B
RUNNING PROGRAMS

If you have a program open in Mu, running it is a simple matter of pressing F5 or clicking the Run button at the top of the window. This is an easy way to run programs while writing them, but opening Mu to run your finished programs can be a burden. There are more convenient ways to execute Python scripts, depending on which operating system you're using.

Running Programs from the Terminal Window

When you open a terminal window (such as Command Prompt on Windows or Terminal on macOS and Linux), you'll see a mostly blank window into which you can enter text commands. You can run your programs from the terminal, but if you're not used to it, using your computer through a terminal (also called a *command line*) can be intimidating: unlike a graphical user interface, it offers no hints about what you're supposed to do.

To open a terminal window on Windows, click the Start button, enter Command Prompt, and press ENTER. On macOS, click on the Spotlight icon in the upper right, type Terminal, and press ENTER. On Ubuntu Linux, you can press the WIN key to bring up Dash, type Terminal, and press ENTER. The keyboard shortcut CTRL-ALT-T will also open a terminal window on Ubuntu.

Just as the interactive shell has a >>> prompt, the terminal will display a prompt for you to enter commands. On Windows, it will be the full path of the folder you are currently in:

```
C:\Users\Al>your commands go here
```

On macOS, the prompt shows your computer's name, a colon, the current working

directory (with your home folder represented as ~ for short), and your username, followed by a dollar sign ($):

```
Als-MacBook-Pro:~ al$ your commands go here
```

On Ubuntu Linux, the prompt is similar to macOS's, except it begins with the username and an @ sign:

```
al@al-VirtualBox:~$ your commands go here
```

It's possible to customize these prompts, but that's beyond the scope of this book.

When you enter a command, like python on Windows or python3 on macOS and Linux, the terminal checks for a program with that name in the folder you're currently in. If it doesn't find it there, it will check the folders listed in the PATH environment variable. You can think of *environment variables* as variables for your entire operating system. They'll contain a few system settings. To see the value stored in the PATH environment variable, run echo %PATH% on Windows and echo $PATH on macOS and Linux. Here's an example on macOS:

```
Als-MacBook-Pro:~ al$ echo $PATH
/Library/Frameworks/Python.framework/Versions/3.7/bin:/usr/local/bin:/usr/
bin:/bin:/usr/sbin:/sbin
```

On macOS, the *python3* program file is located in the */Library/Frameworks/Python.framework/Versions/3.7/bin* folder, so you don't have to enter /Library/Frameworks/Python.framework/Versions/3.7/bin/python3 or switch to that folder first to run it; you can enter python3 from any folder, and the terminal will find it in one of the PATH environment variable's folders. Adding a program's folder to the PATH environment variable is a convenient shortcut.

If you want to run a *.py* program, you must enter python (or python3) followed by the *.py* filename. This will run Python, and in turn Python will run the code it finds in that *.py* file. After the Python program finishes, you'll return to the terminal prompt. For example, on Windows, a simple "Hello, world!" program would look like this:

```
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Al>python hello.py
Hello, world!

C:\Users\Al>
```

Running python (or python3) without any filename will cause Python to launch the interactive shell.

Running Python Programs on Windows

There are a few other ways you can run Python programs on Windows. Instead of opening a terminal window to run your Python scripts, you can press WIN-R to open the Run Dialog box and enter py C:*path\to\your\pythonScript.py*, as shown in Figure B-1. The *py.exe* program is installed at *C:\Windows\py.exe*, which is already in the PATH environment variable, and typing the *.exe* file extension is optional when running programs.

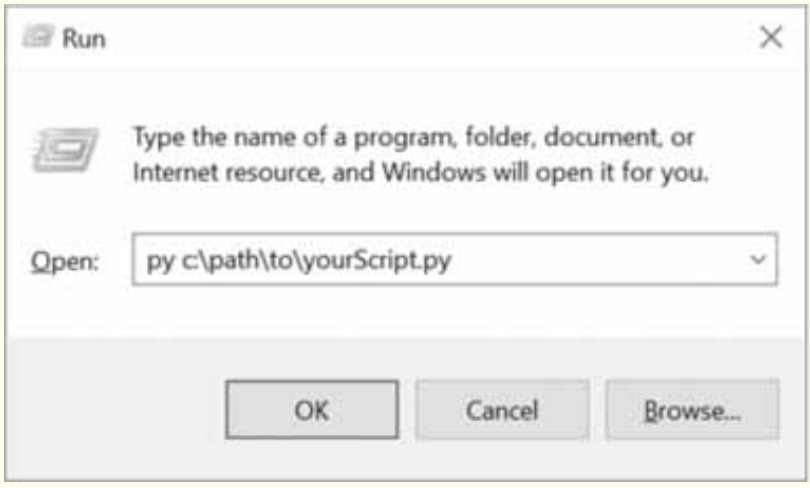

Figure B-1: The Run dialog on Windows

The downside of this method is that you must enter the full path to your script. Also, while running your Python script from the dialog box will open a new terminal window to display its output, this window will automatically close when the program ends, and you

might miss some output.

