



# Music Player with Python

**In this article, you will learn to create a simple music player application in Python.**

[Music is Love](#)

[Our kick-starter](#)

[Let's get started with our code:](#)

[Step 1 - Get the Libraries](#)

[Step 2: Creating GUI of our Music Player](#)

- [1. Inserting a list box.](#)
- [2. Getting songs from the directory and inserting them to the Listbox.](#)
- [3. Adding Play, Pause, Unpause, Previous, Next, and Exit button to the Music Player.](#)
- [4. Functions for play, pause,unpause, previous,next and exit buttons.](#)
- [5. Binding buttons to their respective functions.](#)
- [6. How to change the volume of music ?!](#)
- [7. A Song Label](#)

[Ready to Spin the Music!](#)

## Music is Love

We all love listening to music. It could be when we're writing code, pondering some good old memories, trying not to deviate, and letting our hair down; music is everything that can help us relax in a jiffy. Having said that, have you ever thought of creating your own music? As exciting as it sounds, in this project, you shall learn to create music in Python.

Python has a vast collection of packages and libraries that shall help us in building this application with ease. Get ready to experience some pythonic music!

## Our kick-starter

As a rule of thumb, we first have to import the necessary libraries and packages. Getting to know what we're going to import followed by the installation steps is what comprises this section.

Here's the list of packages and their respective 'what-do-they-do':

- **Tkinter** - It is a standard GUI library for Python. Python when combined with Tkinter helps us in building GUI for our music app in no time. It is the one of the most popular and an easy-to-use library that comes with a myriad of widgets. These help us in creating seamless and appealing GUI applications.
- **OS** - The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. We need not install it explicitly. In this project, we are going to use OS for fetching the playlist of songs from the specified directory.
- **Pygame** - It is a Python library that gives us the power of playing with different formats of multimedia like audio, video, etc.


Among the discussed libraries that we're going to use, we've to install pygame (The rest come with Python). Here's the pip command:

```
pip install pygame
```

Make sure to have Python (3.8) on your system beforehand. Here's the link:

#### Download Python

Information about specific ports, and developer info Source and binary executables are signed by the release manager or binary builder using their OpenPGP key.

 <https://www.python.org/downloads/>



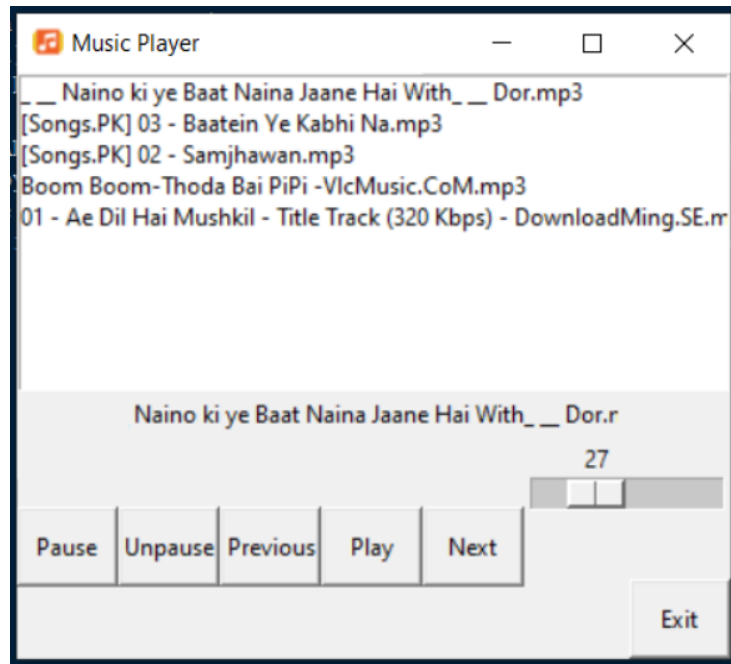
Now that we have downloaded all the necessary libraries, let's get started with our code.

## Let's get started with our code:

A Music Player has to have a display to let the users interact with it. Here, we take a 350×300 grid in which we shall build our Music Player.

Here's a snapshot of the Music Player:

.



