5

SPEAKING APPLICATIONS

Now that you know how to make Python talk and listen, we'll create several real-world applications that utilize those skills. But before that, you'll create a local package. Since you'll use the *mysr* and *mysay* local modules in every chapter for the reminder of the book, you'll create a Python package to contain all local modules. This way, you don't need to copy and paste these modules to the folders of individual chapters. This also helps keep the code consistent throughout the book. You'll learn how a Python package works and how to create one yourself along the way.

In the first application, you'll build a Guess the Number game that takes voice commands and talks back to you in a human voice.

A Python *module* is a single file with the *.py* extension. In contrast, a Python *package* is a collection of Python modules contained in a single directory. The directory must have a file named *__init__.py* to distinguish it from a directory that happens to have *.py* extension files in it.

I'll guide you through the process of creating a local package step-by-step.

Create Your Own Python Package

To create a local Python package, you need to create a separate directory for it and place all related files into it. In this section, you'll create a local package to contain both our speech recognition and text-to-speech module files—namely, *mysr.py* and *mysay.py*.

Create a Package Directory

First, you need to create a directory for the package.

In this book, you use a separate directory for each chapter. For example, all Python scripts and related files in this chapter are placed in the directory */mpt/ch05/*. Since you are creating a package to be used for all chapters in this book, you'll create a directory parallel to all chapters. Specifically, you'll use the directory */mpt/mptpkg/*, where *mptpkg* is the package name. The diagram in Figure 5-1 explains the position of the package relative to the book chapters.

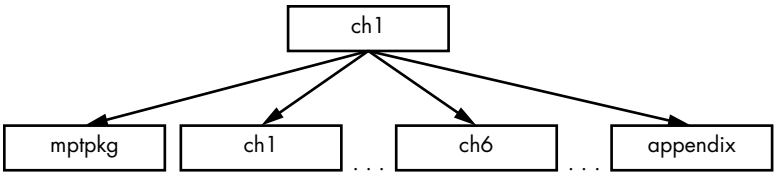

Figure 5-1: The position of the mptpkg *package relative to the chapter folders*

As you can see, the package directory is parallel to the chapter directories, which are all contained in the directory for the book, */mpt*, as in *Make Python Talk*.

Create Necessary Files for Your Package

Next, you need to create and place necessary files in the package.

First, copy and paste the two modules you created in Chapters 3 and 4, *mysr.py* and *mysay.py*, in the package directory */mpt/mptpkg/*. Do not make any changes to the two files.

Then save the following script, *__init__.py*, in the package directory */mpt/mptpkg/* (or you can download it from the book's resources):

```
from .mysr import voice_to_text
from .mysay import print_say
```

The purpose of this file is twofold: it imports `voice_to_text()` and `print_say()` so you can use those functions at the package level, and it also tells Python that the directory is a package, not a folder that happens to have Python scripts in it.

Finally, save the following script, *setup.py*, in the book directory */mpt*, one level above the package directory */mpt/mptpkg/*. The script is also available from the book's resources.

```
from setuptools import setup
setup(name='mptpkg',
version='0.1',
description='Install local package for Make Python Talk',
author='Mark Liu',
author_email='mark.liu@uky.edu',
packages=['mptpkg'],
zip_safe=False)
```

The file provides information about the package, such as the package name, author, version, descriptions, and so on.

You'll learn how to install this local package on your computer next.

Install Your Package

Because you'll modify the local package and add more features to it later in the book, it's better to install the package in editable mode.

Open your Anaconda prompt (Windows) or a terminal (Mac or Linux) and activate your virtual environment for this book, *chatting*. Run the following command:

```
pip install -e path-to-mpt
```

Replace *path-to-mpt* with the actual directory path of */mpt*. For example, the book directory */mpt* is *C:\mpt* on my office computer that runs the Windows operating system, so I installed the local package using this command:

```
pip install -e C:\mpt
```

On my Linux machine, the path to the */mpt* directory is */home/mark/Desktop/mpt*, so I installed the local package using this command:

```
pip install -e /home/mark/Desktop/mpt
```

The `-e` option tells the Python to install the package in editable mode so that you can modify the package anytime you need to.

