TRAN DUC LOI

PYTHON FOR DESKTOP APPLICATIONS

Develop, pack and deliver Python apps with TkInter and Kivy

FIRST EDITION

PYTHON FOR DESKTOP APPLICATIONS

How to develop, pack and deliver Python applications

with TkInter and Kivy

First Edition

Copyright

Copyright © 2020 Tran Duc Loi

All right reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor publisher, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

First released: October 2020

Revision: 20210803

Asset files used in this book are created by ***Tran Duc Loi***.

About the Author

Tran Duc Loi aka ***Leo Tran*** is a fan of minimalism and an enthusiast Python programmer since 2014 when he accidentally asked to join a new Python community with some friends.

Beside Python, he also added JavaScript to his stack with React JS for frontend and React Native for mobile development.

By the same Author

- Python For Desktop Applications - How to develop, pack and deliver Python applications with TkInter and Kivy
- METABASE 101 – Everything you need to setup your open source business intelligence system

Table of Contents

PYTHON FOR DESKTOP APPLICATIONS 2
Copyright 3
About the Author 4
By the same Author 5
Table of Contents 6
Preface 9
What this book covers 9
What you need for this book 11
Who this book is for 12
How to use this book 13
Conventions 14
Reader feedback 16
Downloading example code 17
Chapter 1: Introduction 19
1.1. Introduction 19
PIP 21
Wheels 21
Virtual Environment 22
GIL 24
CLI and GUI 25
Anaconda and Miniconda 27
1.2. Environment Setup 29
Choosing the right Python version 29
32-bit or 64-bit? 30

Text Editor 30
Git 31
The first Python application 31
The second Python application 35
Working with Virtual Environment 37
1.3. References 40
Chapter 2: Creating a File Downloader with Tkinter 41
2.1. Creating a basic GUI application with Tkinter 42
2.2. Creating a Python downloader application 49
Creating a console file downloader with progress information 49
Creating a GUI file downloader 55
Exercises 68
2.3. Creating a executable (.exe) for the GUI Downloader App 69
What about other packager? 69
Installing dependencies 70
Packing the GUI Downloader 70
UPX or NOUPX 73
2.4. Creating an installer for GUI Downloader App 76
Working with the NSI file 77
Running the setup 85
2.5. References 93
Chapter 3: Creating a Music Player with Kivy 94
3.1. Preparing the environment 96
3.2. Creating a simple GUI with Kivy 97
3.3. Working with .kv files 101

3.4. Creating a music player application with Kivy 106
Kivy Logger 117
3.5. Creating an executable for the Python Music Player 119
3.6. References 124
Chapter 4: Debugging 125
4.1. Removing the -w option 126
4.2. Using a file logger 128
4.3. Data files 128
4.4. UPX and vcruntime140.dll 129
4.5. Using DependencyWalker (Windows only) 129
4.6. Using another packager 134
Using cx_Freeze 134
4.7. References 139
Appendix 1: List of figures 140
Appendix 2: List of examples 143

Preface

What this book covers

Chapter 1, Introduction shows you some fundamental concepts of Python such as pip, wheel, virtual environment, GIL, CLI, and GUI, which tools we will use, how to set them up.

Chapter 2, Creating a File Downloader with TKInter introduces how to develop a Python file downloader application with simple GUI using **`TKInter`** library. This chapter also guides you on how to pack your application using **`PyInstaller`** and make a setup using **`NSIS`**.

Chapter 3, Creating a Music Player with Kivy walks through how to make a music player with **`Kivy`**. We will start with a very simple **`Kivy`** application then eventually build a more complex one. We also pack our music player up using **`PyInstaller`**.

Chapter 4, Debugging shows you how to debug your applications if something goes wrong. Useful tips and handy **`DependencyWalker`** debug tool guide. In this chapter, you will also be introduced to **`cx_Freeze`** to build/freeze a **`wx_Python`** application.

Please note that to keep the book briefly and to help readers focus on main target of the book, creating desktop applications, I will not cover tests in this edition. Depend on the received feedback, I will consider adding more features in the next editions.

What you will need for this book

In this book, we'll use:

- Python 3.7.9 64-bit
- Windows 10 64-bit
- PyInstaller 4.0
- NSIS 3.06.1
- Microsoft Visual Code 1.49.0 as my main editor
- A Git client (Git-scm 2.27.0)
- Kivy 1.11.1
- cx_Freeze 6.2
- DependencyWalker 2.2 64-bit

All the packages/tools are the latest versions at the time of writing.