We shall build this Music Player in a Pythonic way soon! But before that, let's get the essential libraries into our workspace

## Step 1 - Get the Libraries

Before starting our code, we need a few libraries to help us with calling specific functions.

```
import os
import pygame
from tkinter import *
from tkinter.filedialog import askdirectory
```

There are two main methods used that we need to remember while creating the Python application with GUI.

1. **Tk():** To create the main window, Tkinter offers a method `Tk()`. The basic code used to create the main window of the application is:

```
root = Tk()
```

root is the name of the main window object.

2. **mainloop():** `mainloop()` is used when you are ready for the application to run. `mainloop()` is an infinite loop used to run the application, wait for an event to occur, and process the event till the window is not closed.

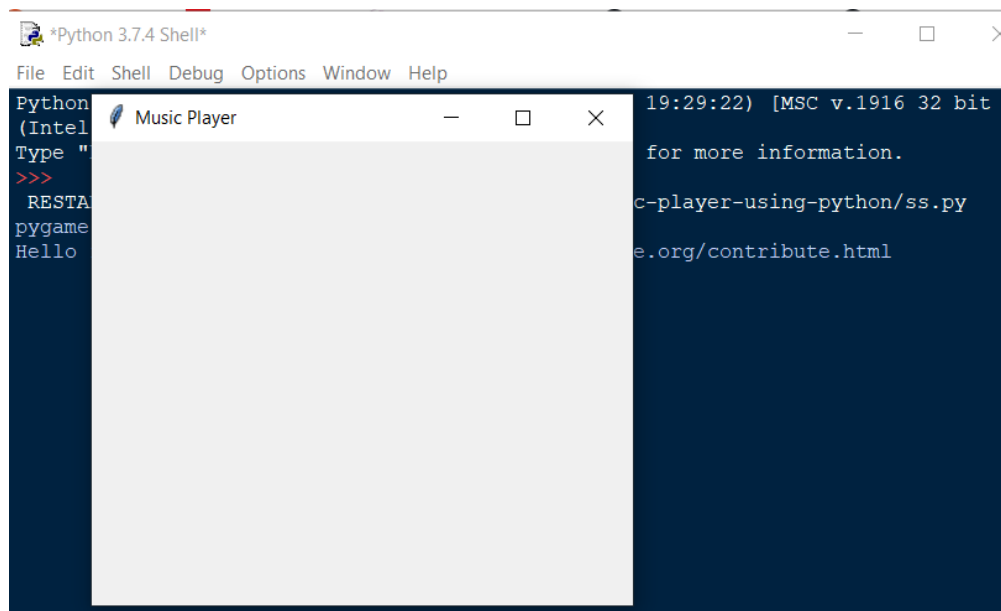
```
root.mainloop()
```

## Step 2: Creating GUI of our Music Player

```
root=Tk()
root.title('Music Player')
root.minsize(350,300)
root.mainloop()
```

- To create the main window, Tkinter offers a method `Tk()`. `root=Tk()` is the basic code use to create the main window of the application.
- `root` is the name of the main window object.
- `root.title()` sets the name of the main window.
- `root.minsize()` sets the size of the main window.
- `mainloop()` is an infinite loop used to run the application

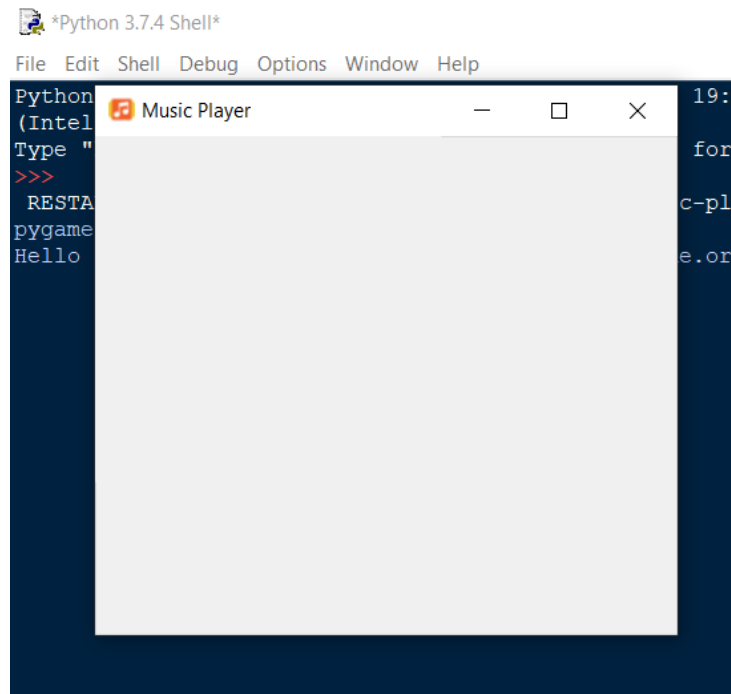
Output:



