

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and thank Bro. Mundru balaswamy, csc, principal of NDHCS for providing me an opportunity to do the project work and giving us all support and guidance which made me complete the project duly. I am extremely thankful to [her/him] for providing such a nice support and guidance, although he had busy schedule managing the corporate affairs.

I owe my deep gratitude to our project guide and teacher Mr.K.A.David Paul,MCA, who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system.

I am thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staffs of Computer Science which help us in successfully completing our project work. Also, I would like to extend our sincere esteems to all staff in laboratory for their timely support.

1. OVERVIEW OF PYTHON:

General

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation.

History

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL), capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989.

Python's name is derived from the British comedy group Monty Python, whom Python creator Guido van Rossum enjoyed while developing the language. Monty Python references appear frequently in Python code and culture; for example, the metasyntactic variables often used in Python literature are spam and eggs instead of the traditional foo and bar. The official Python documentation also contains various references to Monty Python routines.

The prefix Py- is used to show that something is related to Python. Examples of the use of this prefix in names of Python applications or libraries include Pygame, a binding of SDL to Python (commonly used to create games); PyQt and PyGTK, which bind Qt and GTK to Python respectively; and PyPy, a Python implementation originally written in Python.

Features

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution. Python's design offers some support for functional programming in the Lisp tradition. It has `filter`, `map`, and `reduce` functions; list comprehensions, dictionaries, sets, and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

Syntax and semantics

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.^[62] Thus, the program's visual structure accurately represents the program's semantic structure.^[1] This feature is sometimes termed the off-side rule, which some other languages share, but in most languages indentation doesn't have any semantic meaning.

Methods

Methods on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter to access instance data, in contrast to the implicit `self` (or `this`) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).

Typing

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, `SpamClass()` or `EggsClass()`), and the classes are instances of the metaclass `type` (itself an instance of itself), allowing metaprogramming and reflection.

Expressions

Python makes a distinction between lists and tuples. Lists are written as `[1, 2, 3]`, are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as `(1, 2, 3)`, are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The `+` operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable `t` initially equal to

(1, 2, 3), executing `t = t + (4, 5)` first evaluates `t + (4, 5)`, which yields (1, 2, 3, 4, 5), which is then assigned back to `t`, thereby effectively "modifying the contents" of `t`, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.

Python features sequence unpacking wherein multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the produced values to the corresponding expression on the left.

Python has array index and array slicing expressions on lists, denoted as `a[key]`, `a[start:stop]` or `a[start:stop:step]`. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example `a[:]` returns a copy of the entire list. Each element of a slice is a shallow copy.

Mathematics

Python has the usual symbols for arithmetic operators (`+`, `-`, `*`, `/`), and the remainder operator `%` (where the remainder can be negative, e.g. `4%-3 == -2`). It also has `**` for exponentiation, e.g. `5**3 == 125` and `9**0.5 == 3.0`, and a new matrix multiply `@` operator is included in version 3.5.^[86] I.e. all these operators work as in traditional math; with same precedence rules, the operators infix (or `-` can additionally be unary). Additionally, it

has a unary operator (`~`), which essentially inverts all the bits of its one argument. For integers, this means `~x=-x-1`. Other operators include bitwise shift operators `x << y`, which shifts `x` to the left `y` places, the same as `x*(2**y)`, and `x >> y`, which shifts `x` to the right `y` places, the same as `x/(2**y)`.

Python provides a `round` function for rounding a float to the nearest integer. For tie-breaking, versions before 3 use round-away-from-zero: `round(0.5)` is 1.0, `round(-0.5)` is -1.0. Python 3 uses round to even: `round(1.5)` is 2, `round(2.5)` is 2.

Python has extensive built-in support for arbitrary-precision arithmetic. Integers are transparently switched from the machine-supported maximum fixed-precision (usually 32 or 64 bits), belonging to the python type `int`, to arbitrary precision, belonging to the Python type `long`, where needed. The latter have an "L" suffix in their textual representation. (In Python 3, the distinction between the `int` and `long` types was eliminated; this behavior is now entirely contained by the `int` class.) The `Decimal` type/class in module `decimal` (since version 2.4) provides decimal floating point numbers to arbitrary precision and several rounding modes. The `Fraction` type in module `fractions` (since version 2.6) provides arbitrary precision for rational numbers.

Libraries

Due to Python's extensive mathematics library, and the third-party library NumPy that further extends the native capabilities, it is frequently used as a scientific scripting language to aid in problems such as numerical data processing and manipulation.

Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user

interface, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary-precision decimals, manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation `wsgiref` follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

Development environments

Most Python implementations (including CPython) include a read–eval–print loop (REPL), permitting them to function as a command line interpreter for which the user enters statements sequentially and receives results immediately.

Other shells, including IDLE and IPython, add further abilities such as auto-completion, session state retention and syntax highlighting.

As well as standard desktop integrated development environments, there are Web browser-based IDEs; SageMath (intended for developing science and math-related Python programs); PythonAnywhere, a browser-based IDE and hosting environment; and Canopy IDE, a commercial Python IDE emphasizing scientific computing.

Uses

- An empirical study found that scripting languages, such as Python, are more productive than conventional languages, such as C and Java, for programming problems involving string manipulation and search in a dictionary, and determined that memory consumption was often "better than Java and C or C++".
- Large organizations that use Python include Wikipedia, Google, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram,

Spotify and some smaller entities like ILM and ITA. The social news networking site Reddit is written entirely in Python.

- Python can serve as a scripting language for web applications, e.g., via `mod_wsgi` for the Apache web server.
- Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing, with specialized libraries such as Biopython and Astropy providing domain-specific functionality. SageMath is a mathematical software with a notebook interface programmable in Python: its library covers many aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus.
- Python has been successfully embedded in many software products as a scripting language, including in finite element method software such as Abaqus, 3D parametric modeler like FreeCAD, 3D animation packages such as 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, the visual effects compositor Nuke, 2D imaging programs like GIMP, Inkscape, Scribus and Paint Shop Pro, and musical notation programs like scorewriter and capella. GNU Debugger uses Python as a pretty printer to show complex structures such as C++ containers. Esri promotes Python as the best choice for writing scripts in ArcGIS. It has also been used in several video games, and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go.
- Python is commonly used in artificial intelligence projects with the help of libraries like TensorFlow, Keras and Scikit-learn. As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.

2.PROJECT OVERVIEW:

2.1 INTRODUCTION:

Digital Music Player is a hardware or software that plays audio files encoded in MP3 and other audio formats. On the software side, applications that reside in the user's computer, such as iTunes, Windows Media Player and RealPlayer, are used to organize a music collection, play audio files and rip music from a CD. Software players may also provide access to Internet radio stations and other streaming audio sites.

2.2 OBJECTIVE:

- To build a python project to construct PICTO Music Player.
- Allow the users to sort the songs based on the order in which they want to listen to them.
- User changes the track or play different track without noticing her/his state of mood.
- A fast music player with stylish design.
- Play songs from folders.
- Music player supports almost all type of music files like mp3, wav, ac raw and many others.

2.3 EXISTING SYSTEM:

The current system of the referenced project lacks going back to the previous audio/song, selecting audio from different locations, have simple interface, no much detail about its build and author, etc.

2.4 PROPOSED SYSTEM:

This PICTO Music Player helps the users by giving them the facility to add as many songs in the playlist, adding or deleting songs and option to get back to previous song. This has developed GUI interface to pause, play, stop, rewind, mute, adjust volume, etc. This interface also displays the name of current song playing, total song length, current running time, any action done, help window, etc.

