# Create a Voice Recorder using Python

#### **About Voice Recorder:**

A voice recorder helps to save audio recordings. This tool comes in handy when we intend to refer to some audio recordings for reference in the future. Hence we will see how to create an audio recorder using python in this project

#### **Python Voice Recorder:**

Here, we create a simple Voice recorder using tkinter, sounddevice, soundfile and threads in python.

#### **Project Prerequisites:**

The project is an introduction to sounddevice, soundfile and multithreading in python. Hence understanding the syntactic meaning should suffice. The modules necessary are tkinter, sounddevice, soundfile, threading, and queue. Since tkinter is already available in python, import to see if it works.

import tkinter

If it is imported successfully, it will not display any missing module error. Use the following command to install it on linux systems:

sudo apt-get install python3-tkinter, for ubuntu users

sudo pacman -S tk, for arch linux users.

Windows users can reinstall python or follow the steps given in [Tkinter installation on windows](https://tkdocs.com/tutorial/install.html#install-win-python). Use pip to install the other packages

* sounddevice: pip install sounddevice
* soundfile: pip install soundfile

#### **Download Voice recorder code:**

You can download the source code for the voice recorder in the given link: <add link>

#### **Project File Structure:**

Below is the flow of the project.

1. Importing libraries
2. Defining the functions for threading, recording voice and providing a callback
3. Creating the voice recorder GUI
4. Create the components and the button

##### **Importing libraries:**

#ProjectGurukul's Voice recorder

#Import necessary modules

import sounddevice as sd

from tkinter import \*

import queue

import soundfile as sf

import threading

from tkinter import messagebox

**Code Explanation:**

* **Import sounddevice as sd:** Sounddevice contains the functions to record voice input using the microphone.
* **from tkinter import \*:** Tkinter is the GUI library we use to create the GUI of the application.
* **import queue:** To use the data structure queue, we import it.
* **import soundfile as sf:** To save the audio recorded, we use soundfile.
* **import threading:** To simulate multithreading.
* **from tkinter import messagebox:** Messagebox contains prompts to display to the user when there is a missing input or when the file does not exist.

##### **Defining the functions for threading, recording voice and providing a callback:**

There are three functions in use.

1. threading\_rec
2. callback
3. record\_audio

We will see each of the functions and its purpose in detail.

* **threading\_rec():**

Threading allows parallel execution of various processes or tasks at once. The main use of threads here is to keep the other buttons active even if a function is running because of another button click.

#Functions to play, stop and record audio

#The recording is done as a thread to prevent it being the main process

def threading\_rec(x):

if x == 1:

#If recording is selected, then the thread is activated

t1=threading.Thread(target= record\_audio)

t1.start()

elif x == 2:

#To stop, set the flag to false

global recording

recording = False

messagebox.showinfo(message="Recording finished")

elif x == 3:

#To play a recording, it must exist.

if file\_exists:

#Read the recording if it exists and play it

data, fs = sf.read("trial.wav", dtype='float32')

sd.play(data,fs)

sd.wait()

else:

#Display and error if none is found

messagebox.showerror(message="Record something to play")

**Code explanation:**

* **def threading\_rec(x):** Declaration of the function to initiate threads to record and stop. It contains a parameter, x.
* **if x == 1:** If x == 1, start recording the audio by creating a thread and allowing the thread to call the function. threading.Thread() creates a thread and target calls the function it is assigned with. start() starts the execution of the thread.
* **elif x == 2**: If x==2, then it indicates to stop the audio recording. It is done by reassigning a global variable to False. This global variable is a test condition in the recording function, record\_audio. Setting it to False terminates the while loop which records this. Then notify the user with a prompt, messagebox.showinfo, to denote the completion of the audio recording.
* **x == 3:** To play the audio, it must be recorded. Hence we check if file\_exists is True indicating a recording previously done by the user and it is played using sf.read() which accepts an input file and a datatype to read it from. ‘float32’ is the default datatype. To play the audio, use play() where you play only the data and not the sample rate, fs. Use wait to create a delay to play the audio. The audio will not be played otherwise. If file\_exists does not exist, raise a prompt using messagebox.showerror with a warning message.