we can also change the icon of our title window. All we need to do is download an image with '.ico' extension. Make sure to place the input image in the workspace directory, else the absolute path has to be mentioned while loading the image (Any image could be used, as per one's choice).

```
root.iconbitmap('icon.ico')
```

`iconbitmap()` function is used to set the window icon.



## 1. Inserting a list box.

Now that we are done with the basic window. Let's create one list box for our songs. Let's see how we do that!

```
listbox = Listbox(root)
listbox.pack(fill = BOTH)
```

The ListBox widget is used to display different types of items.

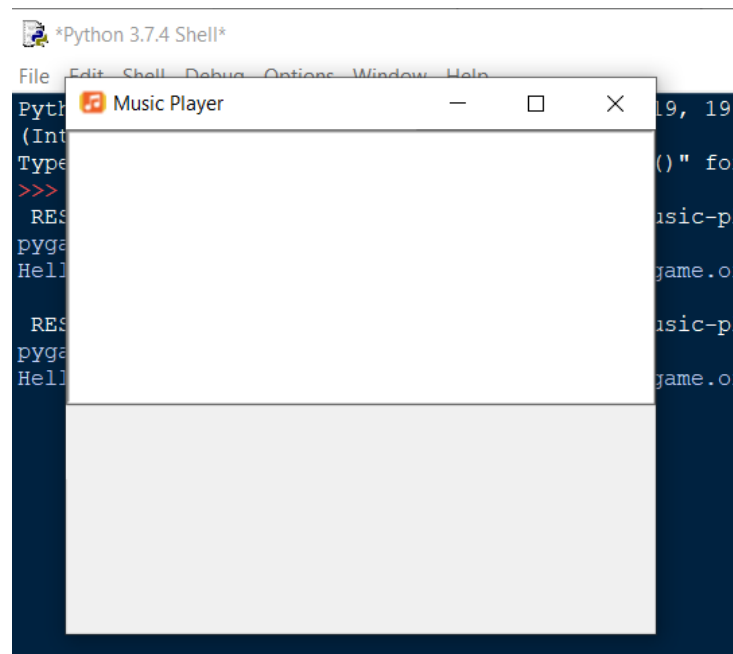
All widgets in the **Tkinter** will have some geometry measurements. These measurements give you to organize the widgets and their parent frames, windows, etc.,

- `pack()` :- It organizes the widgets in the block, which means it occupies the entire available width. It's a standard method to show the widgets in the window.

The **fill** option tells the manager that the widget wants fill the entire space assigned to it. The value controls how to fill the space; `fill = BOTH` means that the widget should expand both horizontally and vertically.

**NOTE: Make sure you define all your widgets and functions before running the `mainloop()`.**

**Output:**



## 2. Getting songs from the directory and inserting them to the Listbox.

BOOM!! We got a Listbox for our songs. Let's fill it with songs that we have in our system. For this, we are going to use the `os` module and `askdirectory()` that we have imported in STEP 1.