Who this book is for

This book is for any level of Python programmers.

If you have known Python before, it would be easier for you to read this book.

If you are a beginner, it still be fine for you to get start Python programming with this book. You need to read chapter 01 and explore as much as possible to understand some basic concepts: virtual environment, multithreading, pip, wheels.

At any level, I recommend you try to explore hyperlinks at the end of each chapter.

How to use this book

This is a ***practical*** book. That means you will need to get your hands dirty.

You need to:

1. Install the tools.
2. Clone the code from the book's repository. All examples of a chapter will be in a single folder.
3. Run the examples to see how it works.
4. Dive into the code with your favorite IDE.
5. Read the notes to understand deeply, follow the links for more detail.
6. Complete the exercises if available.
7. Modify the code to adapt your purposes.

It's not about using the examples in this book to make commercial products, they are here for demonstration and being kept as simple as possible hence they are not shiny enough to be in production. You need to read them to understand the fundamental concepts and techniques.

Conventions

This book has some conventions that you should be familiar with in order to get more from this book.

One of my favourite wheel libraries is:
https://www.lfd.uci.edu/~gohlke/pythonlibs/

This is the information box, tips, and tricks. I usually explain things and answer some common questions here. It is an interesting box.

```
12  # ------------------------------------
13
14  import datetime, os, sys, platform
15
16  def hello():
17      print("Hello, this is Python version", s
```

This is the source code of the examples. We need the line number as we will use line numbers as an anchor to explain the code.

```
L:\python4desktop\chapter01>python --version
Python 3.7.9

L:\python4desktop\chapter01>pip --version
pip 20.1.1 from c:\python37\lib\site-packages\
```

This is a command that runs in PowerShell or Command-line (in Windows) or Terminal (Linux).

`print()` command to print to output.
`sys.version` return the current version of the running
`platform.system()` returns the name of the operating
`datetime.datetime.now()` returns current date and tir
the computer's datetime

File names, commands, folder names usually use bold format with different font family than normal text just to emphasize them.

Reader feedback

All feedback about the content of the book should be sent to the Telegram group at https://t.me/py4da or emailed directly the author at loitranduc@gmail.com. Let us know what you think about this book – what you liked or may have disliked.

To give any feedback about the source code, please:

- To find support interactively, chat with us at https://t.me/py4da (the official book's Telegram group)
- If you don't think you can provide a patch, open a New Issue at: https://github.com/loitd/python4desktop/issues
- If you think you can make some patch/fix the issue, fork the repository and make a Pull Request at: https://github.com/loitd/python4desktop/pulls.
- Please don't email me problems with source code. It should be done using the Git tool.

For more information about **`Github`** flow, please refer to: https://docs.github.com/en/free-pro-team@latest/github/collaborating-with-issues-and-pull-requests

Reader feedback is important for us to develop titles that you really get the most out of.

Downloading example code

All the source code in the book can be found at https://github.com/loitd/python4desktop. You should use **`git clone https://github.com/loitd/python4desktop`** command or download directly from the **`Github`** repository.

