

    if BAR_r.fullmatch(conc):
        token_arr.append(4)
        conc = acceptor('\nSe detecto palabra reservada #BAR')
        token_arr.append(5)

    if comment.fullmatch(conc):
        accepted_message = 'Se identifico comentario: ' + conc
        conc = acceptor(accepted_message)
        line_error += 1
        token_arr.append(4)
        token_arr.append(5)
        continue

    line_error += 1

    # detector de nota
if note_m.fullmatch(conc):
    if user_input[i] == 's':
        time = 250

    conc += ciclo(i, user_input)

    if note2_m.fullmatch(conc):
        create_note(conc)
        accepted_message = 'Se detecto una nota musical: '
+ conc
        conc = acceptor(accepted_message)

```

```

        token_arr.append(1)
        continue
    else:
        line_error += 1
        conc = input_error(conc)
        lex = False
        break

if pause.fullmatch(conc):
    conc += ciclo(i, user_input)

    if pause2.fullmatch(conc):
        create_note(conc)
        accepted_message = 'Se detecto un descanso: ' +
conc
        conc = acceptor(accepted_message)
        token.append(2)
        continue
    else:
        line_error += 1
        conc = input_error(conc)
        lex = False
        break

if particion.fullmatch(conc):
    conc = acceptor('\nlimitador de medida')
    token.append(3)
    continue

if user_input_a == 13 or user_input[i] == '':
    if name.fullmatch(conc):
        accepted_message = 'Se detecto nombre: ' + conc
        conc = acceptor(accepted_message)
        token.append(0)
        continue

    if conc == " " or conc == "":

```

```

        conc = ""
    else:
        if user_input_a != 13:
            line_error += 1
            conc = input_error(conc)
            lex = False
            break

```

Dentro de nuestra carpeta donde tenemos nuestro programa tendremos que colocar nuestros archivos txt para poder escuchar la melodía dentro del programa una vez que lo corramos. Debería de verse de la siguiente manera.

```

In [63]: runfile('/Users/leonardogzz/Desktop/Cuarto Semestre/implementacion de
la computacion/DSL/untitled1.py', wdir='/Users/leonardogzz/Desktop/Cuarto
Semestre/implementacion de la computacion/DSL')

Escribe el nombre del archivo para compilarlo: escala_musical
Compilando archivo...

Iniciando el analisis de lexico...

Analizando sintaxis...

Reproduciendo canción...

Fin de análisis

```