3. SYSTEM REQUIREMENTS:

3.1 SOFTWARE REQUIREMENTS:

- Windows 7/8/10 or later 64-bit operating system.
- Python IDLE 3.x version
- Anaconda navigator 1.x version

3.2 HARDWARE REQUIREMENTS:

- Processors: Intel Atom® processor or Intel® Core™ i3 processor
- 4 GB RAM (Minimum)
- 20 GB Hard Disk Drive
- 512 MB ROM (Recommended)
- Sound card (any)
- Speakers
- 3.5mm jack

4. PACKAGES AND MODULES:

- pygame
- mutagen.mp3
- os
- threading
- time
- tkinter.ttk
- tkinter.messagebox
- tkinter.filedialog
- tkinter.themed_tk
- ttkthemes

5. CONCLUSION:

- The music player is capable of playing wav file and output to a 3.5mm jack which can be connected to any standard speaker like earphone to woofer system.
- Sound produced is well considered 8bit to 8kHz wave file.
- The core part of the music player is mainly composed of main interface, playlists, play Settings, file browsing and song search.
- Music player system realized the basic function of player: play, pause and stop, up/down, volume adjustment, play mode, song search, file browser, playlists query and other functions.
- Hence, PICTO Music Player, the project on python is completed successfully.
- This will help the user to work more efficiently.

6.SOURCE CODE

```
import os

import threading

import time

import tkinter.messagebox

from tkinter import *

from tkinter import filedialog


from tkinter import ttk

from ttkthemes import themed_tk as tk


from mutagen.mp3 import MP3

from pygame import mixer


"""Pygame is a cross-platform set of

Python modules designed for writing video games.

It includes computer graphics and sound libraries designed

to be used with the Python programming language."""


"""Mutagen is a Python module to handle audio metadata.

It supports ASF, FLAC, MP4, Monkey's Audio, MP3, Musepack,

Ogg Opus, Ogg FLAC, Ogg Speex, Ogg Theora,Ogg Vorbis, True Audio,

WavPack, OptimFROG, and AIFF audio files."""
```

"""The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads. Threading in python is used to run multiple threads (tasks, function calls) at the same time."""

"""The OS module in Python provides a way of using operating system dependent functionality. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux."""

"""ttkthemes includes a wide variety of different themes, and there is always room for more themes, no matter how ugly or obscure! Even though some themes may not be used in practice, the original goal of the project has not been forgotten: To gather and preserve all themes.
"""

"""Tkinter is the standard GUI library for Python.
Python when combined with Tkinter provides a fast and easy way to create GUI applications.
Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit."""