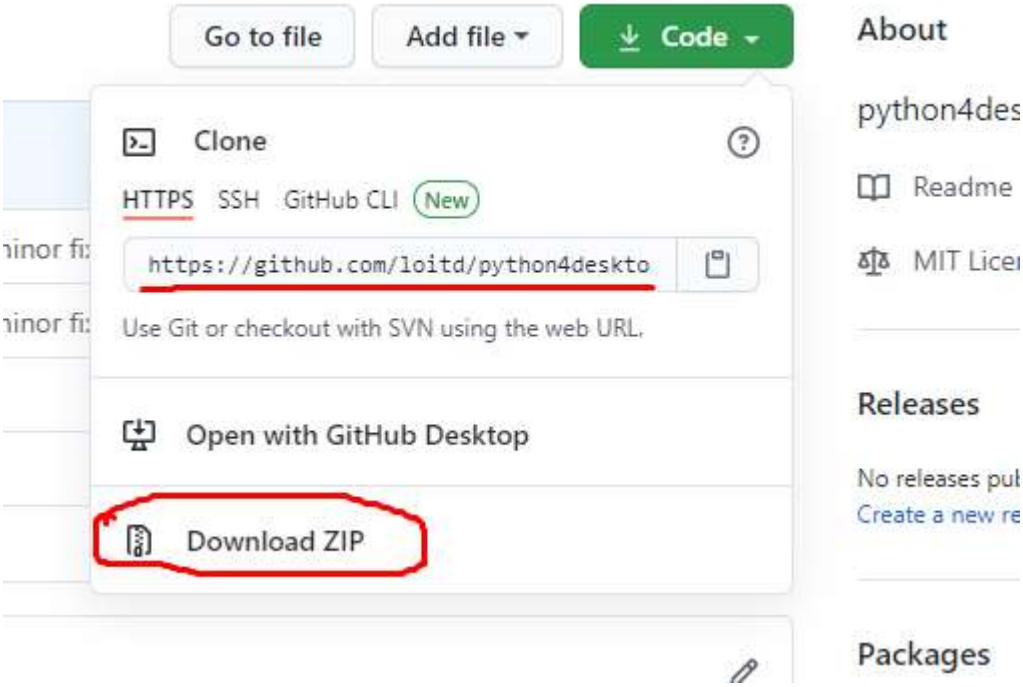

Figure 1. Downloading the source code

This page intentionally left blank

Chapter 1: Introduction

1.1. Introduction

Python, in general, is an easy-to-learn yet powerful high-level scripting programming language. Python does not clearly fall into the interpreted nor compiled programming language, it has several implementations below:

