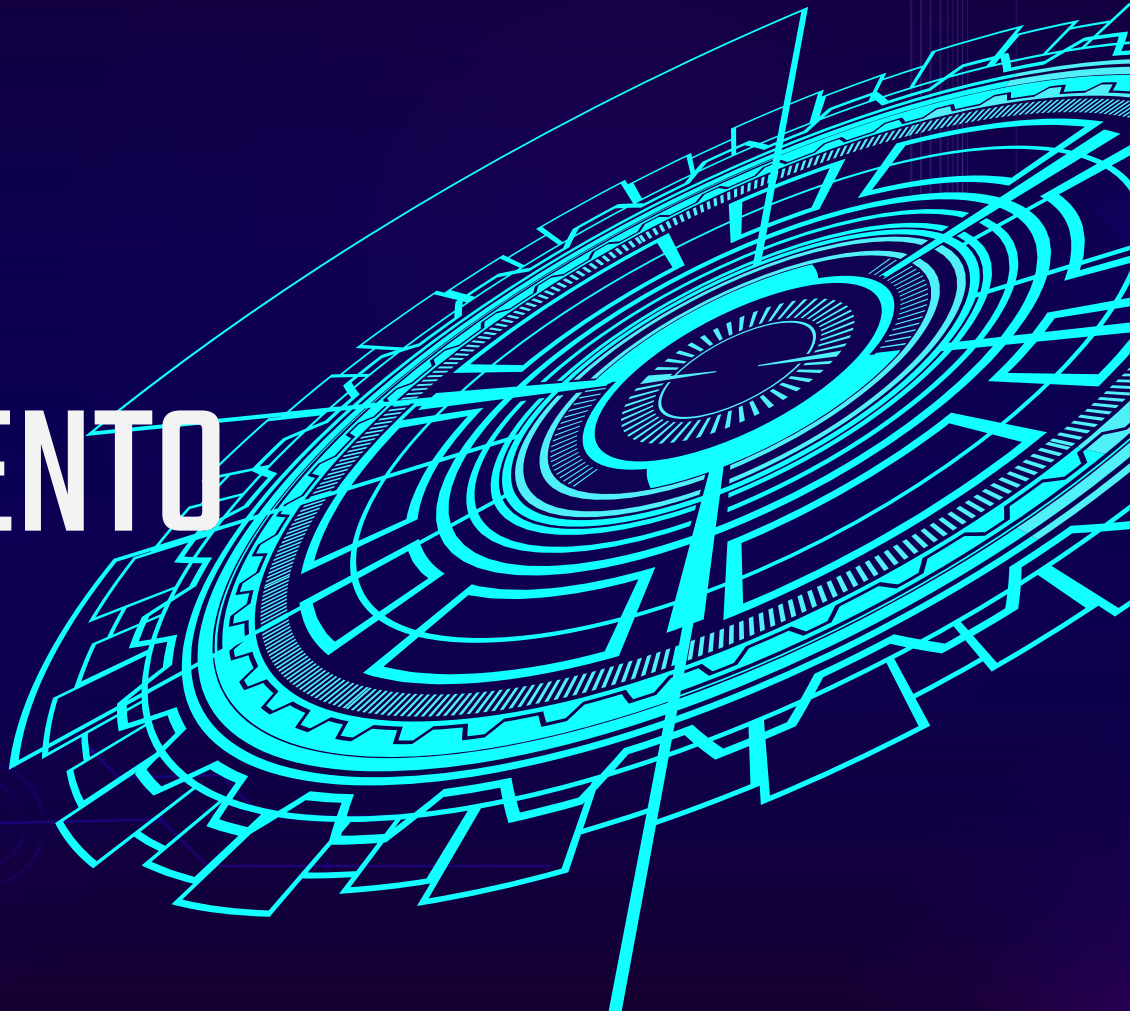
# SONIDO Y PROCESAMIENTO



Andres Felipe Quebrada Agudelo
1225091462

# TABLA DE CONTENIDO


01 Graficar archivo de audio

02 Transformar frecuencia

03 Generar audio

04 Sintetizar tonos

05 Reconocer palabras


# GRAFICAR AUDIO

Leer una archivo de audio (.wav), toma los datos de este archivo y los grafica (Amplitud, tiempo)

