Note from the Authors: This article is an excerpt from Chapter 24, Sections 24.6 and 24.7 of *Python How to Program, 1/e*. This article introduces **`pygame`**, a multimedia package for Python. We use **`pygame`** and **`Tkinter`** to create a working CD-Player with a graphical interface. The simplicity of the program demonstrates the power and expressiveness of Python and its many open-source packages. Readers should be familiar with object-oriented programming and this article is intended for advanced programmers. The code examples included in this article show readers examples using the Deitel™ signature *LIVE-CODE™ Approach*, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

24.6 Introduction to pygame

In this article, we present ***pygame***, a set of Python modules written by Pete Shinners that are designed to create multimedia programs and games. The **pygame** modules use the *Simple DirectMedia Layer* (*SDL*), which is a cross-platform library that provides a uniform API to access multimedia hardware. Module **pygame** allows programmers to access this library through Python. For more information about **pygame**, including extensive documentation, visit **www.pygame.org**.

24.7 Python CD Player

This article demonstrates how to create a simple CD-ROM player using **pygame**'s ***cdrom*** module (Fig. 24.4). Module **cdrom** contains class ***CD*** and methods to initialize a CD-ROM subsystem. Class **CD** represents the user's CD-ROM drive. Methods of class **CD** allow users to access an audio compact disc (CD) in a computer's CD-ROM drive. The program in Figure 24.4 also uses **Tkinter** and **Pmw** to create the CD-player interface. **Tkinter** and **Pmw** are introduced in Chapters 10 and 11 of our textbook *Python How to Program*.

```python
# Fig. 24.4: fig24_04.py
# Simple CD player using Tkinter and pygame.

import sys
import string
import pygame, pygame.cdrom
from Tkinter import *
from tkMessageBox import *
import Pmw

class CDPlayer( Frame ):
   """A GUI CDPlayer class using Tkinter and pygame"""

   def __init__( self ):
      """Initialize pygame.cdrom and get CDROM if one exists"""

      pygame.cdrom.init()

      if pygame.cdrom.get_count() > 0:
         self.CD = pygame.cdrom.CD( 0 )
      else:
         sys.exit( "There are no available CDROM drives." )

      self.createGUI()
      self.updateTime()

   def destroy( self ):
      """Stop CD, uninitialize pygame.cdrom and destroy GUI"""

      if self.CD.get_init():
         self.CD.stop()

      pygame.cdrom.quit()
```

Fig. 24.4 Python CD player. (Part 1 of 5.)

```
      Frame.destroy( self )

   def createGUI( self ):
      """Create CDPlayer widgets"""

      Frame.__init__( self )
      self.pack( expand = YES, fill = BOTH )
      self.master.title( "CD Player" )

      # display current track playing
      self.trackLabel = IntVar()
      self.trackLabel.set( 1 )
      self.trackDisplay = Label( self, font = "Courier 14",
         textvariable = self.trackLabel, bg = "black",
         fg = "green" )
      self.trackDisplay.grid( sticky = W+E+N+S )

      # display current time of track playing
      self.timeLabel = StringVar()
      self.timeLabel.set( "00:00/00:00" )
      self.timeDisplay = Label( self, font = "Courier 14",
         textvariable = self.timeLabel, bg = "black",
         fg = "green" )
      self.timeDisplay.grid( row = 0, column = 1, columnspan = 3,
         sticky = W+E+N+S )

      # play/pause CD
      self.playLabel = StringVar()
      self.playLabel.set( "Play" )
      self.play = Button( self, textvariable = self.playLabel,
         command = self.playCD, width = 10 )
      self.play.grid( row = 1, column = 0, columnspan = 2,
         sticky = W+E+N+S )

      # stop CD
      self.stop = Button( self, text = "Stop", width = 10,
         command = self.stopCD )
      self.stop.grid( row = 1, column = 2, columnspan = 2,
         sticky = W+E+N+S )

      # skip to previous track
      self.previous = Button( self, text = "|<<", width = 5,
         command = self.previousTrack )
      self.previous.grid( row = 2, column = 0, sticky = W+E+N+S )

      # skip to next track
      self.next = Button( self, text = ">>|", width = 5,
         command = self.nextTrack )
      self.next.grid( row = 2, column = 1, sticky = W+E+N+S )

      # eject CD
      self.eject = Button( self, text = "Eject", width = 10,
         command = self.ejectCD )
      self.eject.grid( row = 2, column = 2, columnspan = 2,
```

Fig. 24.4 Python CD player. (Part 2 of 5.)