With that, the local package is installed on your computer.

Test Your Package

Now that you have installed your self-made local package, you'll learn how to import it.

You'll write a Python script to test the package you just created.

Let's revisit the script *repeat_me1.py* from Chapter 4. Enter the following lines of code in your Spyder editor and save it as *repeat_me2.py* in your Chapter 5 directory */mpt/ch05/*:

```
# Import functions from the local package mptpkg
from mptpkg import voice_to_text
from mptpkg import print_say

while True:
    print('Python is listening...')
    inp = voice_to_text()
    if inp == "stop listening":
        print_say(f'you just said {inp}; goodbye!')
        break
    else:
        print_say(f'you just said {inp}')
        continue
```

First, import the functions `voice_to_text()` and `print_say()` from the *mptpkg* package directly. Recall that in the script *__init__.py*, you've already imported the two functions from the modules *.mysr* and *.mysay* to the package. As a result, here you can directly import the two functions from the package.

The rest of the script is the same as that in *repeat_me1.py*. It repeats what you say. If you say, "Stop listening," the script stops.

The following is an interaction with *repeat_me2.py*, with my voice input in bold:

```
Python is listening...
you just said how are you
Python is listening...
you just said I am testing a python package
Python is listening...
you just said stop listening; goodbye!
```

As you can see, the script is working properly, which means you've successfully imported functions from the local package.

More on Python Packages

Before you move on, I want to mention a couple of things about Python packages.

First, you can add more modules to your package. Later in this book, you'll add more modules to the existing local package *mptpkg*. You'll use just one local package for the whole book. This will reduce the number of directories and help organize your files.

Second, if you have an interesting package that you want to share with the rest of the world, you can easily do so. You just need to add a few more files, such as the license, a README file, and so on. For a tutorial on how to distribute your Python packages, see, for example, the Python Packaging Authority website, *https://packaging.python.org/tutorials/packaging-projects/*.

```
  # Ask what you want to know
❶ print_say("What do you want to know?")
  # Capture your voice input
❷ my_query = voice_to_text()
  print_say (f"you said {my_query}")
  # Obtain answer from Wikipedia
  ans = wikipedia.summary(my_query)
  # Say the answer in a human voice
  print_say(ans[0:200])
```

Listing 5-8: Python code for a voice-controlled talking Wikipedia

Once you start the script, a voice asks, "What do you want to know?" ❶. At ❷, the script calls `voice_to_text()` to convert your voice input into text. Then, the script retrieves the response to your question from Wikipedia, saves it as a string variable *ans*, and converts it to a human voice.

After running the script, if you say to the microphone, "US Federal Reserve Bank," you'll get a result similar to this:

```
What do you want to know?
you said U.S. federal reserve bank
The Federal Reserve System (also known as the Federal Reserve or simply the
Fed) is the central banking system of the United States of America. It was
created on December 23,
1913, with the enactment
```

I've added the `[0:200]` character limit behind the variable `ans`, so only the first 200 characters of the result are printed and spoken.

And just like that, you have your own voice-controlled talking Wikipedia. Ask away!

TRY IT OUT

Run *wiki_hs.py* and ask Wikipedia about the city you live in now (or the state if the city is not in Wikipedia). See what the output is like.

Voice-Activated Music Player

Here you'll learn how to get Python to play a certain artist or genre of music just by asking for it with a phrase like "Python, play Selena Gomez." You'll speak the name of the artist you want to listen to, and the script will receive that as keywords and then search for those keywords in a particular folder. To do this, you need to be able to traverse files and folders.

Traverse Files in a Folder

Suppose you have a subfolder *chat* in your chapter folder. If you want to list all files in the subfolder, you can use this *traverse.py* script:

```
import os

with os.scandir("./chat") as files:
    for file in files:
        print(file.name)
```

First, the script imports the *os* module. This module gives the script access to functionalities that are dependent on the operating system, such as accessing all files in a folder.