***Librerías:***

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
```

# ¿CÓMO LO HACEMOS?

01 Leer archivo de audio

02 Normalizar la señal

03 Extraemos un conjunto de valores

04 Generamos el eje de las X (tiempo)

05 Graficamos

01

```
# Lee el archivo de audio
frecuencia_muestreo, senial = wavfile.read('prueba.wav')
```

02

```
# Normalizar la señal
senial = senial / np.power(2, 15)
```

03

```
# Extraer los primeros 50 valores
senial = senial[:50]
```

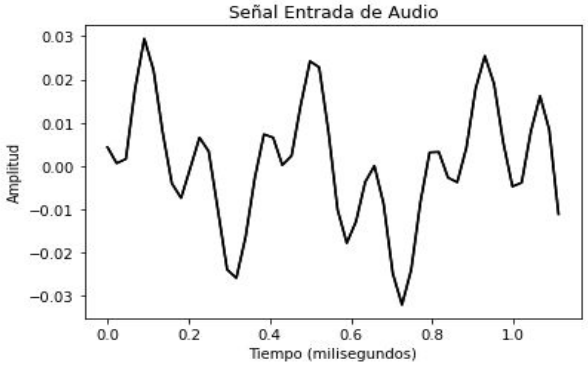
04

```
# Construir el eje de tiempo en milisegundos
eje_del_tiempo = 1000 * np.arange(0, len(senial), 1) / float(frecuencia_muestreo)

# Dibujar la señal de audio
plt.plot(eje_del_tiempo, senial, color='black')
plt.xlabel('Tiempo (milisegundos)')
plt.ylabel('Amplitud')
plt.title('Señal Entrada de Audio')
plt.show()
```

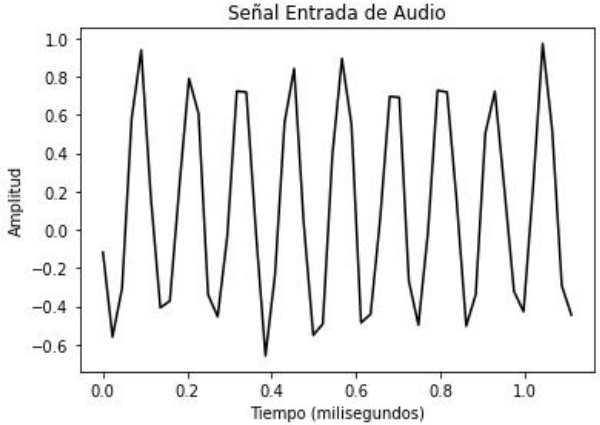
05

# EJEMPLOS

```
Tamaño señal: (1386306, 2)
Tipo de dato: int16
Duracción de la señal: 31.44 seconds
Frecuencia de muestreo:  44100
```




```
Tamaño señal: (132300,)
Tipo de dato: int16
Duracción de la señal: 3.0 seconds
Frecuencia de muestreo:  44100
```

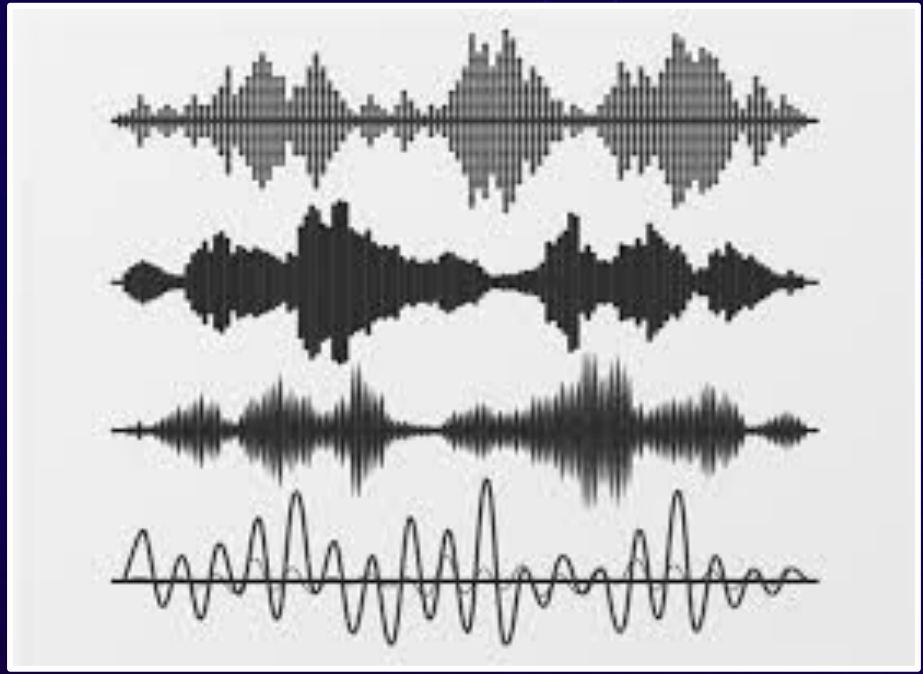


# TRANSFORMAR A FRECUENCIA

Transformamos la señal en función del tiempo en una señal en función de la frecuencia.

***Librerías:***

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
```