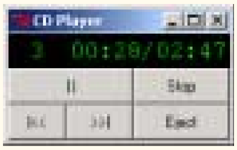
```
         sticky = W+E+N+S )

   def playCD( self ):
      """Play/Pause CD if disc is loaded"""

      # if disc has been ejected, reinitialize drive
      if not self.CD.get_init():
         self.CD.init()
         self.currentTrack = 1

         # if no disc in drive, uninitialize and return
         if self.CD.get_empty():
            self.CD.quit()
            return

         # if disc is loaded, obtain disc information
         else:
            self.totalTracks = self.CD.get_numtracks()

      # if CD is not playing, play CD
      if not self.CD.get_busy() and not self.CD.get_paused():
         self.CD.play( self.currentTrack - 1 )
         self.playLabel.set( "| |" )

      # if CD is playing, pause disc
      elif not self.CD.get_paused():
         self.CD.pause()
         self.playLabel.set( "Play" )

      # if CD is paused, resume play
      else:
         self.CD.resume()
         self.playLabel.set( "| |" )

   def stopCD( self ):
      """Stop CD if disc is loaded"""

      if self.CD.get_init():
         self.CD.stop()
         self.playLabel.set( "Play" )

   def playTrack( self, track ):
      """Play track if disc is loaded"""

      if self.CD.get_init():
         self.currentTrack = track
         self.trackLabel.set( self.currentTrack )

         # start beginning of track
         if self.CD.get_busy():
            self.CD.play( self.currentTrack - 1 )
         elif self.CD.get_paused():
            self.CD.play( self.currentTrack - 1 )
            self.playCD()   # re-pause CD
```

Fig. 24.4 Python CD player. (Part 3 of 5.)

```

   def nextTrack( self ):
      """Play next track on CD if disc is loaded"""

      if self.CD.get_init() and \
         self.currentTrack < self.totalTracks:
         self.playTrack( self.currentTrack + 1 )

   def previousTrack( self ):
      """Play previous track on CD if disc is loaded"""

      if self.CD.get_init() and self.currentTrack > 1:
         self.playTrack( self.currentTrack - 1 )

   def ejectCD( self ):
      """Eject CD from drive"""

      response = askyesno( "Eject pushed", "Eject CD?" )

      if response:
         self.CD.init()   # CD must be initialized to eject
         self.CD.eject()
         self.CD.quit()
         self.trackLabel.set( 1 )
         self.timeLabel.set( "00:00/00:00" )
         self.playLabel.set( "Play" )

   def updateTime( self ):
      """Update time display if disc is loaded"""

      if self.CD.get_init():
         seconds = int( self.CD.get_current()[ 1 ] )
         endSeconds = int( self.CD.get_track_length(
            self.currentTrack - 1 ) )

         # if reached end of current track, play next track
         if seconds >= ( endSeconds - 1 ):
            self.nextTrack()
         else:
            minutes = seconds / 60
            endMinutes = endSeconds / 60
            seconds = seconds - ( minutes * 60 )
            endSeconds = endSeconds - ( endMinutes * 60 )

            # display time in format mm:ss/mm:ss
            trackTime = string.zfill( str( minutes ), 2 ) + \
               ":" + string.zfill( str( seconds ), 2 )
            endTime = string.zfill( str( endMinutes ), 2 ) +  \
               ":" + string.zfill( str( endSeconds ), 2 )

            if self.CD.get_paused():

               # alternate pause symbol and time in display
               if not self.timeLabel.get() == "     ||    ":
```