"""The Python time module provides many ways of representing time in code, such as objects, numbers, and strings. It also provides functionality other than representing time, like waiting during code execution and measuring the efficiency of your code."""

```
root = tk.ThemedTk()
```

```
root.get_themes()          # Returns a list of all themes that can be set
```

```
root.set_theme("radiance")  # Sets an available theme
```

```
# Fonts - Arial (corresponds to Helvetica), Courier New (Courier), Comic Sans MS, Fixedsys,
```

```
# MS Sans Serif, MS Serif, Symbol, System, Times New Roman (Times), and Verdana
```

```
#
```

```
# Styles - normal, bold, roman, italic, underline, and overstrike.
```

```
statusbar = tk.Label(root, text="Welcome to PICTO Music Player",
```

```
                    relief=SUNKEN, anchor=W,
```

```
                    font='Times 10 italic')
```

```
statusbar.pack(side=BOTTOM, fill=X)
```

```
# Create the menubar
```

```
menubar = Menu(root)
```

```
root.config(menu=menubar)
```

```
# Create the submenu
```

```
subMenu = Menu(menubar, tearoff=0)
```

```
playlist = []
```

```
# playlist - contains the full path + filename
```

```
# playlistbox - contains just the filename
```

```
# Fullpath + filename is required to play the music inside play_music load function
```

```
def browse_file():
```

```
    global filename_path
```

```
    filename_path = filedialog.askopenfilename()
```

```
    add_to_playlist(filename_path)
```

```
    mixer.music.queue(filename_path)
```

```
def add_to_playlist(filename):
```

```
    filename = os.path.basename(filename)
```

```
    index = 0
```

```
    playlistbox.insert(index, filename)
```

```
    playlist.insert(index, filename_path)
```

```
index += 1
```

```
menubar.add_cascade(label="File", menu=subMenu)
```

```
subMenu.add_command(label="Open", command=browse_file)
```

```
subMenu.add_command(label="Exit", command=root.destroy)
```

```
def about_us():
```

```
    tkinter.messagebox.showinfo('About PICTO Music Player',
```

```
                                "This is a music player developed using Python Tkinter and  
                                mutagen.mp3 module built by
```

```
                                \n\t      Sudarsun.S")
```

```
subMenu = Menu(menubar, tearoff=0)
```

```
menubar.add_cascade(label="Help", menu=subMenu)
```

```
subMenu.add_command(label="About Player", command=about_us)
```

```
mixer.init() # initializing the mixer
```

```
root.title("PICTO Music Player")
```

```
root.iconbitmap(r'images/picto.ico')
```

```
# Root Window - StatusBar, LeftFrame, RightFrame
```

```
# LeftFrame - The listbox (playlist)
```

```
# RightFrame - TopFrame,MiddleFrame and the BottomFrame
```



```
leftframe = Frame(root)

leftframe.pack(side=LEFT, padx=30, pady=30)


playlistbox = Listbox(leftframe)

playlistbox.pack()


addBtn = ttk.Button(leftframe, text="+ Add",
                    command=browse_file)

addBtn.pack(side=LEFT)


def del_song():

    selected_song = playlistbox.curselection()

    selected_song = int(selected_song[0])

    playlistbox.delete(selected_song)

    playlist.pop(selected_song)


delBtn = ttk.Button(leftframe, text="- Del",
                    command=del_song)

delBtn.pack(side=LEFT)


rightframe = Frame(root)

rightframe.pack(pady=30)
```

```
topframe = Frame(rightframe)
```

```
topframe.pack()
```

```
lengthlabel = ttk.Label(topframe, text='Total Length : --:--')
```

```
lengthlabel.pack(pady=5)
```

```
currenttimelabel = ttk.Label(topframe, text='Current Time : --:--',  
                             relief=GROOVE)
```

```
currenttimelabel.pack()
```

```
def show_details(play_song):
```

```
    file_data = os.path.splitext(play_song)
```

```
    if file_data[1] == '.mp3':
```

```
        audio = MP3(play_song)
```

```
        total_length = audio.info.length
```

```
    else:
```

```
        a = mixer.Sound(play_song)
```

```
        total_length = a.get_length()
```

```
    # div - total_length/60, mod - total_length % 60
```

```
    mins, secs = divmod(total_length, 60)
```

```
    mins = round(mins)
```

```
secs = round(secs)

timeformat = '{:02d}:{:02d}'.format(mins, secs)

lengthlabel['text'] = "Total Length" + ' - ' + timeformat
```

```
t1 = threading.Thread(target=start_count, args=(total_length,))

t1.start()
```

```
def start_count(t):

    global paused

    # mixer.music.get_busy(): - Returns FALSE when we press the stop button (music stop
    playing)

    # Continue - Ignores all of the statements below it. We check if music is paused or not.

    current_time = 0

    while current_time <= t and mixer.music.get_busy():

        if paused:

            continue

        else:

            mins, secs = divmod(current_time, 60)

            mins = round(mins)

            secs = round(secs)

            timeformat = '{:02d}:{:02d}'.format(mins, secs)

            currenttimelabel['text'] = "Current Time" + ' - ' + timeformat

            time.sleep(1)

            current_time += 1
```

```
def play_music():
    global paused
    if paused:
        mixer.music.unpause()
        statusbar['text'] = "Music Resumed"
        paused = False
    else:
        try:
            stop_music()
            time.sleep(1)
            selected_song = playlistbox.curselection()
            selected_song = int(selected_song[0])
            play_it = playlist[selected_song]
            mixer.music.load(play_it)
            mixer.music.play()
            statusbar['text'] = "Playing music" + ' - ' + os.path.basename(play_it)
            show_details(play_it)
        except:
            tkinter.messagebox.showerror('File not found',
                                         'Player could not find the file. Please check again.')

def stop_music():
    mixer.music.stop()
    statusbar['text'] = "Music Stopped"
```

```
paused = FALSE
```

```
def pause_music():
```

```
    global paused
```

```
    paused = TRUE
```

```
    mixer.music.pause()
```

```
    statusbar['text'] = "Music Paused"
```

```
def rewind_music():
```

```
    play_music()
```

```
    statusbar['text'] = "Previous Music"
```

```
def set_vol(val):
```

```
    volume = float(val) / 100
```

```
    mixer.music.set_volume(volume)
```

```
# set_volume of mixer takes value only from 0 to 1. Example - 0, 0.1,0.55,0.54,0.99,1
```

```
muted = FALSE
```

```
def mute_music():
```

```
    global muted
```

```
    if muted: # Unmute the music
```

```
        mixer.music.set_volume(0.7)
```

```
        volumeBtn.configure(image=volumePhoto)
```

```
        scale.set(70)
```

```
        muted = FALSE
```

```
else: # mute the music
```

```
    mixer.music.set_volume(0)
```

```
    volumeBtn.configure(image=mutePhoto)
```

```
    scale.set(0)
```

```
    muted = TRUE
```

```
middleframe = Frame(rightframe)
```

```
middleframe.pack(pady=30, padx=30)
```

```
playPhoto = PhotoImage(file='images/pla.png')
```

```
playBtn = ttk.Button(middleframe, image=playPhoto,  
                     command=play_music)
```

```
playBtn.grid(row=0, column=0, padx=10)
```

```
stopPhoto = PhotoImage(file='images/st.png')
```

```
stopBtn = ttk.Button(middleframe, image=stopPhoto,  
                    command=stop_music)
```

```
stopBtn.grid(row=0, column=1, padx=10)
```

```
pausePhoto = PhotoImage(file='images/pau.png')
```

```
pauseBtn = ttk.Button(middleframe, image=pausePhoto,  
                     command=pause_music)
```

```
pauseBtn.grid(row=0, column=2, padx=10)
```

```
# Bottom Frame for volume, rewind, mute etc.
```

```
bottomframe = Frame(rightframe)
```

```
bottomframe.pack()

rewindPhoto = PhotoImage(file='images/rewind.png')

rewindBtn = ttk.Button(bottomframe, image=rewindPhoto,
                        command=rewind_music)

rewindBtn.grid(row=0, column=0)

mutePhoto = PhotoImage(file='images/mut.png')

volumePhoto = PhotoImage(file='images/vol.png')

volumeBtn = ttk.Button(bottomframe, image=volumePhoto,
                       command=mute_music)

volumeBtn.grid(row=0, column=1)

scale = ttk.Scale(bottomframe, from_=0, to=100,
                  orient=HORIZONTAL, command=set_vol)

scale.set(70) # implement the default value of scale when music player starts

mixer.music.set_volume(0.7)

scale.grid(row=0, column=2, pady=15, padx=30)

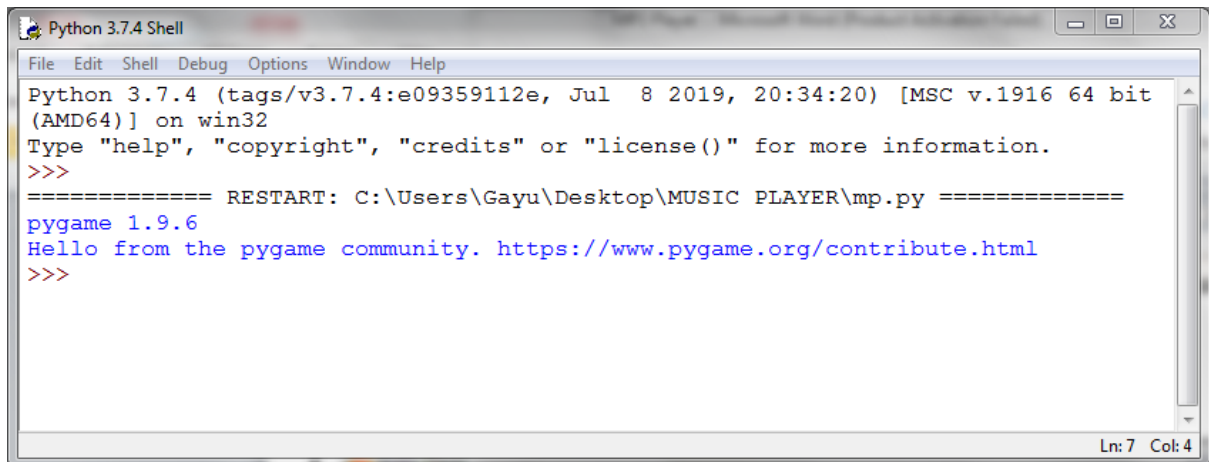

def on_closing():
    stop_music()
    root.destroy()


root.protocol("WM_DELETE_WINDOW", on_closing)

root.mainloop()
```