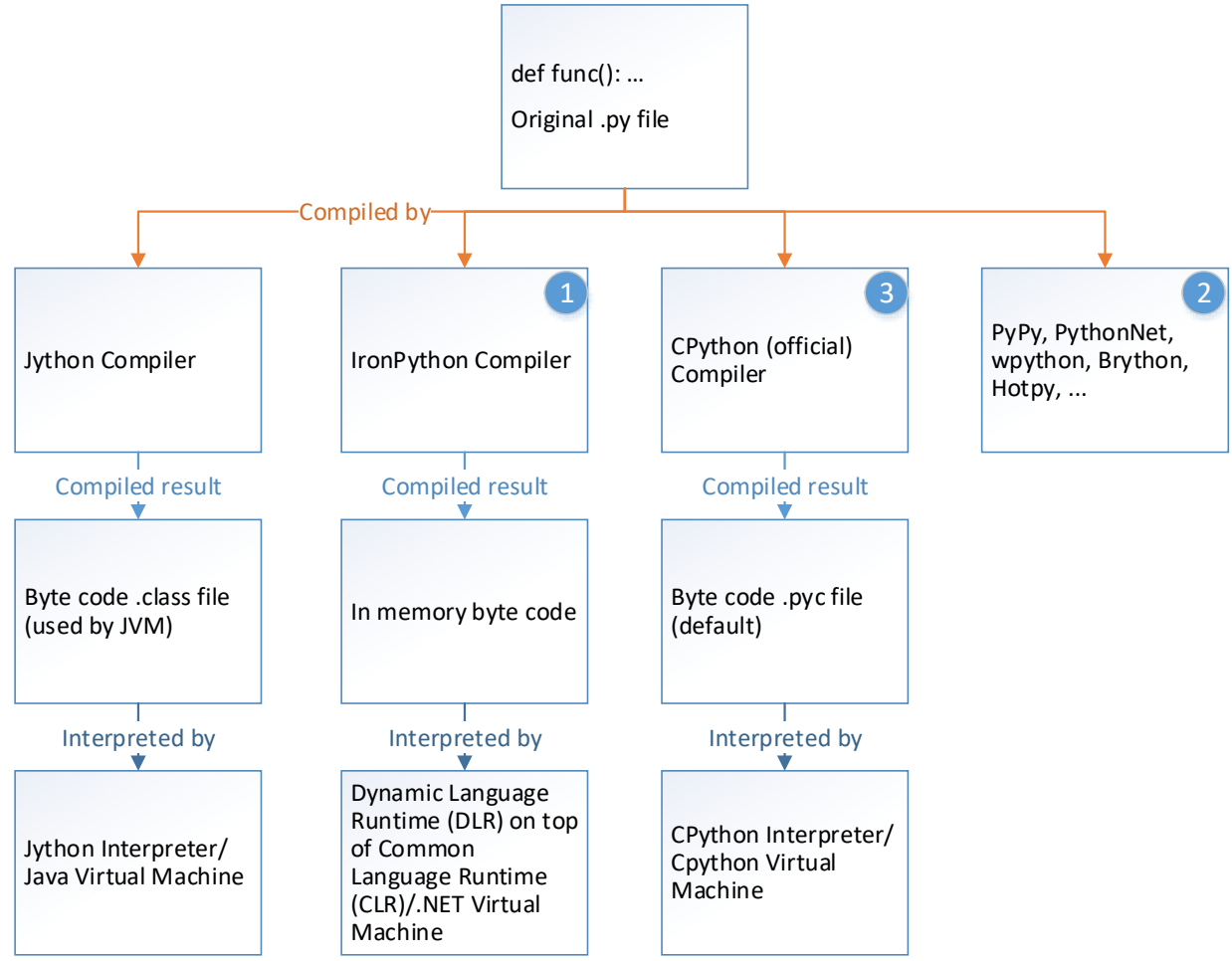

Figure 2. The Python Implementations

Notes from the above illustration:

1. At the time of writing, IronPython supports Python v2.7 and does not officially support Python 3.

2. PyPi (Package Python Index) is a Python repository, should not be confused with PyPy. At the time of writing, PyPy v7.3.1 supports Python 2.7 and 3.6.

   As PyPy wrote on their home page, PyPy is 4.2 times faster than CPython.

3 This is the official default implementation of Python. This book, without explicitly mentioned, follows this implementation.

PIP

pip is a package installer for Python. You can use **pip** to install packages from the Python Package Index and other indexes.

By default, **pip** is shipped with Python installer so you don't have to install **pip** manually.

You can check your current pip version with the **pip --version** command:

```
PS L:\> cd .\python4desktop\chapter01\
PS L:\python4desktop\chapter01> .\venv\Scripts\activate
(venv) PS L:\python4desktop\chapter01> pip --version
pip 20.1.1 from f:\code\src\github.com\loitd\python4desktop\chapter01\venv\li
b\site-packages\pip (python 3.7)
(venv) PS L:\python4desktop\chapter01>
```

Please note that **pip** will be created per virtual environment as you can see in the output above.

Wheels

Wheels are new standards of Python distribution. Please note that wheels are supported from **pip** >= 1.4 and **setuptools** >= 0.8.

Wheels end with **.whl** extension. You can easily find wheels at any project's download tab as below:

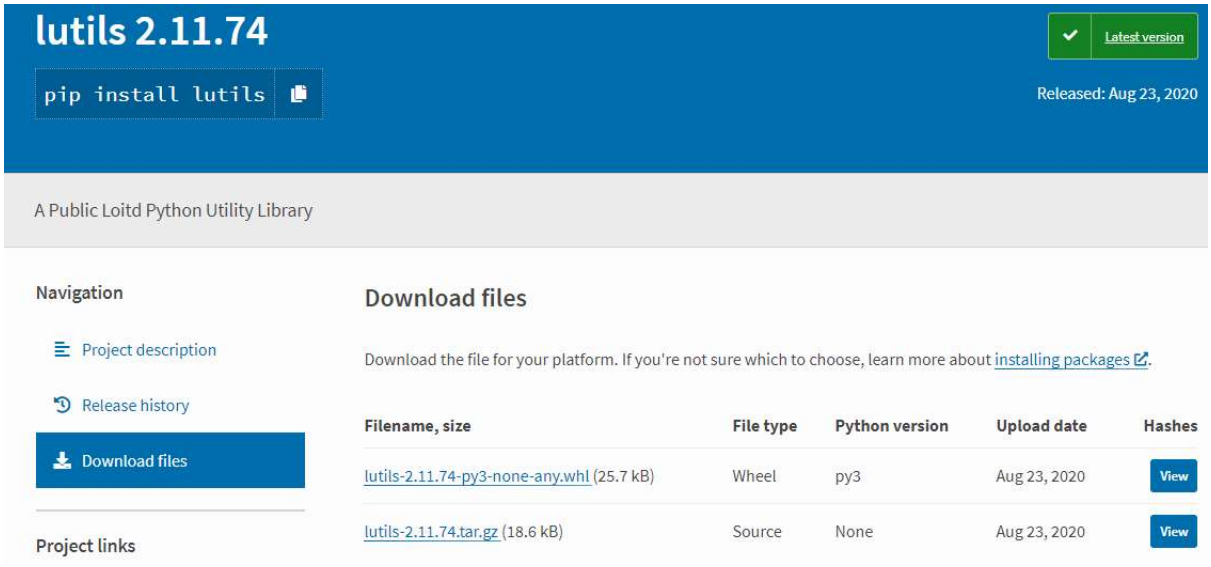

Figure 3. A Python wheel example

Once you have wheels downloaded, you can install the package locally using:

```
pip install file-name-of-the-wheel.whl
```

Wheels are extremely helpful for some kinds of packages those don't have any wheels available at pypi.org. There is my experience with wheels at Python and choosing the right version in section 1.2 in this chapter.