-----------------------------------------------

* **callback():**

#Fit data into queue

def callback(indata, frames, time, status):

q.put(indata.copy())

**Code explanation:**

* **def callback(indata, frames, time, status):** Declare a function callback to record data into the queue. This method is necessary and exists in the documentation of sounddevice.
* **q.put(indata.copy()):** Queue is a data structure that follows FIFO. It means the first element to get into the queue is the first to exit the queue. A queue contains two methods: pop/get and push/put. Pop/get extracts the first content in the queue. Push/put puts the data into the queue

-----------------------------------------------

* **record\_audio():**

#Recording function

def record\_audio():

#Declare global variables

global recording

#Set to True to record

recording= True

global file\_exists

#Create a file to save the audio

messagebox.showinfo(message="Recording Audio. Speak into the mic")

with sf.SoundFile("trial.wav", mode='w', samplerate=44100,

channels=2) as file:

#Create an input stream to record audio without a preset time

with sd.InputStream(samplerate=44100, channels=2, callback=callback):

while recording == True:

#Set the variable to True to allow playing the audio later

file\_exists =True

#write into file

file.write(q.get())

**Code explanation:**

* **def record\_audio():** Declaration of the recording function
* **global recording:** Declaration of a global variable. This global variable controls the audio recording loop
* **recording= True:** Setting record to True to initiate recording
* **global file\_exists:** Declare a global variable to indicate the presence of a recording. A variable when made global, is accessible by all functions in the code irrespective of where it is declared.
* **messagebox.showinfo(message="Recording Audio. Speak into the mic"):** Display a prompt to the user to record the audio
* **sf.SoundFile():** Create a file object to save the audio recording. The parameters are: 1. Name of the file with .wav extension, 2. Mode which denotes the mode to write the file, 3. samplerate which is the frames per second, 4. Channels denoting the number of input and output channels.
* **sd.InputStream():** This function records the voice input without a preset duration. The additional parameter here is the callback which contains the array to put the input data
* **while recording == True:** Recording will happen only if it is set to true.
* **file\_exists =True:** Marking the existence of a file if recoding happens
* **file.write(q.get()):** Writing the contents of the queue into the soundfile created, by extracting the queue contents using get()

##### **Creating the voice recorder GUI:**

#Define the user interface

voice\_rec = Tk()

voice\_rec.geometry("360x200")

voice\_rec.title("ProjectGurukul's Voice Recorder")

voice\_rec.config(bg="#107dc2")

#Create a queue to contain the audio data

q = queue.Queue()

#Declare variables and initialise them

recording = False

file\_exists = False

**Code explanation:**

* **voice\_recorder = Tk():** Creation of an object to use the widgets in Tkinter and define the window of the app
* **voice\_rec.geometry("360x200"):** Define the dimensions of the window using geometry. It is of the format ‘width x height’
* **voice\_rec.title("ProjectGurukul's Voice Recorder"):** Assign a title for the application
* **voice\_rec.config(bg="#107dc2")**: To use a background colour, use config with the parameter bg to assign a colour. The colour can be given using the name or it can be specified using the colour code.
* **q = queue.Queue():** Initialize the queue to contain the voice data
* **recording = False, file\_exists = False:** Declare and define the variable to False

##### **Create the components and the button:**

#Label to display app title

title\_lbl = Label(voice\_rec, text="ProjectGurukul's Voice Recorder", bg="#107dc2").grid(row=0, column=0, columnspan=3)

#Button to record audio

record\_btn = Button(voice\_rec, text="Record Audio", command=lambda m=1:threading\_rec(m))

#Stop button

stop\_btn = Button(voice\_rec, text="Stop Recording", command=lambda m=2:threading\_rec(m))

#Play button

play\_btn = Button(voice\_rec, text="Play Recording", command=lambda m=3:threading\_rec(m))

#Position buttons

record\_btn.grid(row=1,column=1)

stop\_btn.grid(row=1,column=0)

play\_btn.grid(row=1,column=2)

