Author: Cole Vikupitz
Last Modified: 11/8/2017

Introduction

In this document, I will cover at a high level how the new auditory clock program functions. This new version allows the user to set the day and time and receive audio feedback, but now makes use of Python's Tkinter library. This application attempts to make use of visual displays to help the user better learn the system. The application was written in Python 3, and makes use of the Tkinter library. It also uses the PyAudio packages and sound.py, along with the sound files used in the previous version.

Files & Structure

The file structure is the same as the previous application. The sound files used are in the directory *program_wav_files* and its subfolders. The directory and sub-directories are hardcoded, therefore should not be changed. The sound.py file is also used, so must be placed in the same directory as set_clock.py.

sound.py

This section is the same as the previous version, no changes were made. The documentation for sound.py has been copied over for the reader's convenience.

This source file is what handles all of the sound files within the application, and was provided by Anthony Hornof. There are 3 things in this file for the programmer to understand: the class *Play*, the method *cleanup*, and the method *combine_wav_files*.

The Play class allows the programmer to play single audio files. To do this, the programmer just needs to instantiate the Play class (no need to assign it to anything). The programmer passes in the name of the file to play, and the file is played in the constructor. The sound file keeps track of the last file played (which is None if no previous file was played), which is an audio stream; therefore the programmer must invoke *sound.cleanup()* at the end of the program to close the last stream. Example of proper use shown below.

```
import sound

if __name__ == "__main__":

	sound.Play("test_file.wav")
	# Do other stuff here ...
	sound.cleanup()
```

When the application needs to combine and play multiple audio source files together, it uses the combine_wav_files method. This method takes 2 or more arguments; the first argument is the name of the audio file that will be created, and the second and following arguments are the audio files that are to be combined together. The files are all combined and saved into one bigger audio file with the name the programmer gives it in the first parameter. Example of proper usage is shown below.

```
import sound

if __name__ == "__main__":

	wav_file = "temp.wav"
	sound.combine_wav_file(wav_file, "one.wav", "two.wav", "three.wav")
	sound.Play(wav_file)
	sound.cleanup()
```

set_clock.py

Many changes have been made since the previous version. Since a GUI is now being used, the application must build and display a window with labels, buttons, displays, etc. All of these components are then used to help the user learn the auditory clock system.

The clock setting program is now designed as an enormous class called *ClockWindow*, then calls a constructor on the class to create and display the window. Above the constructor are several variables copied and pasted from the previous version for frequently played sound files. Inside the constructor itself, the basic window is built with a set dimension, title, and all the components are created and displayed. There are buttons that are created displayed that allow the user to use them to set the day and time. In addition, the keys 'j', 'k', and the spacebar are registered to local functions so that the user may use these keys as well. This means that the readchar package is not used in this version.

Outside the constructor, a number of methods exist in the class. One method exists for scrolling forward and another for scrolling backwards, depending on what mode the user is in. For example, scrolling forward in weekday mode means that the user could scroll from Saturday to Sunday, and vice versa for scrolling backwards. There is also another function for transitioning to the next mode, invoked when the user presses the spacebar or clicks the set button.

Other methods have been copied over from the previous version to broadcast auditory output to the user. There is a method for reporting the current hour, weekday, minute, AM or PM, and the full time. There also exists a method for announcing the current mode the user is in.

An outline of the code is shown below, with a high-level description for each of the methods listed.

Author: Cole Vikupitz
Last Modified: 11/8/2017

```
import tkinter

import sound.py


Class ClockWindow:

	# Assigns audio variables here

	def __init__(self):

		# Creates the window, components, initializes variables, etc.

	def display_time(self):

		# Displays the currently selected day and time at top of window

	def forward(self):

		# Scroll forward in weekday, hour, minute, etc.

	def backward(self):

		# Scroll backward in weekday, hour, minute, etc.

	def next_mode(self):

		# Transition the user to the next mode

	def play_current_weekday(self):

		# Auditory output of the currently selected weekday

	def play_current_hour(self):

		# Auditory output of the currently selected hour

	def play_current_minute(self):

		# Auditory output of the currently selected minute

	def play_AM_PM(self):

		# Auditory output of the A.M. or P.M., whichever is selected

	def play_complete_time(self):

		# Auditory output of the entire currently selected time


# Create & display the window

window = ClockWindow()
```