7.OUTPUT

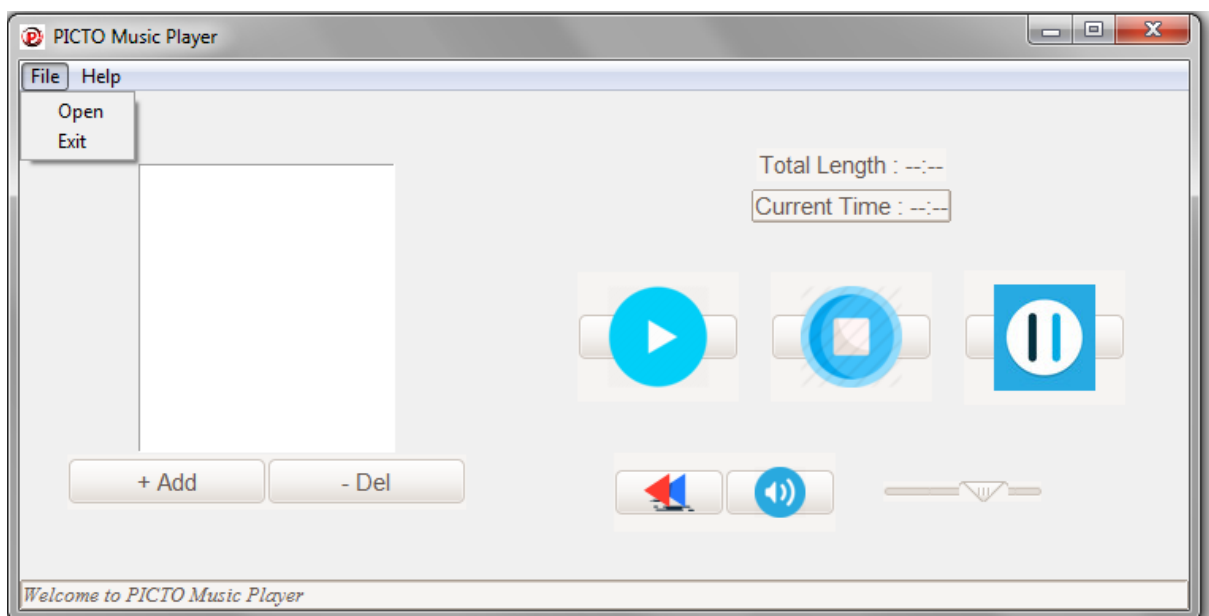
INITIAL OUTPUT OF PYTHON SHELL



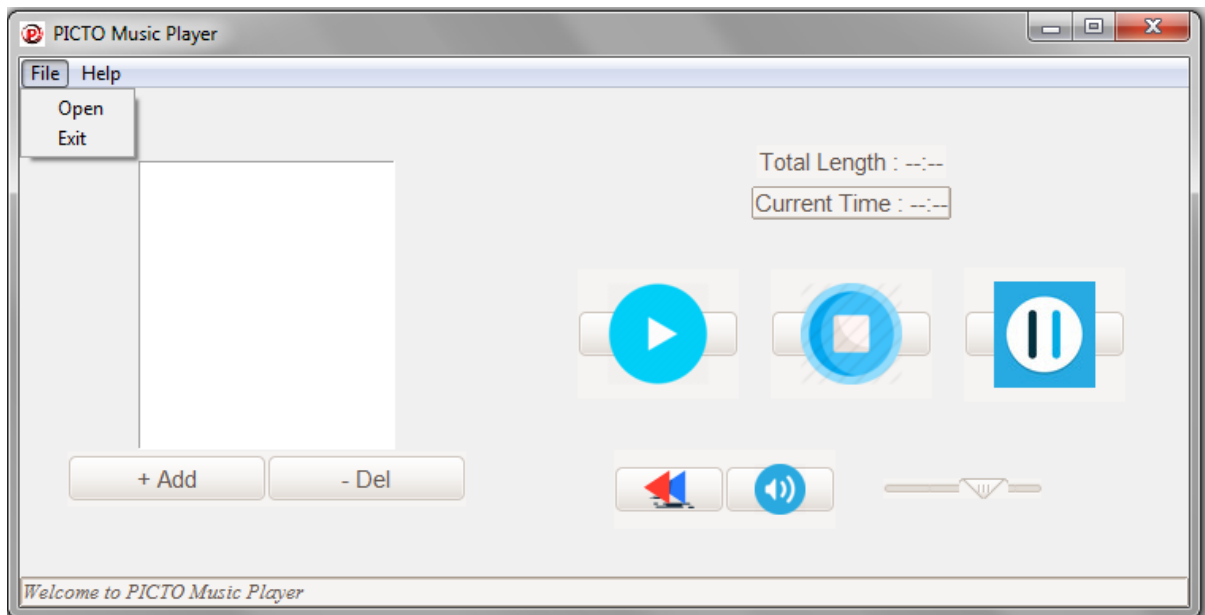
```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Gayu\Desktop\MUSIC PLAYER\mp.py =====
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
>>>
```