Virtual Environment

Python virtual environment is a self-contained directory that contains a Python installation for a particular version of Python and a number of additional packages.

In this book, you will use a separated virtual environment for each chapter. E.g. chapter 01 will have its own virtual environment inside **`chapter01`** directory.

Python has a built-in virtual environment package named **`venv`** to allow you to create a virtual environment:

```
pip -m venv venv
```

The above command creates a virtual environment into a folder inside the current folder named **venv**.

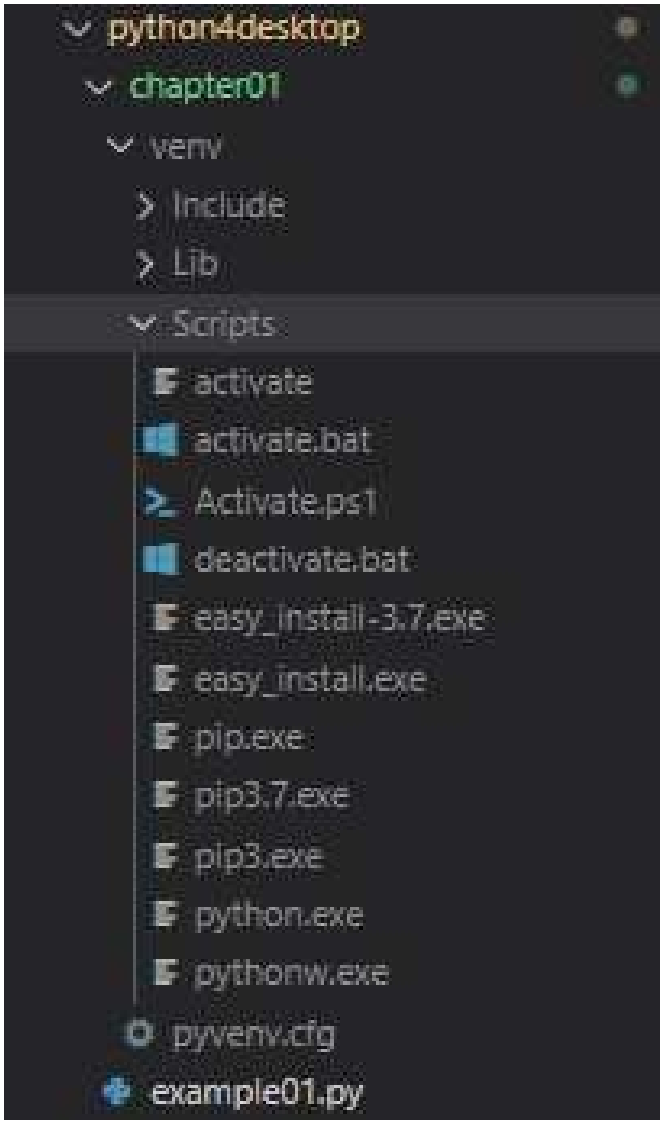

Figure 4. A Python virtual environment structure in Windows

To use the newly created virtual environment, you have to activate it using **`activate`** command.

```
PS L:\> cd .\python4desktop\chapter01\
PS L:\python4desktop\chapter01> .\venv\Scripts\activate
```

After activating process is completed, there will be a prefix **(venv)** at the beginning of the command prompt to let you know that you are in a virtual environment.

To deactivate an active virtual environment, you need to use the **deactivate** command.

```
(venv) L:\python4desktop\chapter01>venv\Scripts\deactivate.bat
L:\python4desktop\chapter01>
```

To install a package inside a virtual environment, you can use the **pip** command as normal. The packages will be installed at the **venv/Lib** directory.

To delete a virtual environment, you simply delete the created virtual environment folder (in this case the **venv** folder).