Next, you put all files in the subfolder *chat* into a list called *files*. The script goes through all items in the list, and prints out the name of each item.

The output from the preceding script is as follows after I run it on my computer:

```
book.xlsx
desk.pdf
storm.txt
graduation.pptx
--snip--
HilaryDuffSparks.mp3
country
classic
lessons.docx
SelenaGomezWolves.mp3
TheHeartWantsWhatItWantsSelenaGomez.mp3
```

As you can see, we can traverse all the files and subfolders in a folder and print out their names. Filenames include the file extension. Subfolders have no extension after the subfolder name. For example, I have two folders, *country* and *classic*, in the folder *chat*. As a result, you see `country` and `classic` in the preceding output.

Next, you'll use this feature to select a song you want to play.

Python, Play Selena Gomez

The script in Listing 5-9, *play_selena_gomez.py*, can pick out a song by whatever artist you name (for example, Selena Gomez) and play it. Either save your songs in the subfolder *chat* or replace the file path with a path to somewhere on your computer that you keep music.

```
# Import the required modules
import os
import random
from pygame import mixer
```

```
# Import functions from the local package mptpkg
from mptpkg import voice_to_text
from mptpkg import print_say

# Start an infinite loop to take your voice commands
❶ while True:
    print_say("how may I help you?")
    inp = voice_to_text()
    print_say(f"you just said {inp}")
    # Stop the script if you say 'stop listening'
    if inp == "stop listening":
        print_say("Goodbye! ")
        break
    # If 'play' is in voice command, music mode is activated
  ❷ elif "play" in inp:
        # Remove the word play from voice command
      ❸ inp = inp.replace('play ','')
        # Separate first and last names
        names = inp.split()
        # Extract the first name
        Firstname = names[0]
        # Extract the last name
        if len(names)>1:
            lastname = names[1]
        # If no last name, use the first name as last name;
        else:
            lastname = firstname
        # Create a list to contain songs
        mysongs = []
        # If either first name or last name in the file name, put in list
        with os.scandir("./chat") as files:
            for file in files:
          ❹ if (firstname in file.name or lastname in file.name) \
and "mp3" in file.name:
                mysongs.append(file.name)
        # Randomly select one from the list and play
      ❺ mysong = random.choice(mysongs)
        print_say(f"play the song {mysong} for you")
        mixer.init()
        mixer.music.load(f'./chat/{mysong}')
        mixer.music.play()
        break
```

Listing 5-9: Python code to voice activate a song by an artist on your computer

We first import the needed modules. In particular, we import the *os* module to traverse files and the *random* module to randomly select a song from a list the script will build. We use `mixer()` in the *pygame* module to play the music file.

We then start an infinite loop ❶ to put the script in standby mode to wait for your voice commands. If the script detects the word *play* in your voice command, the music mode is activated ❷. We then replace the word `play` and the whitespace behind it with an empty string ❸ so that your command "Play Selena Gomez" becomes `Selena Gomez`. The next command

separates the first name and the last name. For artists who are known by just their first names (such as Madonna, Prince, or Cher), we put their first name as a placeholder in the variable `lastname`.

We then traverse through all files in the subfolder *chat*. If a file has the *mp3* extension and contains either the first or the last name ❹, it will be added to the list *mysongs*. We use `choice()` from the *random* module to randomly select a song in the list *mysongs* ❺ and load it with `mixer.music.load()`. After that, we use `mixer.music.play()` to play it.

As a result, once you say to the script, "Play Selena Gomez," one of the two songs in the subfolder *chat*, *SelenaGomezWolves.mp3* or *TheHeartWantsWhatItWantsSelenaGomez.mp3*, will start playing.