# ¿CÓMO LO HACEMOS?

Leer archivo de audio

Normalizar la señal

Extraemos la longitud completa y media

Transformada de Fourier

Ajustar señal

Eje X, eje Y y graficar

$$f_{(t)} = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F(\omega)e^{j\omega t}d\omega \quad \rightarrow \quad Antitransformada\ de\ Fourier$$

$$F(\omega) = \int_{-\infty}^{+\infty} f_{(t)}e^{-j\omega t}dt \quad \rightarrow \quad Transformada\ de\ Fourier$$

01

```
# Leer el archivo de audio
frecuencia_de_muestreo, senial = wavfile.read('palabra_hablada.wav')
```

02

```
# Normalizar los valores
senial = senial / np.power(2, 15)
```

03

```
# Extraer la longitud de la señal de audio
longitud_senial = len(senial)

# Extraer la mitad de la longitud
mitad_longitud = np.ceil((longitud_senial + 1) / 2.0).astype(int)
```

04

```
# Aplicar la Transformada de Fourier
frecuencia_senial = np.fft.fft(senial)

# Normalización
frecuencia_senial = abs(frecuencia_senial[0:mitad_longitud]) / longitud_senial

# Cuadrado
frecuencia_senial **= 2

# Extrae la longitud de la señal de frecuencia transformada
len_fts = len(frecuencia_senial)
```