Lets see how to do that !!

```
list_songs= []
def choose_directory():
    directory = askdirectory()
    os.chdir(directory)
    for files in os.listdir(directory):
        if files.endswith(".mp3"):
            list_songs.append(files)
    choose_directory()
```

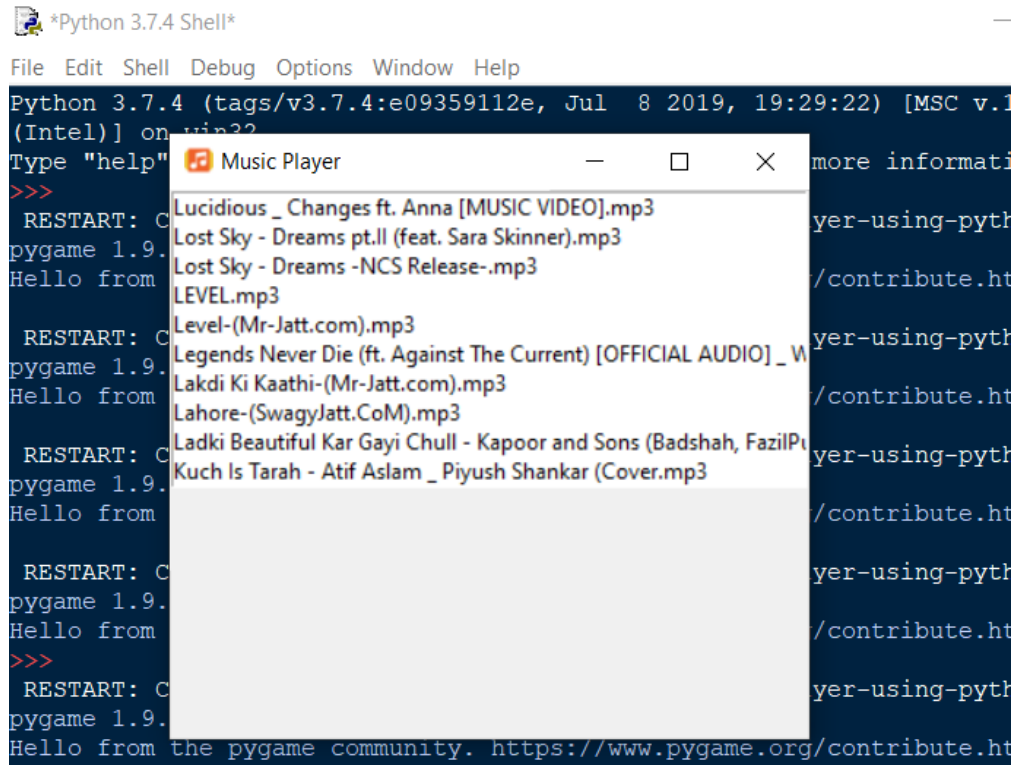
- we are defining an empty list that stores songs.
- In our `choose_directory()` function, we are using `askdirectory()` function to prompt the user to select a directory and storing the path of the selected directory in the directory variable.
- `os.chdir()` function changes the current working directory to the given path.
- Now what we need to do is, we need to append each file in the directory ending with '.mp3' extension to the empty list named 'list\_songs' that we have initialized.
  - Within the `choose_directory()` function, we are using for loop to iterate over the list of entries in the directory. ( We are using `os.listdir()` function to get the list of entries in the directory).
  - Every time we are iterating, we are checking if the file ends with '.mp3' extension then are appending it to the `list_songs`. ( Here, we are using `endswith()` function that returns True if the string ends with the specified suffix, otherwise return False.)
- Now that we have all the music files in the list. we need to insert songs in the Listbox that we created before, For that we shall use for loop outside the choose directory function.

Lets see how to do that!!!

```
for items in list_songs:
    listbox.insert(0,items)
```

After execution we will end up getting this.

