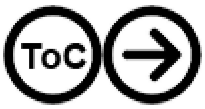

AUTOMATE THE BORING STUFF WITH PYTHON

2ND EDITION

Practical Programming for Total Beginners

by Al Sweigart

San Francisco

ISBN-10: 1-59327-992-2

ISBN-13: 978-1-59327-992-9

Publisher: William Pollock

Production Editor: Laurel Chun

Cover Illustration: Josh Ellingson

Interior Design: Octopod Studios

Developmental Editors: Frances Saux and Jan Cash

Technical Reviewers: Ari Lacenski and Philip James

Copyeditors: Kim Wimpsett, Britt Bogan, and Paula L. Fleming

Compositors: Susan Glinert Stevens and Danielle Foster

Proofreaders: Lisa Devoto Farrell and Emelie Burnette

Indexer: BIM Indexing and Proofreading Services

For information on distribution, translations, or bulk sales,
please contact No Starch Press, Inc. directly:

No Starch Press, Inc.

245 8th Street, San Francisco, CA 94103

phone: 1.415.863.9900; info@nostarch.com

www.nostarch.com

The Library of Congress Control Number for the first edition is: 2014953114

About the Author

Al Sweigart is a software developer and tech book author. Python is his favorite programming language, and he is the developer of several open source modules for it. His other books are freely available under a Creative Commons license on his website *https://inventwithpython.com/*. His cat now weighs 11 pounds.

About the Tech Reviewer

Philip James has been working in Python for over a decade and is a frequent speaker in the Python community. He speaks on topics ranging from Unix fundamentals to open source social networks. Philip is a Core Contributor to the BeeWare project and lives in the San Francisco Bay Area with his partner Nic and her cat River.

INTRODUCTION

"You've just done in two hours what it takes the three of us two days to do." My college roommate was working at a retail electronics store in the early 2000s. Occasionally, the store would receive a spreadsheet of thousands of product prices from other stores. A team of three employees would print the spreadsheet onto a thick stack of paper and split it among themselves. For each product price, they would look up their store's price and note all the products that their competitors sold for less. It usually took a couple of days.

"You know, I could write a program to do that if you have the original file for the printouts," my roommate told them, when he saw them sitting on the floor with papers scattered and stacked all around.

After a couple of hours, he had a short program that read a competitor's price from a file, found the product in the store's database, and noted whether the competitor was cheaper. He was still new to programming, so he spent most of his time looking up documentation in a programming book. The actual program took only a few seconds to run. My roommate and his co-workers took an extra-long lunch that day.

This is the power of computer programming. A computer is like a Swiss Army knife that you can configure for countless tasks. Many people spend hours clicking and typing to perform repetitive tasks, unaware that the machine they're using could do their job in seconds if they gave it the right instructions.

Whom Is This Book For?

Software is at the core of so many of the tools we use today: nearly everyone uses social networks to communicate, many people have internet-connected computers in their phones, and most office jobs involve interacting with a computer to get work done. As a result, the demand for people who can code has skyrocketed. Countless books, interactive web tutorials, and developer boot camps promise to turn ambitious beginners into software engineers with six-figure salaries.

This book is not for those people. It's for everyone else.

On its own, this book won't turn you into a professional software developer any more than a few guitar lessons will turn you into a rock star. But if you're an office worker, administrator, academic, or anyone else who uses a computer for work or fun, you will learn the basics of programming so that you can automate simple tasks such as these:

- Moving and renaming thousands of files and sorting them into folders
- Filling out online forms—no typing required
- Downloading files or copying text from a website whenever it updates
- Having your computer text you custom notifications
- Updating or formatting Excel spreadsheets
- Checking your email and sending out prewritten responses

These tasks are simple but time-consuming for humans, and they're often so trivial or specific that there's no ready-made software to perform them. Armed with a little bit of programming knowledge, however, you can have your computer do these tasks for you.