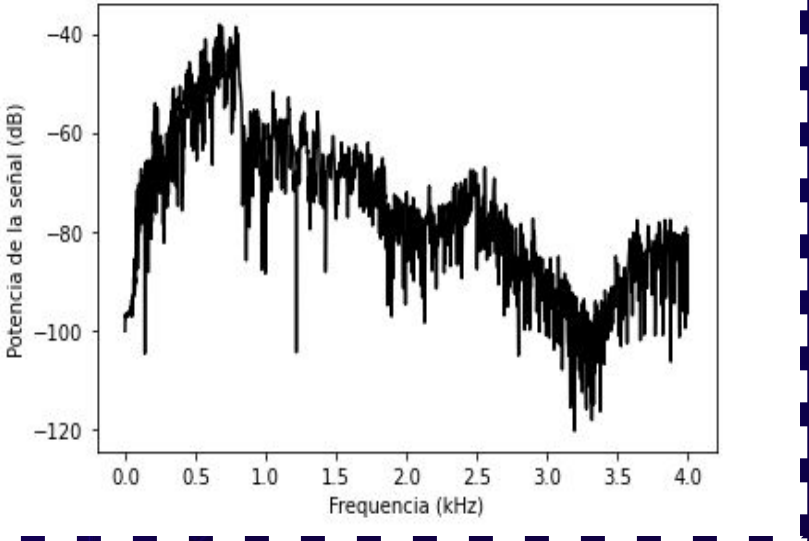
05

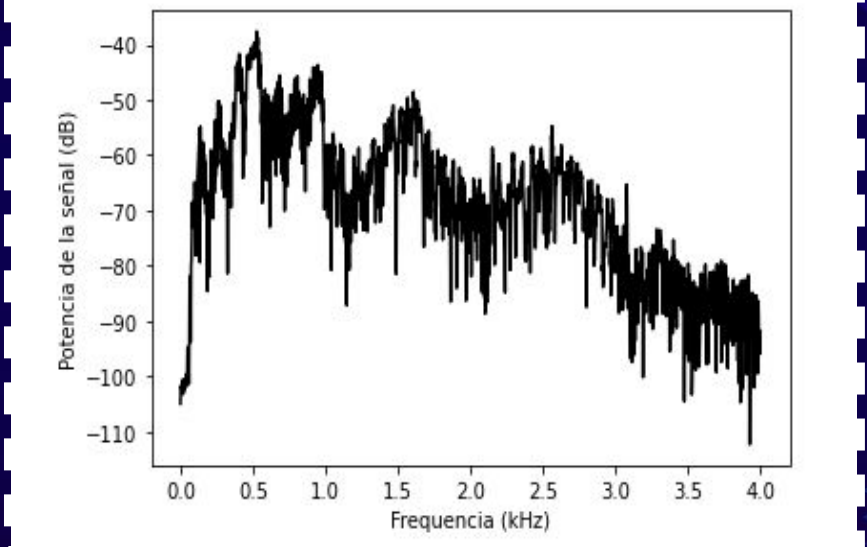
```python
# Ajustar la señal para casos pares e impares
if longitud_senial % 2:
    frecuencia_senial[1:len_fts] *= 2
else:
    frecuencia_senial[1:len_fts-1] *= 2
```

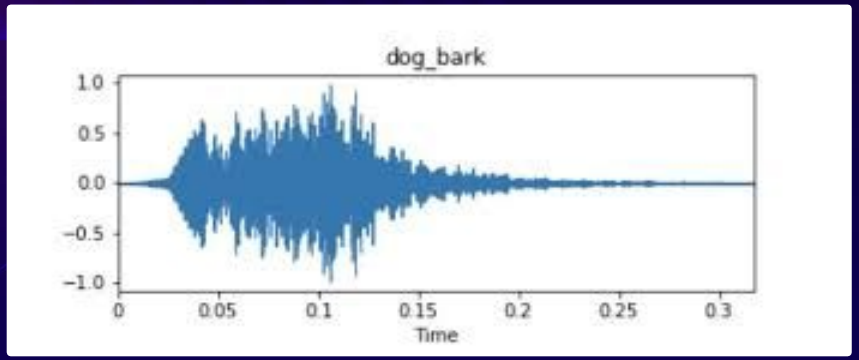
06

```python
# Extraer el valor de potencia en dB
potencia_senial = 10 * np.log10(frecuencia_senial)

# Construir el eje X
eje_x = np.arange(0, mitad_longitud, 1) * (frecuencia_de_muestreo / longitud_senial) / 1000.0

# Graficar la figura
plt.figure()
plt.plot(eje_x, potencia_senial, color='black')
plt.xlabel('Frequencia (kHz)')
plt.ylabel('Potencia de la señal (dB)')
plt.show()
```

# EJEMPLOS


Potencia de la señal (dB)
−40
−60
−80
−100
−120
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Frequencia (kHz)


Potencia de la señal (dB)
−40
−50
−60
−70
−80
−90
−100
−110
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Frequencia (kHz)




# GENERAR AUDIO

```
# Especificar los parámetros del audio
duracion = 4  # in seconds
frecuencia_muestreo = 44100  # in Hz
frecuencia_tono = 784
valor_minimo = -4 * np.pi
valor_maximo = 4 * np.pi
```