You can solve these problems by creating a *batch script*, which is a small text file with the *.bat* file extension that can run multiple terminal commands, much like a shell script in macOS and Linux. You can use a text editor such as Notepad to create these files.

To make a batch file, make a new text file containing a single line, like this:

```
@py.exe C:\path\to\your\pythonScript.py %*
@pause
```

Replace this path with the absolute path to your own program and save this file with a *.bat* file extension (for example, *pythonScript.bat*). The @ sign at the start of each command prevents it from being displayed in the terminal window, and the %* forwards any command line arguments entered after the batch filename to the Python script. The Python script, in turn, reads the command line arguments in the sys.argv list. This batch file will keep you from having to type the full absolute path for the Python program every time you want to run it. In addition, @pause will add "Press any key to continue..." after the end of the Python script to prevent the program's window from disappearing too quickly. I recommend you place all your batch and *.py* files in a single folder that already exists in the PATH environment variable, such as *C:\Users\<USERNAME>*.

With a batch file set up to run your Python script, you don't need to open a terminal window and type the full file path and name of your Python script. Instead, just press WIN-R, enter pythonScript (the full *pythonScript.bat* name isn't necessary), and press ENTER to run your script.

Running Python Programs on macOS

On macOS, you can create a shell script to run your Python scripts by creating a text file with the *.command* file extension. Create a new file in a text editor such as TextEdit and add the following content:

```
#!/usr/bin/env bash
python3 /path/to/your/pythonScript.py
```

Save this file with the *.command* file extension in your home folder (for example, on my computer it's */Users/al*). In a terminal window, make this shell script executable by running chmod u+x *yourScript*.command. Now you will be able to click the Spotlight icon (or press ⌘-SPACE) and enter *yourScript*.command to run the shell script, which in turn will run your Python script.

Running Python Programs on Ubuntu Linux

Running your Python scripts in Ubuntu Linux from the Dash menu requires considerable setup. Let's say we have a */home/al/example.py* script (your Python script could be in a different folder with a different filename) that we want to run from Dash. First, use a text editor such as gedit to create a new file with the following content:

```
[Desktop Entry]
Name=example.py
Exec=gnome-terminal -- /home/al/example.sh
Type=Application
Categories=GTK;GNOME;Utility;
```

Save this file to the */home/<al>/.local/share/applications* folder (replacing *al* with your own username) as *example.desktop*. If your text editor doesn't show the *.local* folder (because folders that begin with a period are considered hidden), you may have to save it to your home folder (such as */home/al*) and open a terminal window to move the file with the mv /home/al/example.desktop /home/al/.local/share/applications command.

When the *example.desktop* file is in the */home/al/.local/share/applications* folder, you'll be able to press the Windows key on your keyboard to bring up Dash and type *example.py* (or whatever you put for the Name field). This opens a new terminal window (specifically, the gnome-terminal program) that runs the */home/al/example.sh* shell script, which we'll create next.

In the text editor, create a new file with the following content:

```
#!/usr/bin/env bash
python3 /home/al/example.py
```

```
bash
```

Save this file to */home/al/example.sh*. This is a shell script: a script that runs a series of terminal commands. This shell script will run our */home/al/example.py* Python script and then run the bash shell program. Without the bash command on the last line, the terminal window would close as soon as the Python script finishes and you'd miss any text that `print()` function calls put on the screen.

You'll need to add execute permissions to this shell script, so run the following command from a terminal window:

```
al@ubuntu:~$ chmod u+x /home/al/example.sh
```

With the *example.desktop* and *example.sh* files set up, you'll now be able to run the *example.py* script by pressing the Windows key and entering *example.py* (or whatever name you put in the `Name` field of the *example.desktop* file).

Running Python Programs with Assertions Disabled

You can disable the `assert` statements in your Python programs to gain a slight performance improvement. When running Python from the terminal, include the `-O` switch after `python` or `python3` and before the name of the *.py* file. This will run an optimized version of your program that skips the assertion checks.

Read the author's other Creative Commons licensed Python books.