Output :



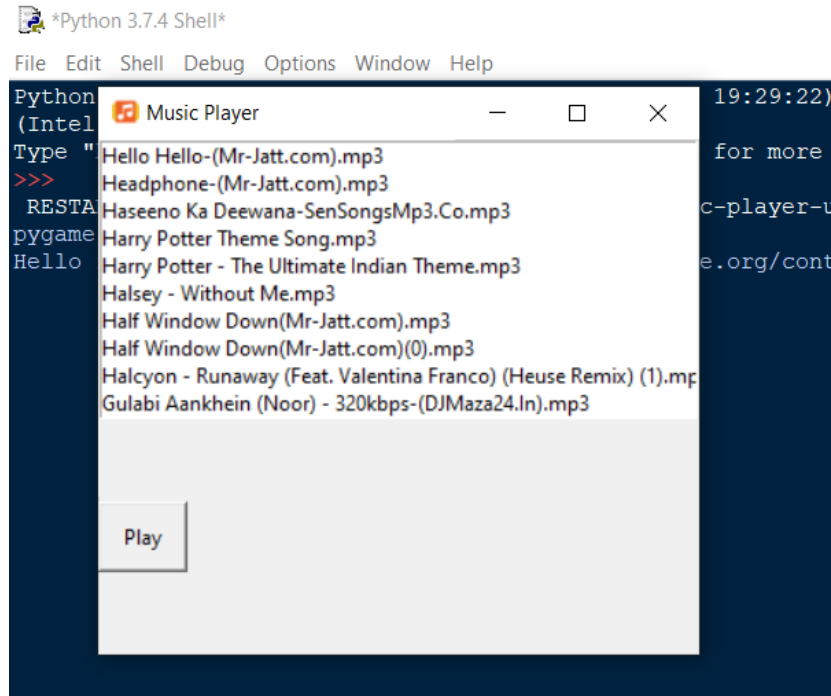
### 3. Adding Play, Pause, Unpause, Previous, Next, and Exit button to the Music Player.

```
playbutton= Button(root,text = 'Play',height = 2, width = 6)
playbutton.pack(side=LEFT)
'''side - Determines which side of the parent widget packs against: TOP (default), BOTTOM,
LEFT, or RIGHT.'''
```

- Button widget is used to add buttons in a Python application. Inside the `Button` function, height determines the height and width determines the width of the button in pixels.
- Side inside the `pack` function determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, or RIGHT.

After execution, we will end up getting this.





Similarly we shall add pause, unpause, previous, next, and exit button to the Music Player.

```
pausebutton = Button(root,text = 'Pause',height = 2, width = 6)
pausebutton.pack(side=LEFT)

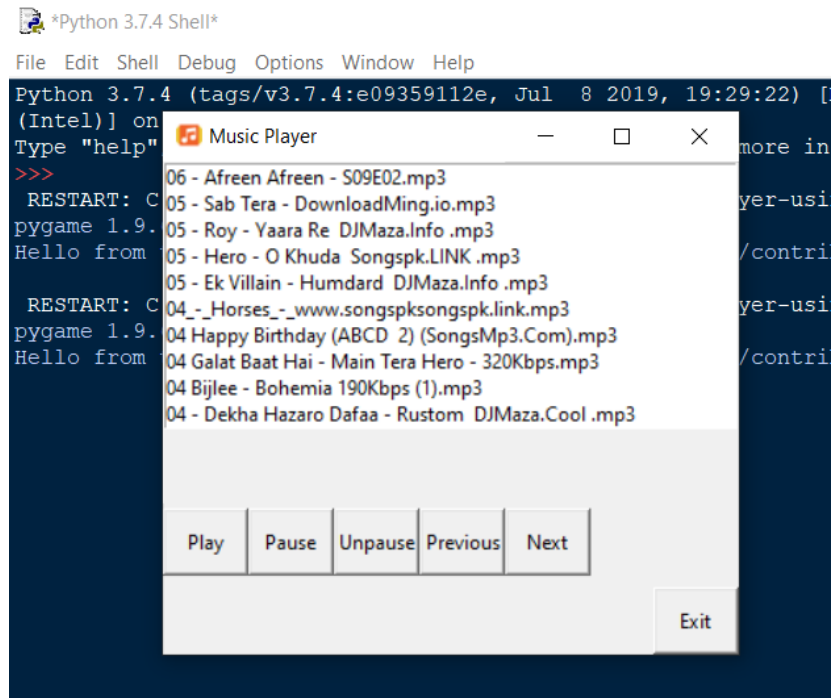
unpausebutton = Button(root,text = 'Unpause',height = 2, width = 6)
unpausebutton.pack(side=LEFT)

prevbutton = Button(root,text = 'Previous',height = 2, width = 6)
prevbutton.pack(side = LEFT)

nextbutton = Button(root,text = 'Next',height = 2, width = 6)
nextbutton.pack(side= LEFT)

exitbutton = Button(root,text = 'Exit',height = 2, width = 6)
exitbutton.pack(anchor = 'e',side = BOTTOM )
```

After execution, we will end up getting this.



