**Project Name:**

**Music Player in Python**

**Personal Details:**

**Name**: VINNAKOTA AKHIL SAI SRINIVAS

**Intern ID-** CC40693 **Designation-** Python Development Intern  
**Batch-** Jul 2023

**Mail id:** [20981a4960@raghuenggcollege.in](mailto:20981a4960@raghuenggcollege.in)

Context Introduction and usage:

Music players are very popular these days because music has become popular. This is why music players like Spotify, Wynk, and Gaana have become popular because they bring music closer to you. Well, in this python music player project, we are going to help you make your own GUI based mp3 music player and have fun while doing it!

Objective:

The objective of this project is to create a GUI based python music player from scratch using python. For this project, you will need intermediate knowledge of the Tkinter widgets, basic knowledge about tkinter.filedialog, pygame.mixer, and os libraries.

Project Prerequisites:

To create this music player python project, you will need intermediate understanding of Python Tkinter, and basic knowledge about the Pygame and ttkwidgets libraries.

1. **Tkinter –** To create the GUI for the project.
2. **Pygame.mixer –** This is a pygame module that is used to load and play music.
3. **OS –** To fetch the playlist of songs from the specified directories.

Not all the libraries come pre-installed with Python, so you will have to run the following command to install the required libraries.

Project File Structure:

Here are the steps you will need to perform to build python music player project:

STEP 1:

Importing all the libraries

STEP 2”

Initializing the root window and pygame.mixer

STEP 3:

Defining the play, stop, pause, resume and load functions

STEP 4:

Creating the LabelFrames and StringVar variables

STEP 5:

Placing all the objects in all the three LabelFrames

STEP 6:

Creating the final Label that will display the status of the song

STEP 1:

#### **1. Importing all the libraries:**

# Importing all the necessary modules

from tkinter import \*

from tkinter import filedialog

import pygame.mixer as mixer # pip install pygame

import os

#### STEP 2**:**

#### **2.Initializing the root window and pygame.mixer:**

a. Initializing pygame.mixer:

# Initializing the mixer

mixer.init()

b. Initializing the root window:

# Creating the master GUI for python music player

root = Tk()

root.geometry('700x220')

root.title('PythonGeeks Music Player')

root.resizable(0, 0)

# Finalizing the GUI

root.update()

root.mainloop()

STEP 1 AND 2 EXPLANATION:

**Code Explanation:**

* Now that we have initialized the GUI window and the mixer file, we need to create user-defined functions that load, play, pause, resume and stop the music using the OS and mixer modules.
* In the **load** function, we will ask the user for a directory that has the audio files and then insert all the files in python music player as values in the ListBox widget we will provide as an argument to the function.
  + The **os.chdir()** command is used to change the current working directory to the specified path.
  + The **tkinter.filedialog.askdirectory()** method is used to request a directory from the user.
  + The **os.listdir()** command is used to list all the files in the current working directory in the form of a list.
  + The **.insert()** function, that takes the index, and \*elements arguments, is used to insert new element(s) to the Listbox widget on the index parameter.
* The ***play*** function, which loads and plays a file, requires 3 arguments: 2 StringVar objects and a ListBox object where the StringVar objects manipulate the text in the Labels that display the current song and its status:
  + Firstly, we set the song\_name argument to the name of the song by getting the selected option from the ListBox object and set the status to “Playing”.
  + The **.set()** method of a StringVar class changes the value of the StringVar object.
  + The **.get()** method of the ListBox class is used to get certain values from the object. When ACTIVE is provided as an argument to it, it gives you the selected value in the object.
* The stop(), pause() and resume() functions are all provided with only one StringVar object as argument

STEP 4:

#### **Creating the LabelFrames and StringVar variables:**

***Source*** ***code:***

# All the frames

song\_frame = LabelFrame(root, text='Current Song', bg='LightBlue', width=400, height=80)

song\_frame.place(x=0, y=0)

button\_frame = LabelFrame(root, text='Control Buttons', bg='Turquoise', width=400, height=120)

button\_frame.place(y=80)

listbox\_frame = LabelFrame(root, text='Playlist', bg='RoyalBlue')

listbox\_frame.place(x=400, y=0, height=200, width=300)

# All StringVar variables

current\_song = StringVar(root, value='<Not selected>')

song\_status = StringVar(root, value='<Not Available>')

***Explanation***:

* A ***LabelFrame*** is a container in Python Tkinter GUIs that acts as a container for different window layouts.
  + The **master** parameter is the parent window it is associated with.
  + The **text** parameter is the text that will be displayed on the frame.
  + The **width**, **height** parameters are used to specify the width and height of the widget.
  + The **bg** parameter specifies the background color.
* The **StringVar** class is used to manipulate and edit text in Labels, Entry widgets, and OptionMenus.
  + The **value** parameter denotes the initial value of the widget. Default is ”.
  + The **master** parameter is the same as in LabelFrames.
* The **.pack()** method is one of the 3 Tkinter geometry manager methods that is used to position a widget in its parent using abscissa and ordinate points as though the parent widget/window is a Cartesian Plane. The default is (0, 0) , which is also the North West corner of the parent widget/window.
  + The**x, y** parameters denote the horizontal and vertical offsets of the widget this method is associated with.