Conventions

This book is not designed as a reference manual; it's a guide for beginners. The coding style sometimes goes against best practices (for example, some programs use global variables), but that's a trade-off to make the code simpler to learn. This book is made for people to write throwaway code, so there's not much time spent on style and elegance. Sophisticated programming concepts—like object-oriented programming, list comprehensions, and generators—aren't covered because of the complexity they add. Veteran programmers may point out ways the code in this book could be changed to improve efficiency, but this book is mostly concerned with getting programs to work with the least amount of effort on your part.

What Is Programming?

Television shows and films often show programmers furiously typing cryptic streams of 1s and 0s on glowing screens, but modern programming isn't that mysterious.

Programming is simply the act of entering instructions for the computer to perform. These instructions might crunch some numbers, modify text, look up information in files, or communicate with other computers over the internet.

All programs use basic instructions as building blocks. Here are a few of the most common ones, in English:

- "Do this; then do that."
- "If this condition is true, perform this action; otherwise, do that action."
- "Do this action exactly 27 times."
- "Keep doing that until this condition is true."

You can combine these building blocks to implement more intricate decisions, too. For example, here are the programming instructions, called the *source code*, for a simple program written in the Python programming language. Starting at the top, the Python software runs each line of code (some lines are run only *if* a certain condition is true or *else* Python runs some other line) until it reaches the bottom.

```
❶ passwordFile = open('SecretPasswordFile.txt')
❷ secretPassword = passwordFile.read()
❸ print('Enter your password.')
  typedPassword = input()
❹ if typedPassword == secretPassword:
    ❺ print('Access granted')
    ❻ if typedPassword == '12345':
        ❼ print('That password is one that an idiot puts on their luggage.')
  else:
    ❽ print('Access denied')
```

You might not know anything about programming, but you could probably make a reasonable guess at what the previous code does just by reading it. First, the file *SecretPasswordFile.txt* is opened ❶, and the secret password in it is read ❷. Then, the user is prompted to input a password (from the keyboard) ❸. These two passwords are compared ❹, and if they're the same, the program prints *Access granted* to the screen ❺. Next, the program checks to see whether the password is *12345* ❻ and hints that this choice might not be the best for a password ❼. If the passwords are not the same, the program prints *Access denied* to the screen ❽.

What Is Python?

Python is a programming language (with syntax rules for writing what is considered valid Python code) and the Python interpreter software that reads source code (written in the Python language) and performs its instructions. You can download the Python interpreter for free at *https://python.org/*, and there are versions for Linux, macOS, and Windows.

The name Python comes from the surreal British comedy group Monty Python, not from the snake. Python programmers are affectionately called Pythonistas, and both Monty Python and serpentine references usually pepper Python tutorials and documentation.

Programmers Don't Need to Know Much Math

The most common anxiety I hear about learning to program is the notion that it requires a lot of math. Actually, most programming doesn't require math beyond basic arithmetic. In fact, being good at programming isn't that different from being good at solving Sudoku puzzles.

To solve a Sudoku puzzle, the numbers 1 through 9 must be filled in for each row, each column, and each 3×3 interior square of the full 9×9 board. Some numbers are provided to give you a start, and you find a solution by making deductions based on these numbers. In the puzzle shown in Figure 0-1, since 5 appears in the first and second rows, it cannot show up in these rows again. Therefore, in the upper-right grid, it must be in the third row. Since the last column also already has a 5 in it, the 5 cannot go to the right of the 6, so it must go to the left of the 6. Solving one row, column, or square will provide more clues to the rest of the puzzle, and as you fill in one group of numbers 1 to 9 and then another, you'll soon solve the entire grid.