#### 4. Functions for play, pause,unpause, previous,next and exit buttons.

Make sure to initialize `pygame.mixer` module before using it's functions. Let's see how we do that!!

```
pygame.mixer.init()
```

To learn more about `pygame.mixer` module go through the link given below

( <https://www.pygame.org/docs/ref/music.html>)

After that, we are done initializing the mixer module. Let's define a new play function for the play button.

```
def play():
    pygame.mixer.music.load(list_songs[0])
    pygame.mixer.music.play()
```

- We are using `pygame.mixer.music.load()` function loads a music file for playback.

- `pygame.mixer.music.play()` start the playback of the music stream.

Similarly, we shall write functions for other buttons.

- `pause()` function

```
def pause():  
    pygame.mixer.music.pause()
```

`pygame.mixer.music.pause()` temporarily stop music playback.

- `unpause()` function.

```
def unpause():  
    pygame.mixer.music.unpause()
```

`pygame.mixer.music.unpause()` will resume the playback of a music stream after it has been paused.

- `nextsong()` function.

```
index = 0  
def nextsong():  
    global index  
    index+=1  
    pygame.mixer.music.load(list_songs[index])  
    pygame.mixer.music.play()
```

- Initialize the index variable to 0 before defining your play function.
- Define a new function named `nextsong()`.
- Index defined previously, is being used as a `global` variable.
- Whenever this function is called, the index will get incremented by 1 and the next song in the `list_songs` is loaded with the help of `pygame.mixer.music.load(list_songs[index])`.

- Then `pygame.mixer.music.play()` will play the loaded music stream.

- `prevsong` function

```
def prevsong():  
    global index  
    index -= 1  
    pygame.mixer.music.load(list_songs[index])  
    pygame.mixer.music.play()
```

- This function works similarly to `nextsong` function. The only difference in this function is, we are decrementing the index value by 1. So, that the previous song in the `list_songs` will get loaded and play.

- `exitbutton()` function

```
def exitbutton():  
    pygame.mixer.music.stop()  
    root.destroy()
```

- `pygame.mixer.music.stop()` function stops the music playback.
- `root.destroy()` is a class Method to Close the Tkinter Window.

## 5. Binding buttons to their respective functions.

A Python function or method can be associated with a button. This function or method will be executed, when the button is pressed.

To bind the play function with the play button, we need to pass one more argument in the Button function. i.e command. command option in Tkinter Button widget is triggered when the user presses the button. Let's see how we do that!!

```
playbutton= Button(root,text = 'Play',height = 2, width = 6, command = play)  
playbutton.pack(side=LEFT)
```

Now when we click on the play button, play function will be executed automatically. This will play the loaded music stream. If the music is already playing it will be restarted.

Similarly, we will bind other buttons also.

```
pausebutton = Button(root,text = 'Pause',height = 2, width = 6,command = pause)
pausebutton.pack(side=LEFT)

unpausebutton = Button(root,text = 'Unpause',height = 2, width = 6,command = unpause)
unpausebutton.pack(side=LEFT)

prevbutton = Button(root,text = 'Previous',height = 2, width = 6,command = prevsong)
prevbutton.pack(side = LEFT)

nextbutton = Button(root,text = 'Next',height = 2, width = 6,command = nextsong)
nextbutton.pack(side= LEFT)

exitbutton = Button(root,text = 'Exit',height = 2, width = 6,command = exitbutton)
exitbutton.pack(anchor = 'e',side = BOTTOM )
```

