Beginning Python

From Novice to Professional,
Second Edition

Magnus Lie Hetland

Apress®

Beginning Python: From Novice to Professional, Second Edition

Copyright © 2008 by Magnus Lie Hetland

ISBN-13 (paperback): 978-1-59059-982-2

ISBN-13 (electronic): 978-1-4302-0634-7

Printed and bound in the United States of America 9 8 7 6 5 4 3 2

Lead Editor: Frank Pohlmann
Technical Reviewers: Gregg Bolinger, Richard Taylor
Editorial Board: Clay Andres, Steve Anglin, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Matthew Moodie, Joseph Ottinger, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Richard Dal Porto
Copy Editor: Marilyn Smith
Associate Production Director: Kari Brooks-Copony
Production Editor: Liz Berry
Compositor: Pat Christenson
Proofreader: April Eddy
Indexer: John Collin
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Contents at a Glance

Contents

About the Author

MAGNUS LIE HETLAND is an associate professor of algorithms at the Norwegian University of Science and Technology (NTNU). Even though he loves learning new programming languages—even quite obscure ones—Magnus has been a devoted Python fan and an active member of the Python community for many years, and is the author of the popular online tutorials "Instant Python" and "Instant Hacking." His publications include the forerunner to this book, *Practical Python* (Apress, 2002), as well as several scientific papers. When he isn't busy staring at a computer screen, he may be found reading (even while bicycling), acting (in a local theater group), or gaming (mostly role-playing games).

About the Technical Reviewer

RICHARD TAYLOR is a senior analyst at QinetiQ Ltd in the UK, where he specializes in open systems architectures for command and control systems. He has been developing in Python since about 1994, and has used Python to build many large-scale commercial and research applications. When not working, Richard indulges his keen interest in genealogy and open source software, and is a regular contributor to the GRAMPS (Genealogical Research and Analysis Management Programming System) project.

Preface

Here it is—a shiny new edition of *Beginning Python*. If you count its predecessor, *Practical Python*, this is actually the third edition, and a book I've been involved with for the better part of a decade. During this time, Python has seen many interesting changes, and I've done my best to update my introduction to the language. At the moment, Python is facing perhaps its most marked transition in a very long time: the introduction of version 3. As I write this, the final release isn't out yet, but the features are clearly defined and working versions are available. One interesting challenge linked to this language revision is that it isn't backward-compatible. In other words, it doesn't simply add features that I could pick and choose from in my writing. It also changes the existing language, so that certain things that are true for Python 2.5 no longer hold.

Had it been clear that the entire Python community would instantly switch to the new version and update all its legacy code, this would hardly be a problem. Simply describe the new language! However, a lot of code written for older versions exists, and much will probably still be written, until version 3 is universally accepted as The Way To Go™.

So, how have I gotten myself out of this pickle? First of all, even though there are incompatible changes, *most* of the language remains the same. Therefore, if I wrote entirely about Python 2.5, it would be *mostly* correct for Python 3 (and even more so for its companion release, 2.6). As for the parts that will no longer be correct, I have been a bit conservative and assumed that full adoption of version 3 will take some time. I have based the book primarily on 2.5, and noted things that will change throughout the text. In addition, I've included Appendix D, which gives you an overview of the main changes. I think this will work out for most readers.

In writing this second edition, I have had a lot of help from several people. Just as with the previous two versions (the first edition, and, before it, *Practical Python*), Jason Gilmore got me started and played an important role in getting the project on the road. As it has moved along, Richard Dal Porto, Frank Pohlmann, and Dominic Shakeshaft have been instrumental in keeping it going. Richard Taylor has certainly played a crucial role in ensuring that the code is correct (and if it still isn't, I'm the one to blame), and Marilyn Smith has done a great job tuning my writing. My thanks also go out to other Apress staff, including Liz Berry, Beth Christmas, Steve Anglin, and Tina Nielsen, as well as various readers who have provided errata and helpful suggestions, including Bob Helmbold and Waclaw Kusnierczyk. I am also, of course, still thankful to all those who helped in getting the first two incarnations of this book on the shelves.

Preface to the First Edition

A few years ago, Jason Gilmore approached me about writing a book for Apress. He had read my online Python tutorials and wanted me to write a book in a similar style. I was flattered, excited, and just a little nervous. The one thing that worried me the most was how much time it would take, and how much it would interfere with my studies (I was a Ph.D student at the time). It turned out to be quite an undertaking, and it took me a lot longer to finish than I had expected.

Luckily, it didn't interfere too much with my school work, and I managed to get my degree without any delays.