Fig. 24.4 Python CD player. (Part 4 of 5.)

```
196                     self.timeLabel.set( "     ||     " )
197                  else:
198                     self.timeLabel.set( trackTime + "/" + endTime )
199
200               else:
201                  self.timeLabel.set( trackTime + "/" + endTime )
202
203         # call updateTime method again after 1000ms ( 1 second )
204         self.after( 1000, self.updateTime )
205
206  def main():
207     CDPlayer().mainloop()
208
209  if __name__ == "__main__":
210      main()
```



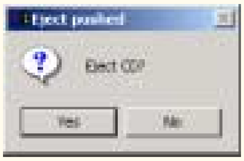


Fig. 24.4 Python CD player. (Part 5 of 5.)

Line 207 creates an object of class **CDPlayer** (lines 11–204)and invokes its **mainloop** method to start the application. The **CDPlayer** constructor (lines 14–25) initializes module **cdrom** (line 17) so the program can access methods for controlling and querying the computer's CD-ROM drive. The **if**/**else** structure (lines 19–22) determines the number of available CD-ROM drives by invoking **cdrom**'s **get_count** function. If at least one drive is present, line 20 creates a **pygame.cdrom.CD** object called **CD**. The value passed to the constructor is the identification (ID) number of the CD-ROM drive. If more than one CD-ROM drive is installed on a computer system, the program uses the primary CD-ROM drive. The constructor receives **0** as an argument, because the primary CD-ROM's drive identification number is always **0**. The program exits (line 22) if no CD-ROM drive exists.

After the program identifies that a CD-ROM drive exists, the program constructs a GUI for the CD player. Line 24 invokes method **createGUI** (lines 36–88) to create the CD-player interface. Method **createGUI** adds the components to the display (each component's action is discussed later in this section). Both the **Label** created to display the track number (**trackDisplay**) and the **Label** that displays the current track time (**timeDisplay**) have **textvariable**s—**trackLabel** and **timeLabel**—that update the CD-player display. Notice also that **Button play** has a **textvariable**—**playLabel**—which changes its display when the user pauses or plays a CD.

Once the GUI has been created, the constructor calls method **updateTime** (discussed momentarily). The program then enters the **mainloop**, in which the user can play, stop, pause, fast forward and backtrack through a CD by manipulating the CD player's GUI.

The other methods provide the functionality of a basic CD player. The **Play** button has callback method **playCD** (lines 90–120), which plays or pauses the CD. Line 94 determines whether the CD-ROM is initialized by invoking **CD** method ***get_init***. If the CD-ROM is not initialized, **playCD** initializes it and sets **currentTrack** to **1**—the first audio track. Line 99 determines whether the CD-ROM drive is empty by invoking **CD** method ***get_empty***. If the drive is empty, line 100 uninitializes the object with **CD** method ***quit*** and returns. Otherwise, line 105 obtains the total number of audio tracks on the disc from **CD** method ***get_numtracks*** and stores that value in object attribute **totalTracks**.

Method **playCD** tests for three cases—the CD is not playing, the CD is not paused and or the CD is paused. Line 108 determines whether the CD is not playing and not paused using methods **get_busy** and **get_paused**, respectively. If both of these conditions are true, line 109 calls method **play**. The method call specifies which track to play. Because the track numbers for a CD object begin with **0** and **currentTrack** is initialized to **1**, the value passed to method **play** is **1** less than **currentTrack** (line 109). Line 110 sets the **Play** button's text to contain the symbol for the pause button,**| |**, so the user can control the CD-ROM application properly.

Line 113 determines whether the CD is paused by invoking **get_paused**. If the CD is playing and not paused (e.g., the user has not pressed the pause button), **CD** method **pause** (line 114) pauses the CD. Line 114 sets the **Play** button's text to contain the word **Play**. Otherwise, the CD is paused, and the program calls **CD** method **resume** (line 119) to continue playing the track. As in line 110, the program sets the label on the button to the symbols for the **Paused** button.