***Librerías:***
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import write

***Parámetros de entrada:***
duración (s).
frecuencia de muestreo(hz).
frecuencia de tono.
valor mínimo.
valor máximo.

# ¿CÓMO LO HACEMOS?

01 Generar archivo de audio

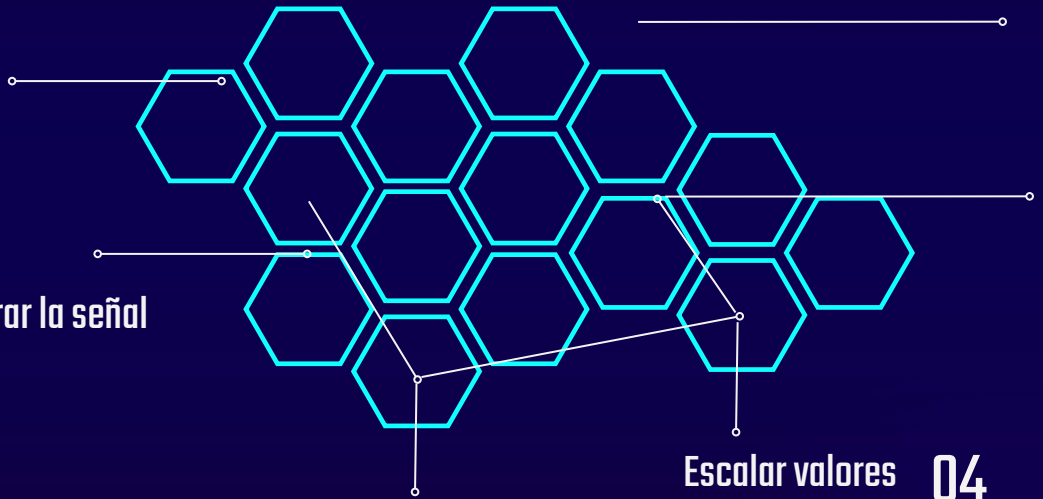
02 Generar la señal

03 Agregar algún ruido

04 Escalar valores

05 Guardamos la señal

06 Graficar audio



01

```
# Archivo de salida en el cual se grabará el audio
archivo_salida = 'audio_generado.wav'
```

02

```
# Generar la señal de audio
t = np.linspace(valor_minimo, valor_maximo, duracion * frecuencia_muestreo)
senial = np.sin(2 * np.pi * frecuencia_tono * t)
```

03

```
# Agregar algún ruido a la señal
ruido = 0.5 * np.random.rand(duracion * frecuencia_muestreo)
senial += ruido
```

04

```
# Escalar a valores enteros de 16 bits
factor_escalamiento = np.power(2, 15) - 1
senial_normalizada = senial / np.max(np.abs(senial))
senial_escalada = np.int16(senial_normalizada * factor_escalamiento)
```

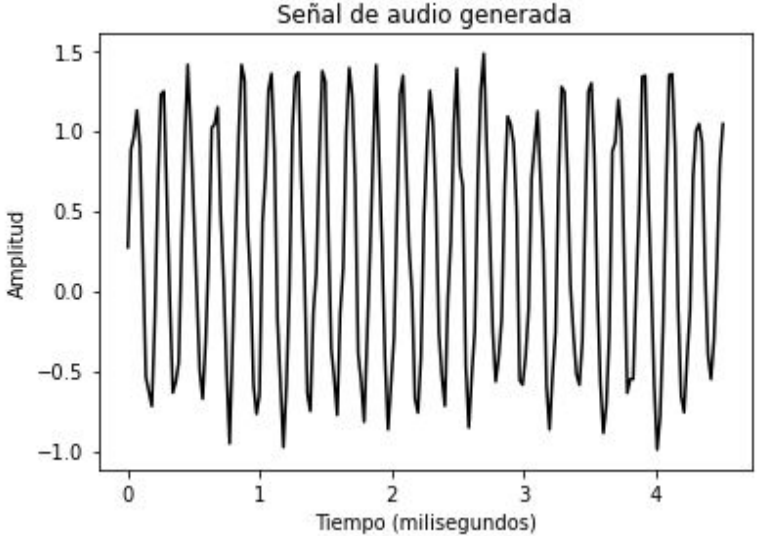
05

```
# Almacenar la señal de audio en el archivo de salida
write(archivo_salida, frecuencia_muestreo, senial_escalada)
```

06

```
# Extraer los primeros 200 valores de la señal de audio
senial = senial[:200]

# Construir el eje del tiempo en milisegundos
eje_tiempo = 1000 * np.arange(0, len(senial), 1) / float(frecuencia_muestreo)

# Graficar la señal de audio
plt.plot(eje_tiempo, senial, color='black')
plt.xlabel('Tiempo (milisegundos)')
plt.ylabel('Amplitud')
plt.title('Señal de audio generada')
plt.show()
```

# EJEMPLOS


Señal de audio generada
Amplitud
1.5
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
Tiempo (milisegundos)

# SINTETIZAR TONOS

## ¿Qué es sintetizar?

- Juntar elementos para crear un nuevo conjunto
- Generar sonido a partir de medio no acústico.