Please note that:

- The Python built-in **venv** package does NOT allow you to specify other Python versions. To overcome this, you need to specify the full path to Python executable file when using the creation command. E.g.
    - **C:\Python36-86\python -m venv venv86**
    - **C:\Python37-64\python -m venv venv64**
- The **deactivate.bat** currently not working in Windows PowerShell.
- These above commands are used in Windows. In Linux/macOS, you have to use the **source venv/bin/activate** command instead.
- The structure of the **venv** folder is different between the Windows and the Unix/macOS operating systems.

GIL

GIL stands for **Global Interpreter Lock**, in **CPython**, is a mutex that protects access to Python objects, preventing multiple threads from executing Python

bytecodes at once. This lock exists mainly because **`CPython`** memory management is not thread-safe.

Because of **`GIL`**, Python has a bottleneck and can't take full advantage of multiprocessor systems.

Because of **`CPython`**'s GIL, several implementations have been introduced to remove GIL. Currently, **`Jython`** and **`IronPython`** are two implements that have no GIL.

Please take a look back at the beginning of this chapter for Python implementations.

CLI and GUI

Computers can display information and let the user give commands to it using two methods: a command-line interface (CLI) or a graphical user interface (GUI).

In a command-line interface or console, users types command using the keyboard to tell the computer to take an action.

In a graphical user interface, users can use the computer mouse to click on buttons.

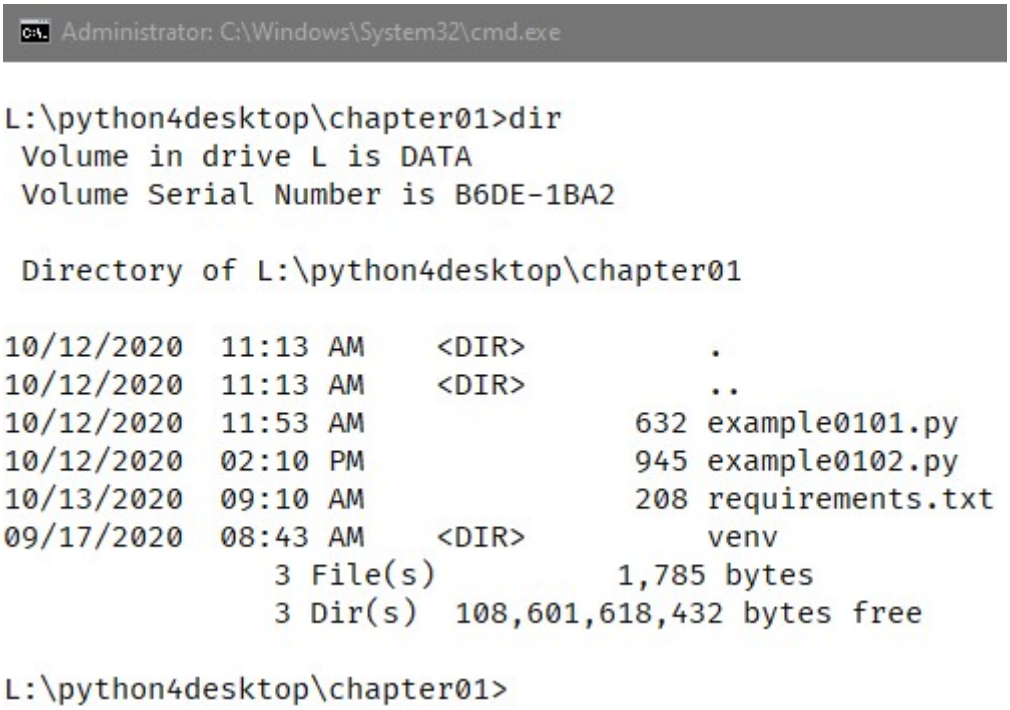

Figure 5. A console or CLI example

Figure 6. A GUI example

The above 2 figures are examples of CLI/Console and GUI applications.

Anaconda and Miniconda

`Anaconda`, named after a giant snake species, is a distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment.

`Miniconda` is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib, and a few others.

Anaconda and **Miniconda** are used heavily in machine learning. They have number of most commonly used libraries automatically installed or can be installed within just one short command.

Unfortunately, they use a specific way to install packages using the **conda install** command and they have their own repositories separated to the famous **pypi.org**. That's mean some of the examples in this book can't be run under **conda** environment because we used some libraries which are not available at the **conda**'s repositories yet (lutils...).

To sum up, in order to run examples in this book under the **conda** environment, you may need to modify the code yourself.