The **Stop** button's associated callback is **stopCD** (lines 122–127). When a user presses **Stop**, line 125 determines whether the **CD** object is initialized. If it is, **CD** method **stop** is invoked to stop the CD, and the text on the **Play** button is set to **"Play"**. If a **CD** object is not initialized, calling **Stop** generates an error. If the **CD** object is initialized but not playing, calling **Stop** does nothing. This application plays the audio tracks sequentially, but the user can press the **|<<** or the **>>|** button to move backward or forward, respectively, through the audio tracks on the CD. The **>>|** button is associated with callback **nextTrack**, which skips to the next track on the CD (lines 143–148). If **CD** is initialized and the current track is not the last one, method **playTrack** is invoked, with the next track number specified (**currentTrack + 1**). Similarly, the **|<<** button is associated with callback **previousTrack**, which skips to the previous track on a CD (lines 150–154). If **CD** is initialized and the current track is not the first one, method **playTrack** is invoked, with the previous track number specified (**currentTrack - 1**).

Method **playTrack** (lines 129–141) plays a CD track. If the CD is initialized, line 133 sets **currentTrack** to the indicated number. Line 134 sets **trackLabel** to the new track number. If the CD is playing another song, line 138 plays the indicated track instead. If the CD is paused, however, lines 140–141 switch to the specified song and then leave the disc paused.

Callback **ejectCD** (lines 156–167) is bound to the **Eject** button. When a user clicks **Eject**, line 159 displays a **tkMessageBox** window with a message that asks the user whether the CD should be ejected. This is a safeguard against accidentally ejecting the CD. If the user chooses to eject the CD, **CD** is initialized because, trying to eject an uninitialized **CD** object is an error. Once the **CD** object has been initialized, the disc is ejected with **CD** method **eject** and **CD** is uninitialized (lines 162–164). Lines 165–167 set the CD-player interface to its initial appearance.

The CD player updates its display using method `updateTime` (lines 169–204), originally called in line 25. Line 172 determines whether `CD` is initialized. If it is not, execution skips to line 204.

Often, audio CDs list the duration of each track. This program displays the time that a song has been playing to the user. If the `CD` is initialized, `CD` method `get_current` returns the number of seconds that the track has played and assigns that number to variable `seconds` (line 173). Method `get_current` returns a two-element tuple of the current track number and the number of seconds that the song has been playing. Lines 174–175 obtain the track length from `CD` method `get_track_length`, specifying the current track (`currentTrack - 1`). This value is assigned to variable `endSeconds`. Lines 178–179 ensure that the tracks play consecutively until the entire disk has been played. Lines 181–184 use `seconds` and `endSeconds` to determine the current time and end time in minutes and seconds.

Lines 187–188 create a string for the current track time (`trackTime`). The string is in the form *mm:ss,* in which *mm* is minutes and *ss* is seconds. Note that `string` function `zfill` pads the string with zeros. This ensures that both minutes and seconds are displayed with two digits.

Line 192 determines whether the CD is paused. If the CD is not paused, `timeDisplay` is updated to display the current time (line 201). Otherwise, `timeDisplay` is updated to either the current time or to the symbol that represents pause (lines 195–198). This causes the display to flash between the track time and the pause symbol when paused.

Method `updateTime` invokes component method `after`. Method `after` registers a callback that executes after a specified number of milliseconds. Line 204 ensures that method `updateTime` is called approximately every 1000 milliseconds (one second).

When the user is finished with the CD player, the program destroys the window and invokes the `CDPlayer`'s `destroy` method to terminate the CD player (lines 27–34). Line 30 determines whether `CD` is initialized. If `CD` is initialized, `CD` method `stop` is invoked to stop the CD. If the program does not call method `stop`, the CD would continue to play after the user destroys the window. Lines 33–34 uninitialize the `pygame cdrom` module and destroy the frame by calling `Frame` method `destroy`.

Good Programming Practice 24.1

For `Tkinter` programs, a `destroy` method acts as a destructor.