***Librerías:***
import ***json***
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import write

# Función para sintetizar

```python
# Sintetizar el tono basado en los parámetros de entrada
def sintetizador_tono(frecuencia, duracion, amplitud=1.0, frecuencia_muestreo=44100):
    # Construir el eje de tiempo
    eje_tiempo = np.linspace(0, duracion, duracion * frecuencia_muestreo)

    # Construir la señal de audio
    senial = amplitud * np.sin(2 * np.pi * frecuencia * eje_tiempo)

    return senial.astype(np.int16)
```

01

```
# Nombres de los archivos de salida
archivo_tono_generado = 'tono_generado.wav'
archivo_secuencia_tono_generada = 'secuencia_de_tono_generada.wav'
```

02

```
# Source: http://www.phy.mtu.edu/~suits/notefrecuencias.html
archivo_mapeo = 'tone_mapping.json'

# Cargue el mapa de tono a frecuencia desde el archivo de mapeo
with open(archivo_mapeo, 'r') as f:
    mapa_tonos = json.loads(f.read())
```

03

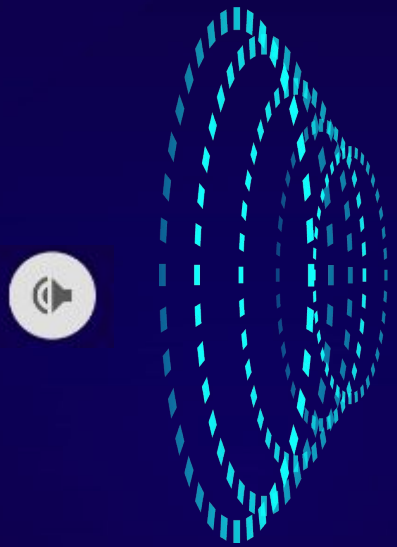
```
# Configure los parámetros de entrada para generar el tono 'F'
nombre_tono = 'F'
duracion = 3     # segundos

amplitud = 12000
frecuencia_muestreo = 44100    # Hz
```

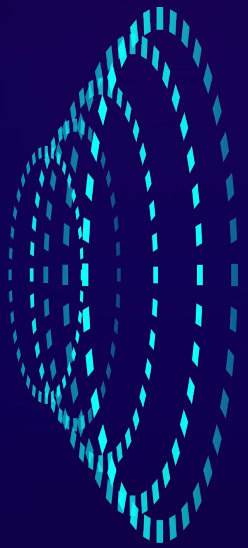
04

```
# Extrae la frecuencia del tono
frecuencia_tono = mapa_tonos[nombre_tono]

# Genere el tono usando los parámetros anteriores
tono_sintetizado = sintetizador_tono(frecuencia_tono, duracion, amplitud, frecuencia_muestreo)

# Escribe la señal de audio en el archivo de salida.
write(archivo_tono_generado, frecuencia_muestreo, tono_sintetizado)
```

```python
# Defina la secuencia de tonos junto con las duraciones correspondientes en segundos
tono_secuencia = [('G', 1), ('D', 1), ('F', 1), ('C', 1), ('A', 1)]

# Construya la señal de audio basándose en la secuencia anterior
senial = np.array([])
for item in tono_secuencia:
    # Obtiene el nombre del tono
    nombre_tono = item[0]

    # Extrae la frecuencia correspondiente del tono.
    frecuencia = int(mapa_tonos[nombre_tono])

    # Extrae la duración
    duracion = item[1]
    duracion = int(duracion)

    # Sintetizar el tono
    tono_sintetizado = sintetizador_tono(frecuencia, duracion, amplitud, frecuencia_muestreo)

    # Aañadir la señal de salida
    senial = np.append(senial, tono_sintetizado, axis=0)

# Guarda el audio en el archivo de salida
write(archivo_secuencia_tono_generada, frecuencia_muestreo, senial)
```

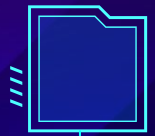
# EJEMPLOS





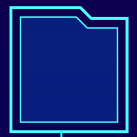
# RECONOCER PALABRAS

# LIBRERÍAS



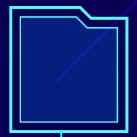
## os

Nos provee funcionalidades dependientes del sistema operativo



## argparse

facilita la escritura de interfaces de línea de comandos



## warnings

Alerta de algunas condiciones el programa

# LIBRERÍAS

## numpy/scipy

Fundamentales para el manejo de datos

## hmmlearn

Aprendizaje e inferencia no supervisados de modelos ocultos de Markov.