Ln: 7 Col: 4

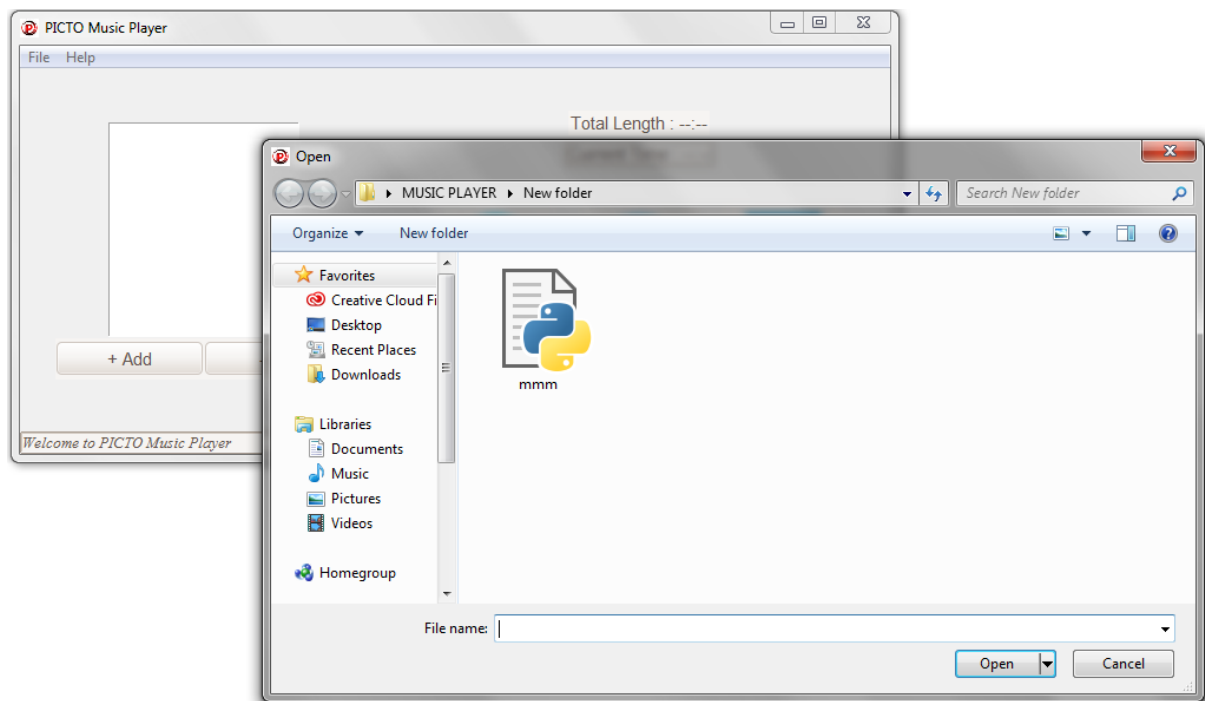
FIRST LOOK OF MUSIC PLAYER



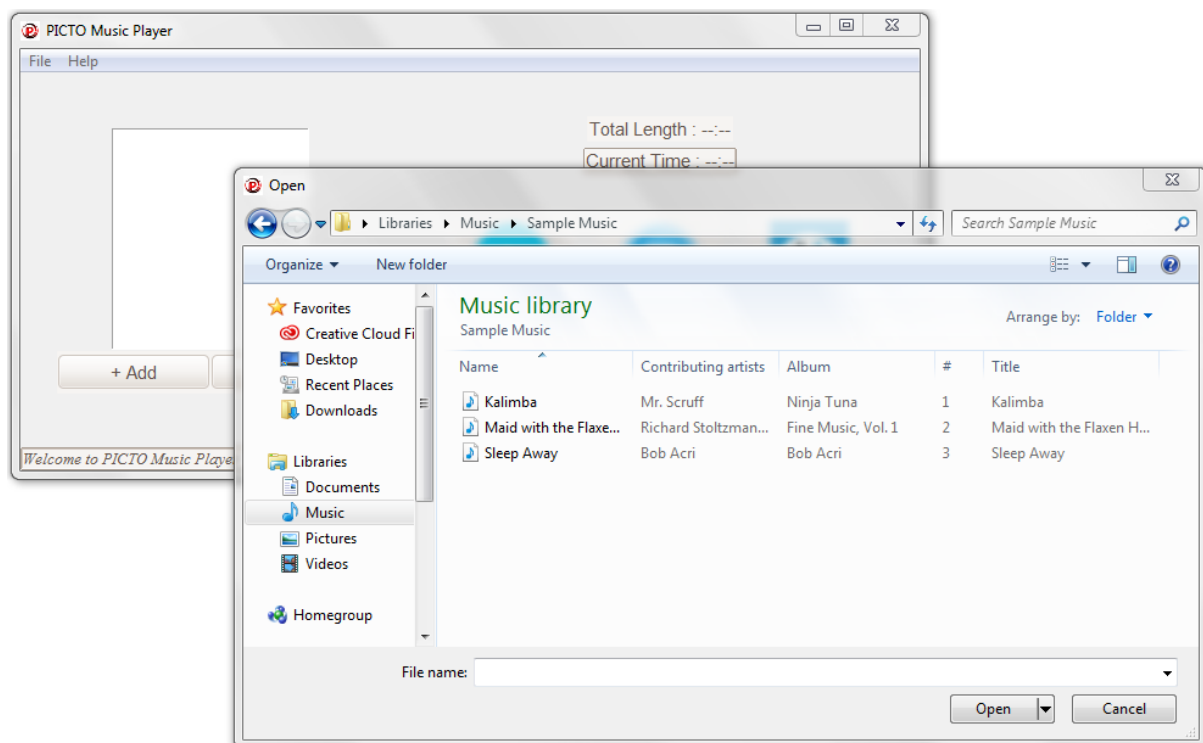
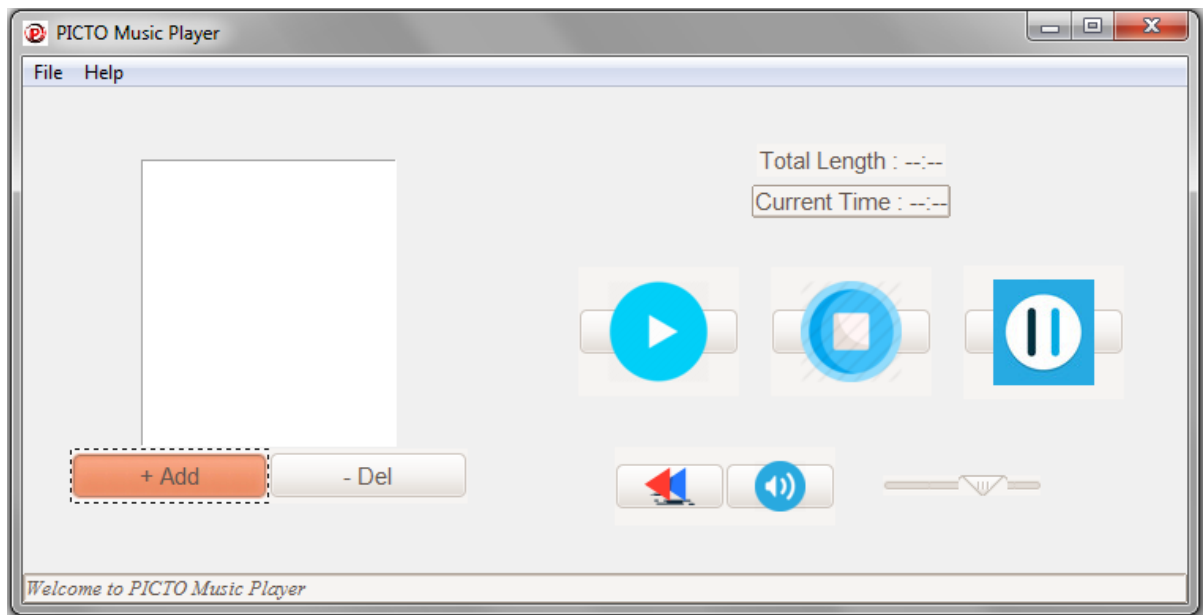
FILE MENU