Last year, Jason contacted me again. Apress wanted an expanded and revised version of my book. Was I interested? At the time, I was busy settling into a new position as associate processor, while spending all my spare time portraying Peer Gynt, so again time became the major issue. Eventually (after things had settled down a bit, and I had a bit more time to spare), I agreed to do the book, and this (as I'm sure you've gathered) is the result. Most of the material is taken from the first version of the book, *Practical Python* (Apress, 2002). The existing material has been completely revised, based on recent changes in the Python language, and several new chapters have been added. Some of the old material has also been redistributed to accommodate the new structure. I've received a lot of positive feedback from readers about the first version. I hope I've been able to keep what people liked and to add more of the same.

Without the persistent help and encouragement from several people, this book would never have been written. My heartfelt thanks go out to all of them. In particular, I would like to thank the team that has worked directly with me in the process of writing the book: Jason Gilmore, for getting the project off the ground and steering it in the right direction; Beckie Stones, for keeping everything together; Jeremy Jones and Matt Moodie for their technical comments and insights; and Linda Marousek for being so patient with me. I'm also grateful to the rest of the team for making the process as smooth as it has been. But this book wouldn't be what it is without several people who worked with me on the previous version: I'd like to thank Jason Gilmore and Alex Martelli for their excellent technical editing (Jason on the entire book, and Alex on the first half) and for going above and beyond the call of duty in dispensing advice and suggestions; Erin Mulligan and Tory McLearn for holding my hand through the process and for nudging me along when that was needed; Nancy Rapoport for her help polishing my prose; and Grace Wong for providing answers when no one else could. Pete Shinners gave me several helpful suggestions on the game in Project 10, for which I am very grateful. My morale has also been heavily boosted by several encouraging emails from satisfied readers—thanks! Finally, I would like to thank my family and friends, and my girlfriend Ranveig, for putting up with me while I was writing this book.

Introduction

A C program is like a fast dance on a newly waxed dance floor by people carrying razors.

—Waldi Ravens

C++: Hard to learn and built to stay that way.

—Anonymous

Java is, in many ways, C++– –.

—Michael Feldman

And now for something completely different . . .

—Monty Python's Flying Circus

I've started this introduction with a few quotes to set the tone for the book, which is rather informal. In the hope of making it an easy read, I've tried to approach the topic of Python programming with a healthy dose of humor, and true to the traditions of the Python community, much of this humor is related to Monty Python sketches. As a consequence, some of my examples may seem a bit silly; I hope you will bear with me. (And, yes, the name Python is derived from Monty Python, not from snakes belonging to the family *Pythonidae*.)

In this introduction, I give you a quick look at what Python is, why you should use it, who uses it, who this book's intended audience is, and how the book is organized.

So, what is Python, and why should you use it? To quote an official blurb (available from `http://python.org/doc/essays/blurb.html`), it is "an interpreted, object-oriented, high-level programming language with dynamic semantics." Many of these terms will become clear as you read this book, but the gist is that Python is a programming language that knows how to stay out of your way when you write your programs. It enables you to implement the functionality you want without any hassle, and lets you write programs that are clear and readable (much more so than programs in most other currently popular programming languages).

Even though Python might not be as fast as compiled languages such as C or C++, what you save in programming time will probably be worth using it, and in most programs, the speed difference won't be noticeable anyway. If you are a C programmer, you can easily implement the critical parts of your program in C at a later date, and have them interoperate with the Python parts. If you haven't done any programming before (and perhaps are a bit confused by my references to C and C++), Python's combination of simplicity and power makes it an ideal choice as a place to start.

So, who uses Python? Since Guido van Rossum created the language in the early 1990s, its following has grown steadily, and interest has increased markedly in the past few years. Python is used extensively for system administration tasks (it is, for example, a vital component of several Linux distributions), but it is also used to teach programming to complete beginners. The US National Aeronautics and Space Administration (NASA) uses Python both for development and as a scripting language in several of its systems. Industrial Light & Magic uses Python in its production of special effects for large-budget feature films. Yahoo! uses it (among other things) to manage its discussion groups. Google has used it to implement many components of its web crawler and search engine. Python is being used in such diverse areas as computer games and bioinformatics. Soon one might as well ask, "Who *isn't* using Python?"

This book is for those of you who want to learn how to program in Python. It is intended to suit a wide audience, from neophyte programmer to advanced computer wiz. If you have never programmed before, you should start by reading Chapter 1 and continue until you find that things get too advanced for you (if, indeed, they do). Then you should start practicing and write some programs of your own. When the time is right, you can return to the book and proceed with the more intricate stuff.