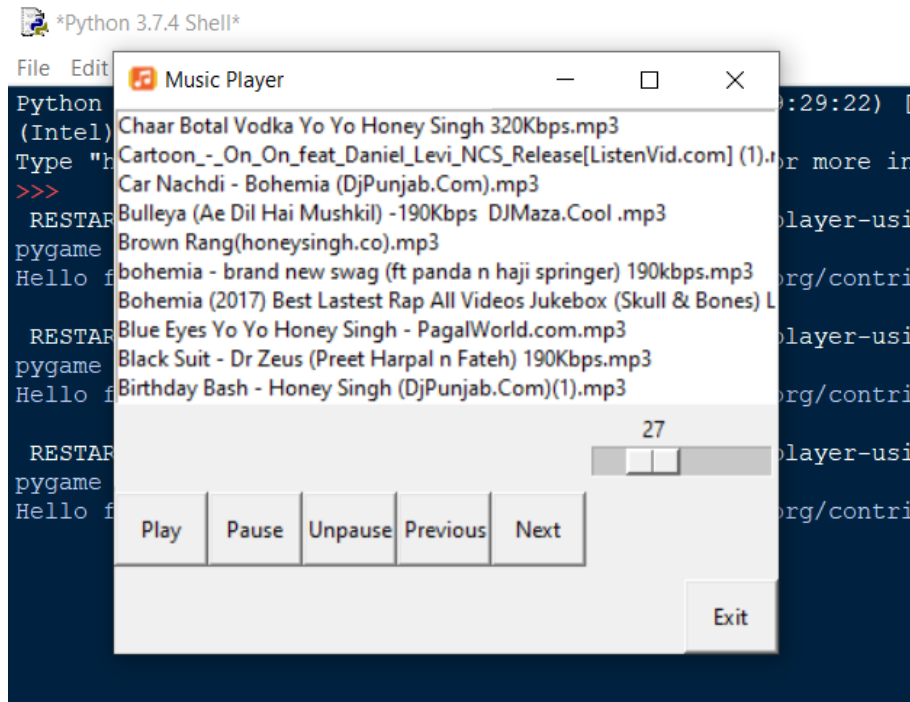
## 6. How to change the volume of music ?!

We will use a widget of Tkinter called as scale to change the volume of music. Let's see how we do that !!

```
scale = Scale(root,from_ = 0, to = 100, orient = HORIZONTAL)
scale.set(27)
scale.pack()
```

- Inside the Scale function, we are setting the range of the scale from 0 to 100 and we are setting its orientation to horizontal (By default it is vertical oriented).
- Then we are using `scale.set()` function to set the default value of scale to 27.

After execution we will end up getting this.



Now that we have the scale but we know that we have to assign a volume function to it.

Let's write the volume function.

```
def volume(val):
    volume = int(val)/100
    pygame.mixer.music.set_volume(volume)
```

we are not manually passing any value to the function. when we change the scale, the scale automatically returns a value in default variable 'val' in string format.

`pygame.mixer.music.set_volume` sets the volume of the music playback. The value argument should be between 0.0 and 1.0 that is why we are dividing the value with 100. When new music is loaded the volume is reset to full volume.

Bind the scale to its respective function.

```
scale = Scale(root, from_ = 0, to = 100, orient = HORIZONTAL, command = volume)
scale.set(27)
scale.pack()
```

## 7. A Song Label

we need to know which song we are playing. we just can't hear and say. So, what we will do right now is we will add a label which shows the current song which is playing. Let's see how we do that!!

```
v = StringVar()
songlabel = Label(root, textvariable = v, width = 35)
songlabel.pack()
```

- Initialize an empty variable that holds a string.
- create a song label using a label widget and pass the empty string variable as an argument. Set the width of the label. You will soon realize why we are passing an empty string variable inside the label.

**NOTE:** Make sure to create the label above the buttons.

Now, let's write the update label function because we need to update the song label every time we click on our buttons.

```
def updateLabel():
    global index
    v.set(list_songs[index])
```

- Index defined previously, is being used as a `global` variable.
- we are passing the name of our song to the string variable that we had created already.

Now what we need to do is, we have to call this `updateLabel()` function from `play`, `nextsong` and `prevsong` function so that the label will get an update every time we play or change our current song.

## Ready to Spin the Music!

After execution, we will end up getting this.