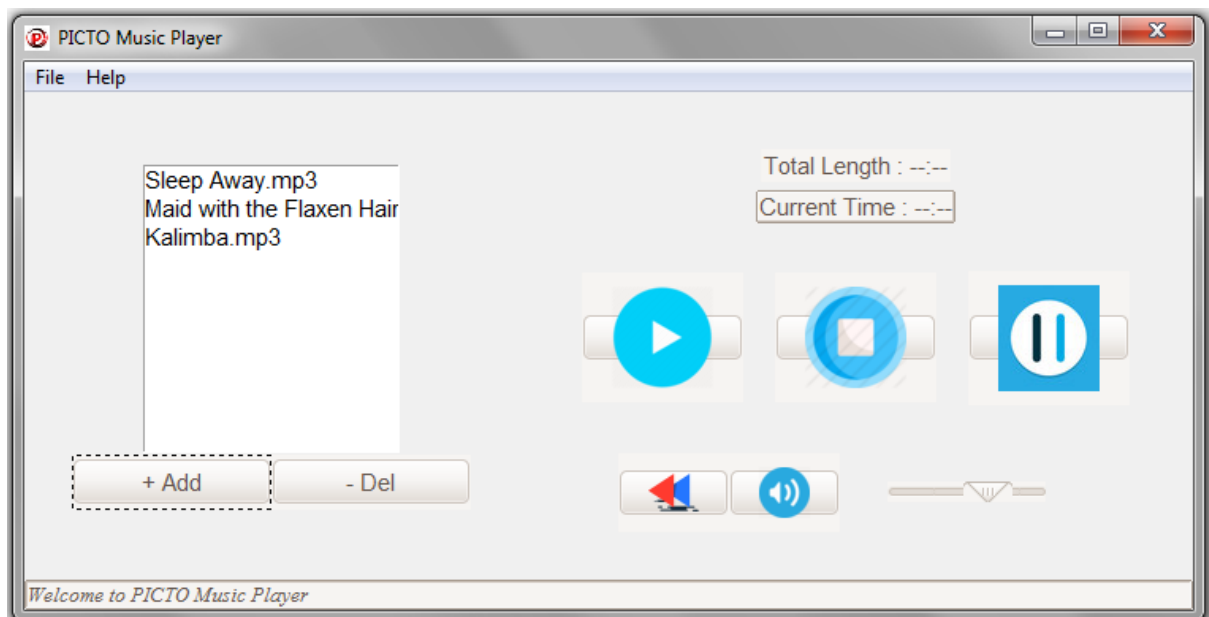
OPENING FOLDER FOR MUSIC



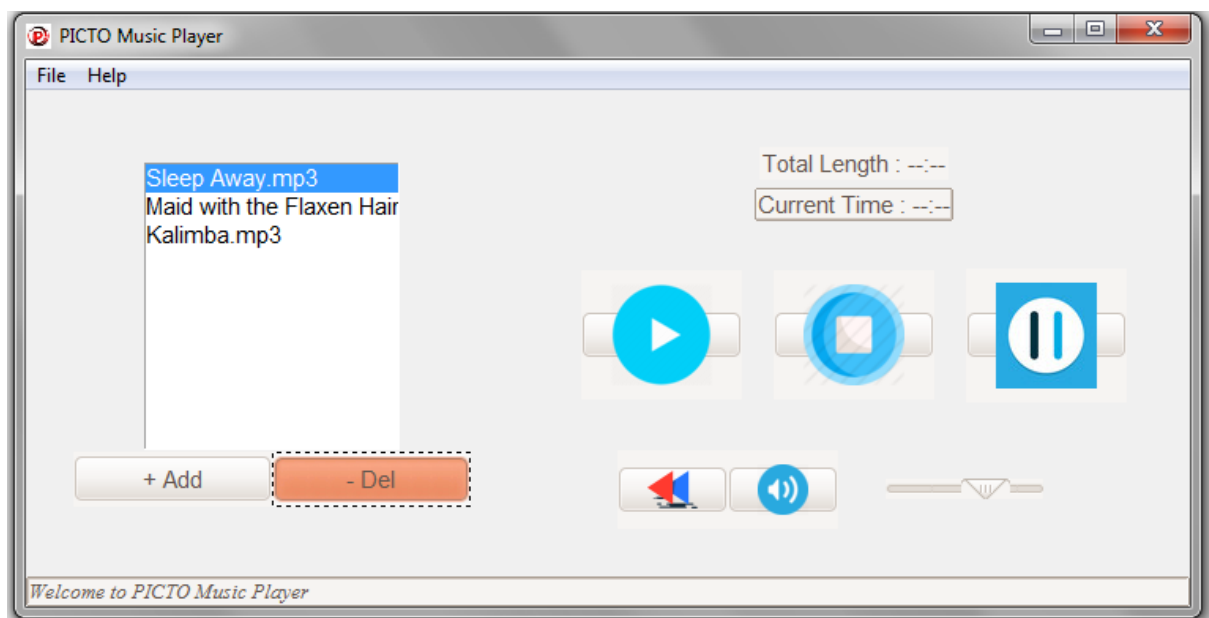
ADDING NEW SONG IN PLAYER



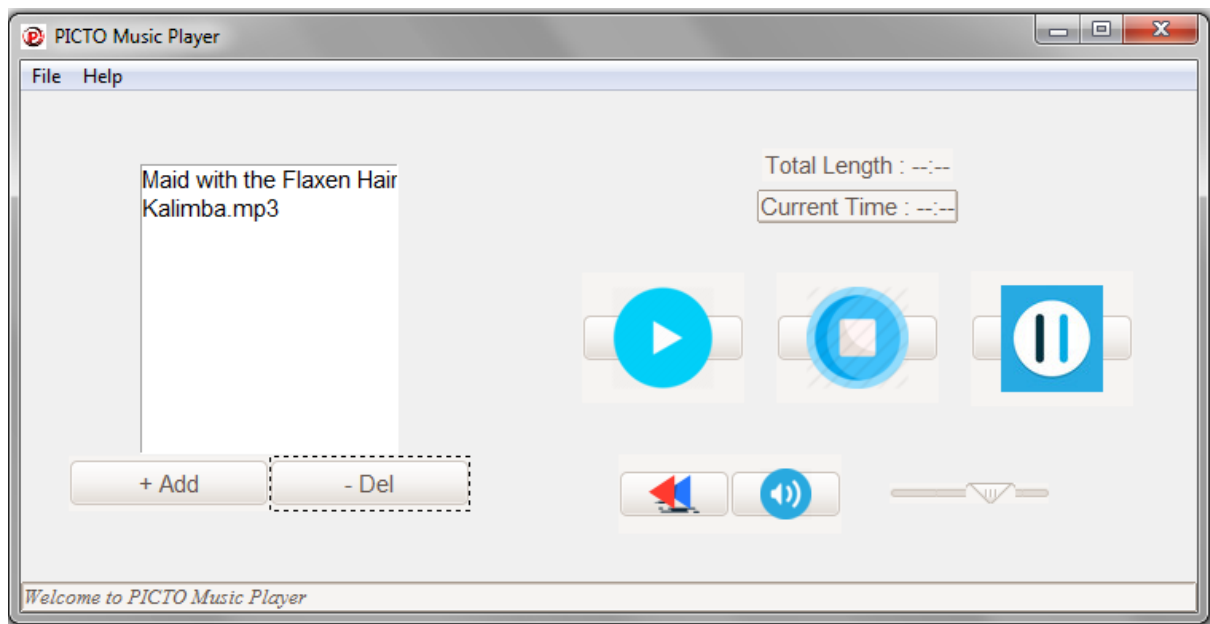
AFTER ADDING SONGS IN PLAYER



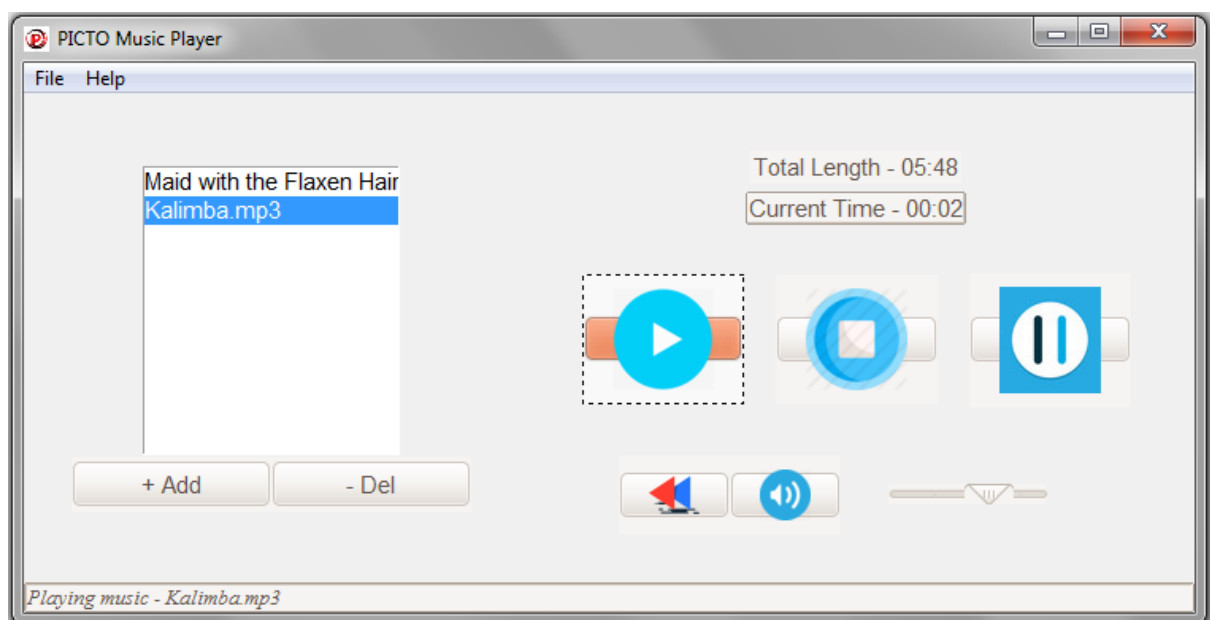
DELETING A SONG FROM PLAYER



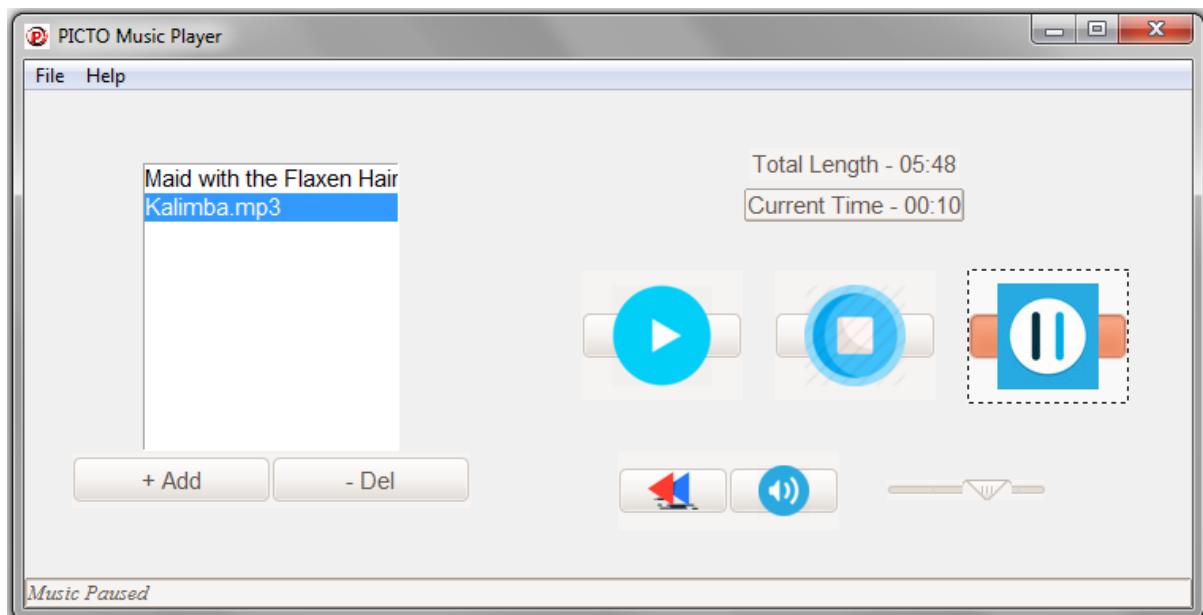