If you already know how to program, some of the introductory material might not be new to you (although there will probably be some surprising details here and there). You could skim through the early chapters to get an idea of how Python works, or perhaps read through Appendix A, which is based on my online Python tutorial "Instant Python." It will get you up to speed on the most important Python concepts. After getting the big picture, you could jump straight to Chapter 10 (which describes the Python standard libraries).

The last ten chapters present ten programming projects, which show off various capabilities of the Python language. These projects should be of interest to beginners and experts alike. Although some of the material in the later projects may be a bit difficult for an inexperienced programmer, following the projects in order (after reading the material in the first part of the book) should be possible.

The projects touch upon a wide range of topics, most of which will be very useful to you when writing programs of your own. You will learn how to do things that may seem completely out of reach to you at this point, such as creating a chat server, a peer-to-peer file sharing system, or a full-fledged graphical computer game. Although much of the material may seem hard at first glance, I think you will be surprised by how easy most of it really is. If you would like to download the source code, it's available from the Source Code/Download section of the Apress web site (`http://www.apress.com`).

Well, that's it. I always find long introductions boring myself, so I'll let you continue with your Pythoneering, either in Chapter 1 or in Appendix A. Good luck, and happy hacking.

CHAPTER 1

Instant Hacking: The Basics

It's time to start hacking.[1] In this chapter, you learn how to take control of your computer by speaking a language it understands: Python. Nothing here is particularly difficult, so if you know the basic principles of how your computer works, you should be able to follow the examples and try them out yourself. I'll go through the basics, startiwng with the excruciatingly simple, but because Python is such a powerful language, you'll soon be able to do pretty advanced things.

First, I show you how to get the software you need. Then I tell you a bit about algorithms and their main components. Throughout these sections, there are numerous small examples (most of them using only simple arithmetic) that you can try out in the Python interactive interpreter (covered in the section "The Interactive Interpreter" in this chapter). You learn about variables, functions, and modules, and after handling these topics, I show you how to write and run larger programs. Finally, I deal with strings, an important aspect of almost any Python program.

Installing Python

Before you can start programming, you need some new software. What follows is a short description of how to download and install Python. If you want to jump into the installation process without detailed guidance, you can simply visit `http://www.python.org/download` to get the most recent version of Python.

Windows

To install Python on a Windows machine, follow these steps:

1. Open a web browser and go to `http://www.python.org`.
2. Click the Download link.
3. You should see several links here, with names such as Python 2.5.*x* and Python 2.5.*x* Windows installer. Click the Windows installer link to download the installer file. (If you're running on an Itanium or AMD machine, you need to choose the appropriate installer.)

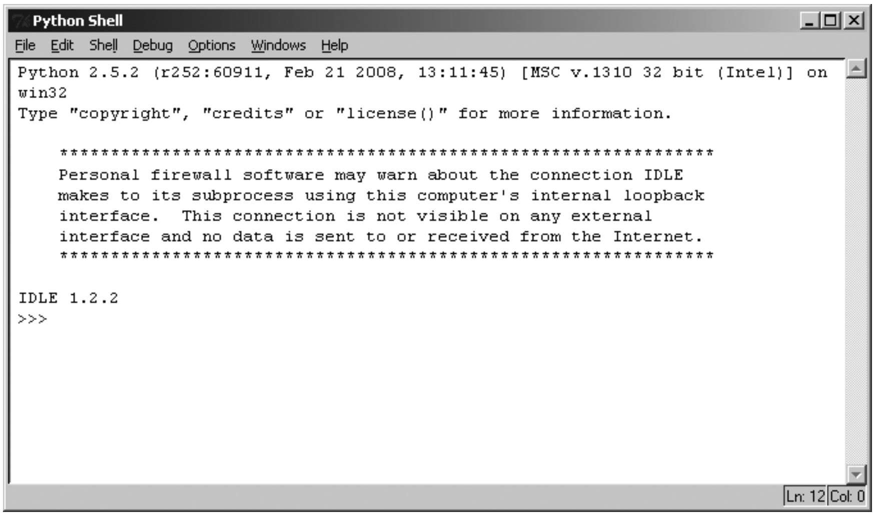
1. *Hacking* is not the same as *cracking*, which is a term describing computer crime. The two are often confused. Hacking basically means "having fun while programming." For more information, see Eric Raymond's article "How to Become a Hacker" at `http://www.catb.org/~esr/faqs/hacker-howto.html`.