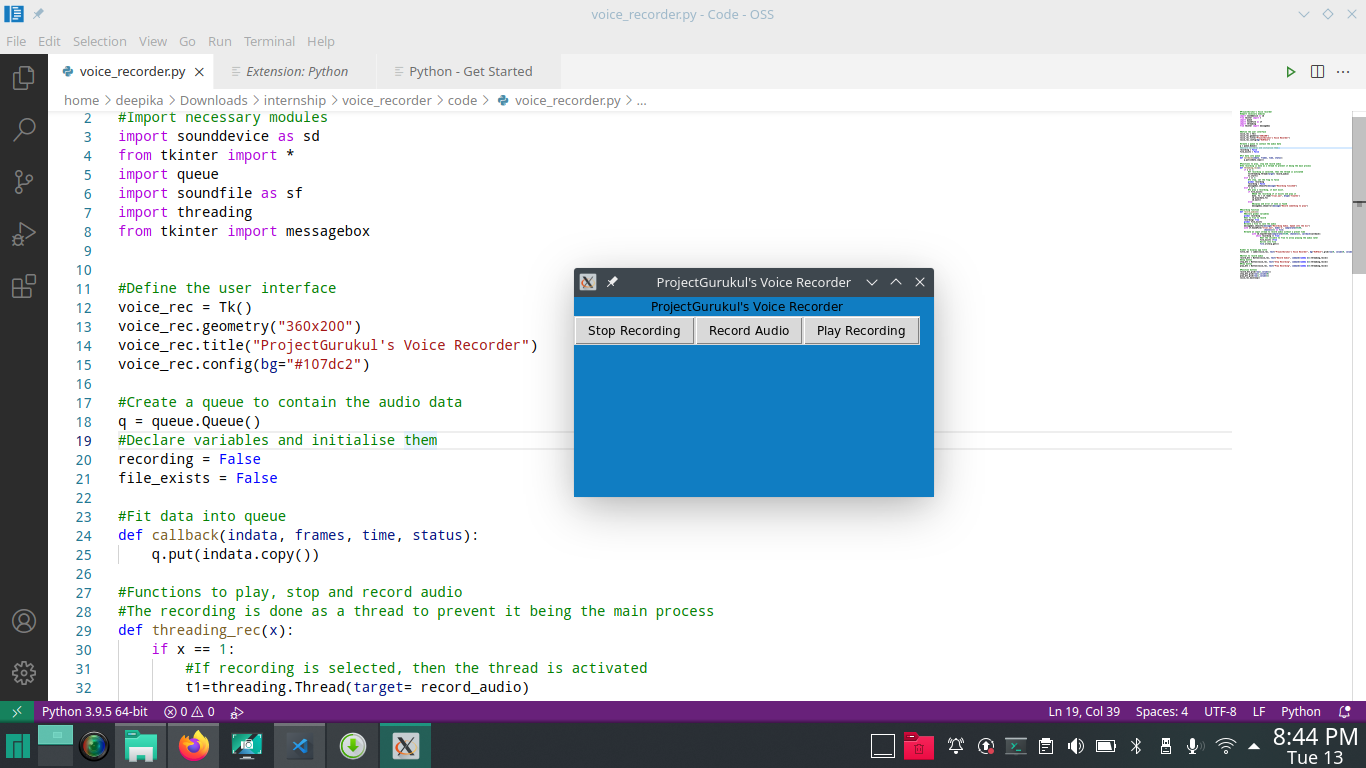
voice\_rec.mainloop()

**Code explanation:**

* **title\_lbl:** To display non editable text which cannot be copied, we use Label. It contains the window of the application and the text to be displayed. Add an optional ‘bg’ equivalent to the window’s background colour or ignore the tag to allow default value. Positioning a widget makes it visible on the app. Hence we use grid, which divides the window into rows and columns. To position the label, we use row 0, column 0 and set a columnspan of 3 to make it appear across the other columns
* **record\_btn, stop\_btn, play\_btn:** To activate the record function, stop or play the audio, we use buttons. Create the buttons using Button() widget and specify the parameters such as main window of the app, text of the button and the function the button must activate. Here we use only one function for all the buttons, threading\_rec. Thus we use lambda to pass different parameters, x values, to the threading\_rec function.
* **grid(row=X,column=X):**Place the buttons using grid function. Specify the row and column to place the widget
* **voice\_rec.mainloop():** To activate the window and execute the application mainloop is used. When the user exits the application, the control flows to after mainloop thus terminating the app

#### **Project output:**

Record the audio and view the output



#### **Summary**

Thus we created a simple voice recorder from scratch. The project introduces audio libraries such as sounddevice and soundfile. We also came across a simple example of threading concept and its usage, along with a data structure.