NOTE *We use the* pygame *module to play music files in this book. Depending on which operating system you are using, other modules, such as* playsound *or* vlc, *can also play music files in Python. Alternatively, you can use* `os.system()` *to open music files in your computer's default music player, as discussed in Chapter 3.*

TRY IT OUT

Save several songs by your favorite artist, making sure that the filenames contain the artist's first and last name. Then edit and run *play_selena_gomez.py* so that when you say, "Python, play *Firstname Lastname*," one of your songs will start playing.

Python, Play a Country Song

What we'll do now is similar to interacting with the script *play_selena_gomez.py*, but here you'll learn how to access different subfolders by using the *os* module as well as a different way of playing music files.

Suppose you've organized your songs by genre. You put all classical music files in the subfolder *classic*, and all country music files in the folder *country*, and so on. You've placed these subfolders in the folder *chat* you just created.

We want to write a script so that when you say, "Python, play a country song," the script will randomly select a song from the folder *country* and play it. Enter the code in Listing 5-10 and save it as *play_genre.py*.

```
# Import needed modules
import os
import random
from pygame import mixer

# Import functions from the local package mptpkg
from mptpkg import voice_to_text
from mptpkg import print_say
```

```
while True:
    print_say("how may I help you?")
    inp = voice_to_text().lower()
    print_say(f'you just said {inp}')
    if inp == "stop listening":
        print_say('Goodbye!')
        break
    elif "play a" in inp and "song" in inp:
        # Remove 'play a' and 'song' so that only the genre name is left
      ❶ inp = inp.replace('play a ','')
      ❷ inp = inp.replace(' song','')

        # Go to the genre folder and randomly select a song
        with os.scandir(f"./chat/{inp}") as entries:
            mysongs = [entry.name for entry in entries]
        # Use pygame mixer to play the song
      ❸ mysong = random.choice(mysongs)
        print_say(f"play the song {mysong} for you")
        mixer.init()
        mixer.music.load(f"./chat/{inp}/{mysong}")
        mixer.music.play()
        break
```

Listing 5-10: Python code to voice activate a song by genre

Python checks for the terms *play a* and *song* in the voice command and activates the music mode if it finds them. The script then replaces `play a` ❶ and `song` ❷ as well as the whitespace behind them with an empty string, leaving only the genre—`country`, in this case—in the voice command. This is used as the folder for the script to search: in this case, *./chat/country*. Finally, the script randomly selects a song from the folder ❸ and plays it.

Note that we use `lower()` after `voice_to_text()` in the script so that the voice command is all lowercase. We do this because the script sometimes converts the voice command into `play A Country Song`. We can avoid mismatch due to capitalization. On the other hand, the path and filenames are not case sensitive, so even if you have capital letters in your path or filenames, there will not be any mismatch.

TRY IT OUT

Organize your music into various categories. Save a few songs in the subfolder *classic* in the *chat* folder you created. If you say, "Play a classic song," see if a song in the folder will start playing.

Summary

In this chapter, you first learned to create a Python package to contain the local text-to-speech and speech recognition modules. After that, you built several real-world applications that can understand voice commands, react, and speak.

You created a voice-controlled, talking Guess the Number game. In the game, you pick a number between one and nine and interact with the script to let it guess. Then you learned how to parse text to extract a news summary from the NPR website, adding the speech recognition and text-to-speech features to make a voice-controlled newscast.

You learned how to use the *wikipedia* module to obtain answers to your inquiries.

You traversed files in a folder on your computer by using the *os* module, and then created a script that plays a genre or artist when you ask it to.

Now that you know how to make Python talk and listen, you'll apply both features to many other interesting situations throughout the rest of the book so that you can interact with your computer via voice only.

End-of-Chapter Exercises

1. Modify *guess_hs.py* so that the third guess of the script is two instead of one.
2. Change *wiki.py* so that it prints out the first 300 characters of the result from Wikipedia.
3. Modify *play_genre.py* so that the script plays music by using the *os* module and your default music player on your computer, instead of the *pygame* module.
4. Suppose the music files on your computer are not in MP3 format but in WAV format. How can you modify *play_selena_gomez.py* so that the script still works?