Note If you can't find the link mentioned in step 3, click the link with the highest version among those with names like Python 2.5.*x*. For Python 2.5, you could simply go to `http://www.python.org/2.5`. Follow the instructions for Windows users. This will entail downloading a file called `python-2.5.x.msi` (or something similar), where 2.5.*x* should be the version number of the newest release.

4. Store the Windows Installer file somewhere on your computer, such as `C:\download\python-2.5.x.msi`. (Just create a directory where you can find it later.)
5. Run the downloaded file by double-clicking it in Windows Explorer. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the installation is finished, and you're ready to roll!

Assuming that the installation went well, you now have a new program in your Windows Start menu. Run the Python Integrated Development Environment (IDLE) by selecting Start ➤ Programs ➤ Python[2] ➤ IDLE (Python GUI).

You should now see a window that looks like the one shown in Figure 1-1. If you feel a bit lost, simply select Help ➤ IDLE Help from the menu to get a simple description of the various menu items and basic usage. For more documentation on IDLE, check out `http://www.python.org/idle`. (Here you will also find more information on running IDLE on platforms other than Windows.) If you press F1, or select Help ➤ Python Docs from the menu, you will get the full Python documentation. (The document there of most use to you will probably be the Library Reference.) All the documentation is searchable.



```
Python 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2.2
>>>
```

Figure 1-1. *The IDLE interactive Python shell*

2. This menu option will probably include your version number, as in Python 2.5.

Once you have the IDLE interactive Python shell running, you can continue with the section "The Interactive Interpreter," later in this chapter.

WINDOWS INSTALLER

Python for Microsoft Windows is distributed as a Windows Installer file, and requires that your Windows version supports Windows Installer 2.0 (or later). If you don't have Windows Installer, it can be downloaded freely for Windows 95, 98, ME, NT 4.0, and 2000. Windows XP and later versions of Windows already have Windows Installer, and many older machines will, too. There are download instructions for the Installer on the Python download page.

Alternatively, you could go to the Microsoft download site, `http://www.microsoft.com/downloads`, and search for "Windows Installer" (or simply select it from the download menu). Choose the most recent version for your platform and follow the download and installation instructions.

If you're uncertain about whether you have Windows Installer, simply try executing step 5 of the previous installation instructions: double-click the MSI file. If you get the install wizard, everything is okay. See `http://www.python.org/2.5/msi.html` for advanced features of the Windows Installer related to Python installation.

Linux and UNIX

In most Linux and UNIX installations (including Mac OS X), a Python interpreter will already be present. You can check whether this is the case for you by running the `python` command at the prompt, as follows:

```
$ python
```

Running this command should start the interactive Python interpreter, with output similar to the following:

```
Python 2.5.1 (r251:54869, Apr 18 2007, 22:08:04)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

■ **Note** To exit the interactive interpreter, use Ctrl-D (press the Ctrl key and while keeping that depressed, press D).

If there is no Python interpreter installed, you will probably get an error message similar to the following:

```
bash: python: command not found
```

In that case, you need to install Python yourself, as described in the following sections.

Using a Package Manager

Several package systems and installation mechanisms exist for Linux. If you're running a Linux system with some form of package manager, chances are you can get Python through it.

■ **Note** You will probably need to have administrator privileges (a root account) in order to install Python using a package manager in Linux.

For example, if you're running Debian Linux, you should be able to install Python with the following command:

```
$ apt-get install python
```

If you're running Gentoo Linux, you should be able to use Portage, like this:

```
$ emerge python
```

In both cases, `$` is, of course, the bash prompt.

■ **Note** Many other package managers out there have automatic download capabilities, including Yum, Synaptic (specific to Ubuntu Linux), and other Debian-style managers. You should probably be able to get recent versions of Python through these.

Compiling from Sources

If you don't have a package manager, or would rather not use it, you can compile Python yourself. This may be the method of choice if you are on a UNIX box but you don't have root access (installation privileges). This method is very flexible, and enables you to install Python wherever you want, including in your own home directory. To compile and install Python, follow these steps:

1. Go to the download page (refer to steps 1 and 2 in the instructions for installing Python on a Windows system).
2. Follow the instructions for downloading the sources.
3. Download the file with the extension `.tgz`. Store it in a temporary location. Assuming that you want to install Python in your home directory, you may want to put it in a directory such as `~/python`. Enter this directory (e.g., using `cd ~/python`).
4. Unpack the archive with the command `tar -xzvf Python-2.5.tgz` (where `2.5` is the version number of the downloaded source code). If your version of `tar` doesn't support the `z` option, you may want to uncompress the archive with `gunzip` first, and then use `tar -xvf` afterward. If there is something wrong with the archive, try downloading it again. Sometimes errors occur during download.