AFTER DELETING A SONG



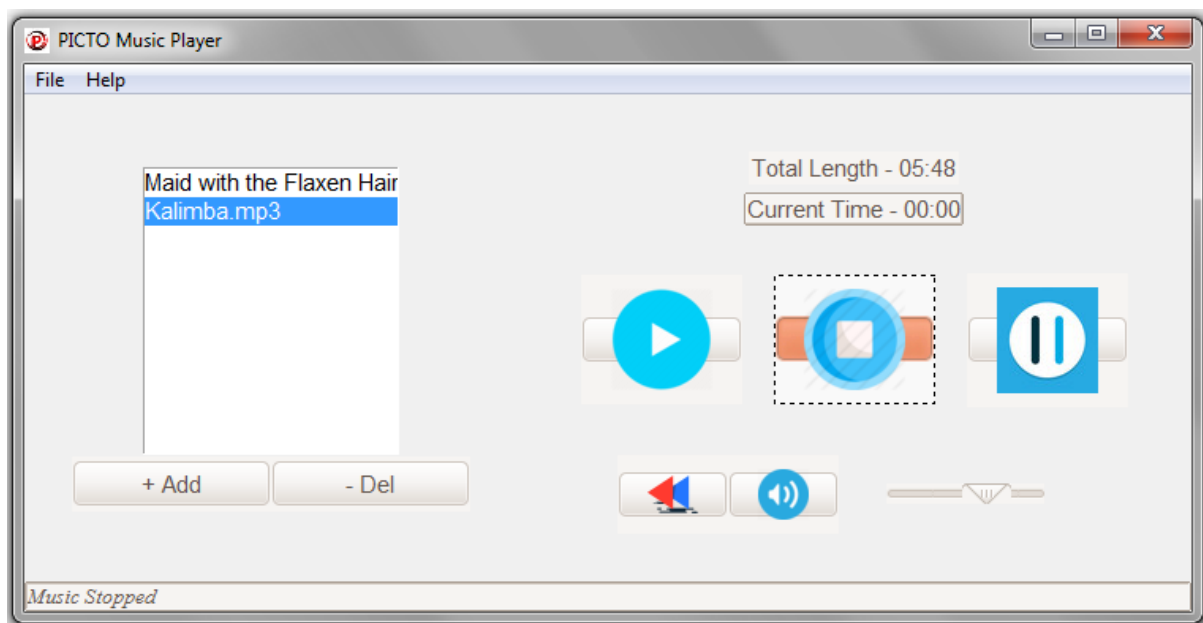
PLAYING A SONG



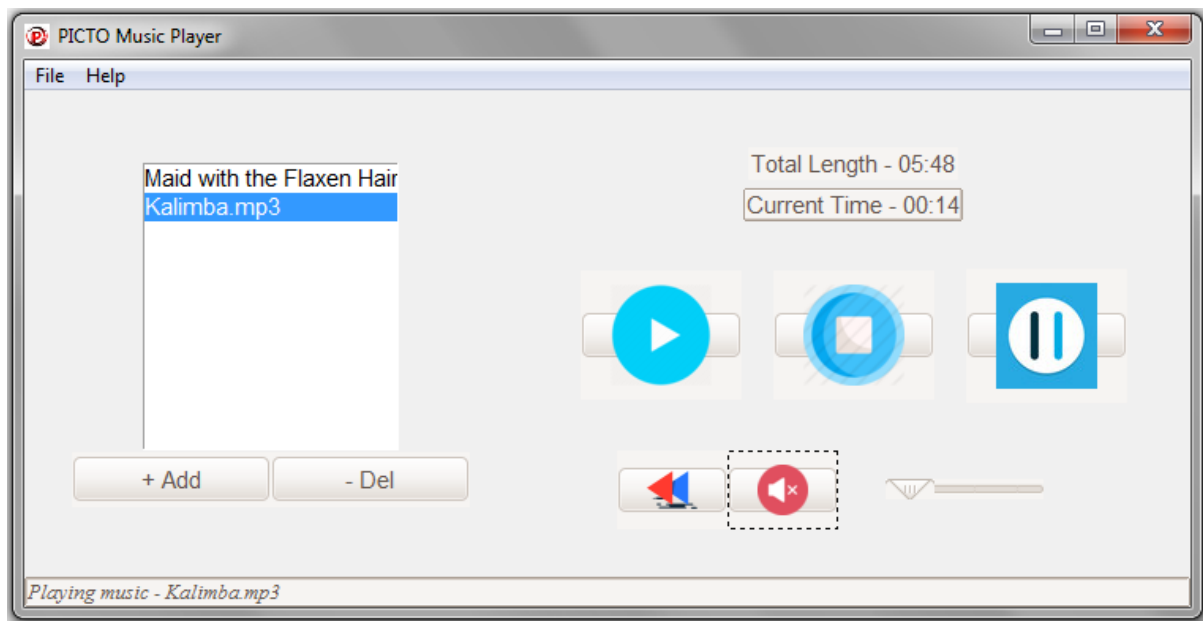
PAUSING A SONG



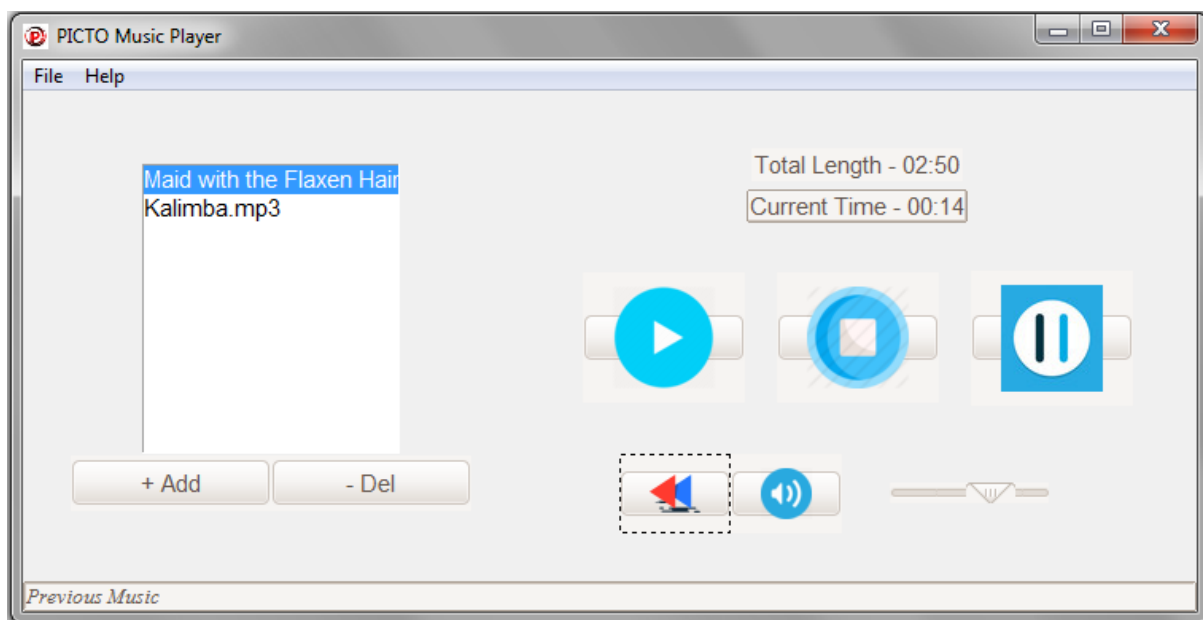
STOPPING THE CURRENT SONG



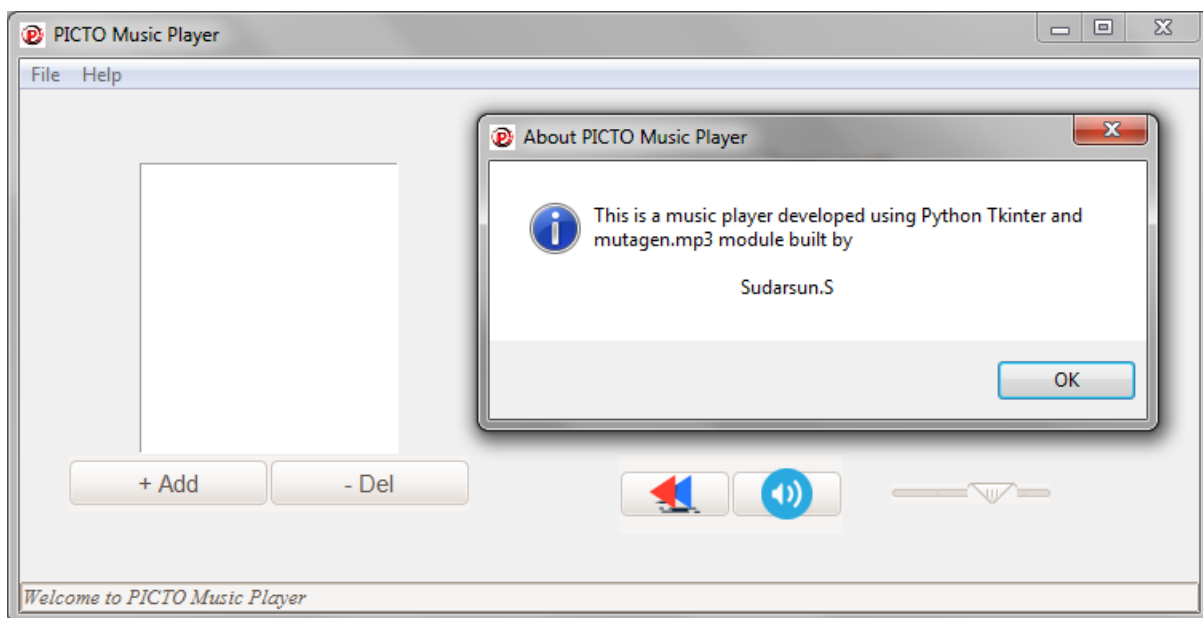
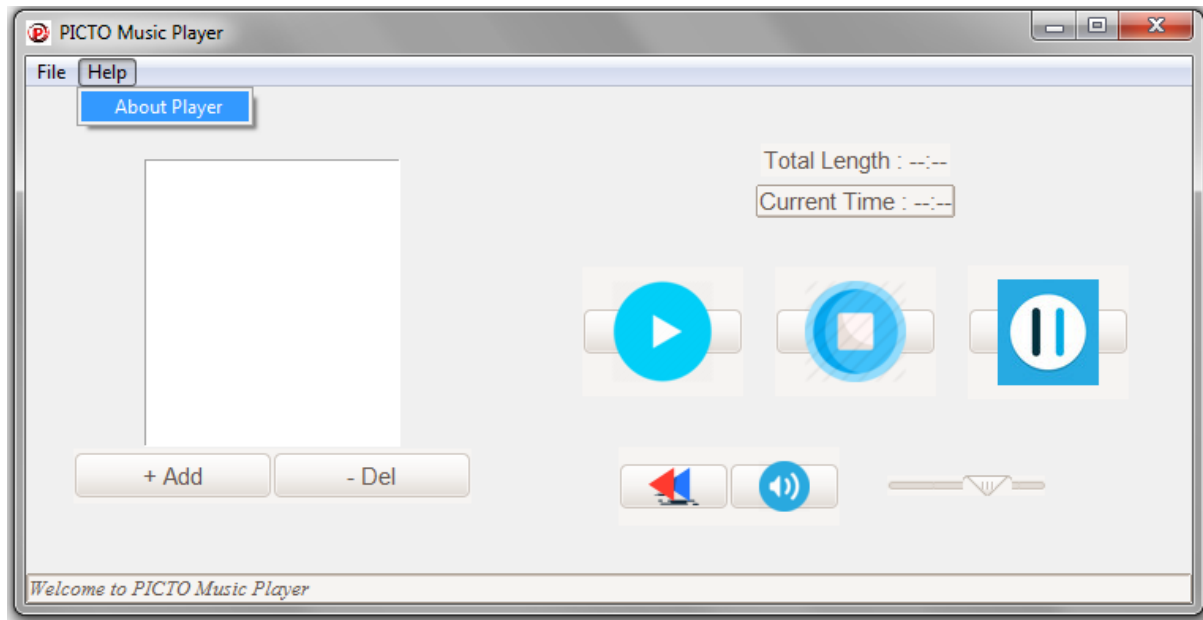
CURRENT SONG IS MUTED



PLAYING PREVIOUS SONG



OPENING HELP WINDOW



8.BIBLIOGRAPHY:

- ✧ www.wikipedia.com
- ✧ www.learnpython.org
- ✧ www.google'spythonclassbook.org
- ✧ www.pythoninstitute.org
- ✧ www.codeacademy.com
- ✧ www.codeclub.in
- ✧ Python for Class 11 by Sumita Arora
- ✧ Python for Class 12 by Sumita Arora