## python speech features

Coeficientes cepstrales de frecuencia de mel

```
args = build_arg_parser().parse_args()
input_folder = args.input_folder
```

```
# Define una función para analizar los argumentos de entrada
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Entrena la voz basada en HMM: \
             sistema de reconocimiento')
    parser.add_argument("--input-folder", dest="input_folder", required=True,
            help="Carpeta de entrada que contiene los archivos de audio para entrenamiento.")
    return parser
```

```
# python reconocer_texto.py --input-folder data
```

```python
# Define una clase para entrenar el HMM
class ModelHMM(object):
    def __init__(self, num_components=4, num_iter=1000):
        self.n_components = num_components
        self.n_iter = num_iter

        self.cov_type = 'diag'
        self.model_name = 'GaussianHMM'

        self.models = []

        self.model = hmm.GaussianHMM(n_components=self.n_components,
                covariance_type=self.cov_type, n_iter=self.n_iter)

    # 'training_data' es un array numpy 2D donde cada fila es 13-dimensional
    def train(self, training_data):
        np.seterr(all='ignore')
        cur_model = self.model.fit(training_data)
        self.models.append(cur_model)

    # corre el modelo HMM para realizar inferencia sobre la entrada de datos
    def compute_score(self, input_data):
        return self.model.score(input_data)
```

```python
# Construye el modelo HMM para cada palabra
speech_models = build_models(input_folder)
```

```python
# Define una función para construir un modelo para cada palabra
def build_models(input_folder):

    # Inicializar la variable para almacenar todos los modelos
    speech_models = []

    # Analiza el directorio de entrada
    for dirname in os.listdir(input_folder):

        # Obtiene el nombre del subfolder
        subfolder = os.path.join(input_folder, dirname)

        if not os.path.isdir(subfolder):
            continue

        # Extrae la etiqueta
        label = subfolder[subfolder.rfind('/') + 1:]

        # Inicializa las variables
        X = np.array([])

        # Crea una lista de archivos a ser utilizados para el entrenamiento
        # Se deja un archivo por folder para validación
        training_files = [x for x in os.listdir(subfolder) if x.endswith('.wav')][:-1]

        # Se itera a través de los archivos de entrenamiento y se construyen los modelos
        for filename in training_files:

            # Se extrae el path actual
            filepath = os.path.join(subfolder, filename)

            # Se lee la señal lde audio desde el archivo de entrada
            sampling_freq, signal = wavfile.read(filepath)

            # Se extraen las características MFCC
            with warnings.catch_warnings():
                warnings.simplefilter('ignore')
                features_mfcc = mfcc(signal, sampling_freq)

            # Se agrega a la variable X
            if len(X) == 0:
                X = features_mfcc
            else:
                X = np.append(X, features_mfcc, axis=0)

        # Se crea el modelo HMM
        model = ModelHMM()

        # Se entrena el HMM
        model.train(X)

        # Se almacena el modelo para la palabra actual
        speech_models.append((model, label))

        # Se reinicia la variable
        model = None
```

```python
#  Verifica los archivos -- el archivo 15 en cada subfolder
test_files = []
for root, dirs, files in os.walk(input_folder):
    for filename in (x for x in files if '15' in x):
        filepath = os.path.join(root, filename)
        test_files.append(filepath)

run_tests(test_files)
```

```python
# Define una función para ejecutar pruebas sobre los archivos de entrada
def run_tests(test_files):

    # Clasifica los datos de entrada
    for test_file in test_files:

        # Lee el archivo de entrada
        sampling_freq, signal = wavfile.read(test_file)

        # Extrae las características MFCC
        with warnings.catch_warnings():
            warnings.simplefilter('ignore')
            features_mfcc = mfcc(signal, sampling_freq)

        # Define variables
        max_score = -float('inf')
        output_label = None

        # Ejecuta el vector de características actual a través
        # de todos los modelos HMM y elige el que tenga el
        # puntaje más alto
        for item in speech_models:
            model, label = item
            score = model.compute_score(features_mfcc)
            if score > max_score:
                max_score = score
                predicted_label = label

        # Muestra la salida prevista
        start_index = test_file.find('/') + 1
        end_index = test_file.rfind('/')
        original_label = test_file[start_index:end_index]
        print('\nOriginal: ', original_label)
        print('Predicción:', predicted_label)
```

# GRACIAS