STEP 5:

#### **5.Placing all the widgets in all the three LabelFrames:**

***Source*** ***code:***

# Playlist ListBox

playlist = Listbox(listbox\_frame, font=('Helvetica', 11), selectbackground='Gold')

scroll\_bar = Scrollbar(listbox\_frame, orient=VERTICAL)

scroll\_bar.pack(side=RIGHT, fill=BOTH)

playlist.config(yscrollcommand=scroll\_bar.set)

scroll\_bar.config(command=playlist.yview)

playlist.pack(fill=BOTH, padx=5, pady=5)

# SongFrame Labels

Label(song\_frame, text='CURRENTLY PLAYING:', bg='LightBlue', font=('Times', 10, 'bold')).place(x=5, y=20)

song\_lbl = Label(song\_frame, textvariable=current\_song, bg='Goldenrod', font=("Times", 12), width=25)

song\_lbl.place(x=150, y=20)

# Buttons in the main screen

pause\_btn = Button(button\_frame, text='Pause', bg='Aqua', font=("Georgia", 13), width=7,

command=lambda: pause\_song(song\_status))

pause\_btn.place(x=15, y=10)

stop\_btn = Button(button\_frame, text='Stop', bg='Aqua', font=("Georgia", 13), width=7,

command=lambda: stop\_song(song\_status))

stop\_btn.place(x=105, y=10)

play\_btn = Button(button\_frame, text='Play', bg='Aqua', font=("Georgia", 13), width=7,

command=lambda: play\_song(current\_song, playlist, song\_status))

play\_btn.place(x=195, y=10)

resume\_btn = Button(button\_frame, text='Resume', bg='Aqua', font=("Georgia", 13), width=7,

command=lambda: resume\_song(song\_status))

resume\_btn.place(x=285, y=10)

load\_btn = Button(button\_frame, text='Load Directory', bg='Aqua', font=("Georgia", 13), width=35, command=lambda: load(playlist))

load\_btn.place(x=10, y=55)

Code Explanation:

* Now that all the LabelFrame objects have been defined, let us place all the components in them.
  + Horizontal and vertical offsets of the widgets, using the .place() method have to be according to their parent widget, not the master window.
* In the playlist LabelFrame we will define a Listbox with a Scrollbar packed to it.
* The **ListBox** class is used to add a Listbox widget on the window, which displays multiple values in different lines to choose from.
  + **master** and **font** parameters have the same description as in the other widgets.
  + **selectbackground** parameter defines the color of the background when a value is selected.
  + **selectmode** parameter specifies how many values can be selected at once. Default is ‘browse’ for single selection, and other options include ‘multiple’ and ‘extended’ that allow multiple selection.
  + **config()** method is used to configure some other parameters to the widget it is associated with.
  + **yscrollcommand** parameter defines the Scrollbar object that will be associated with the widget. Similarly, the xscrollcommand defines the vertical Scrollbar.
* The **ScrollBar** class is used to add a Scrollbar to the music player window.
  + **master** parameter and **.command()** and **.config()** method have the same description as in the other widgets.
  + **orient** parameter defines whether the Scrollbar will control the widget vertically, or horizontally.
  + **set()** method is used to set the Scrollbar to another widget.
  + In the current\_song LabelFrame, we will define 2 Labels; one with a constant text and the other with variable text.
* The ***Label*** widget is used to display static text on its parent widget.
  + **master, width** and **bg** are the same as they are in the Label widget.
  + **font** parameter is used to designate the font family and font effects of the text on the widget.
  + **textvariable** parameter is a Tkinter variable that will automatically update the value in the widget when the argument provided is updated.
* In the control\_panel LabelFrame, we will define the buttons whose commands we defined in the last-to-last step.
* The **Button** class is used to add a button to the GUI application that executes a command when it is pressed.
  + **master, text, bg,** font and width parameters have the same description as in the other widgets.
  + **command** parameter is used to define the command the button will execute when it is pressed. It may be a statement, function with or without arguments. The functions without arguments can be executed without anything extra, but you need to use the lambda: keyword to assign functions with arguments.
* The **.pack()** method, another Tkinter geometry manager method, is used to pack a widget as though the master window or the parent widget was a spreadsheet, in the form of rows and columns.
  + The **side** parameter of the .pack() method is used to specify where the widget will be placed on the parent widget or the master window.
  + The **fill** parameter defines whether the widget will fill the horizontal (X has to be provided as argument) or the vertical (Y has to be provided as argument) parts of the parent widget/window or the entire parent (BOTH has to be provided as argument).
  + The padx, pady parameters define how many pixels to leave between the widget and the nearby borders (horizontal, vertical).

#### **STEP 6:**

#### **Creating the final Label that will display the status of the song:**

***Source Code:***

# Label at the bottom that displays the state of the music

Label(root, textvariable=song\_status, bg='SteelBlue', font=('Times', 9), justify=LEFT).pack(side=BOTTOM, fill=X)

***Explanation:***

* The **justify** parameter of the Label class denotes the alignment of the text on the widget.

******SAMPLE MUSIC PLAYER PROJECT OUTPUT:**