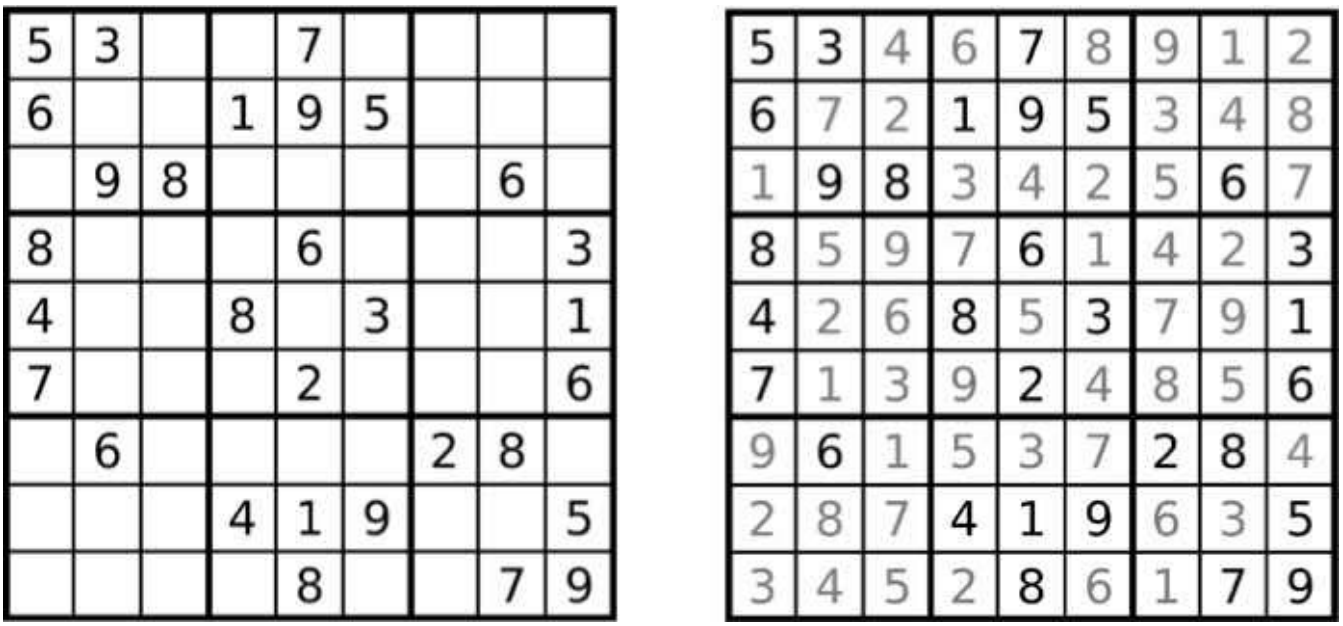

Figure 0-1: A new Sudoku puzzle (left) and its solution (right). Despite using numbers, Sudoku doesn't involve much math. (Images © Wikimedia Commons)

Just because Sudoku involves numbers doesn't mean you have to be good at math to figure out the solution. The same is true of programming. Like solving a Sudoku puzzle, writing programs involves breaking down a problem into individual, detailed steps. Similarly, when *debugging* programs (that is, finding and fixing errors), you'll patiently observe what the program is doing and find the cause of the bugs. And like all skills, the more you program, the better you'll become.

You Are Not Too Old to Learn Programming

The second most common anxiety I hear about learning to program is that people think they're too old to learn it. I read many internet comments from folks who think it's too late for them because they are already (gasp!) 23 years old. This is clearly not "too old" to learn to program: many people learn much later in life.

You don't need to have started as a child to become a capable programmer. But the image of programmers as whiz kids is a persistent one. Unfortunately, I contribute to this myth when I tell others that I was in grade school when I started programming.

However, programming is much easier to learn today than it was in the 1990s. Today, there are more books, better search engines, and many more online question-and-answer websites. On top of that, the programming languages themselves are far more user-friendly. For these reasons, **everything I learned about programming in the years between grade school and high school graduation could be learned today in about a dozen weekends**. My head start wasn't really much of a head start.

It's important to have a "growth mindset" about programming—in other words, understand that people develop programming skills through practice. They aren't just born as programmers, and being unskilled at programming now is not an indication that you can never become an expert.

Programming Is a Creative Activity

Programming is a creative task, like painting, writing, knitting, or constructing LEGO castles. Like painting a blank canvas, making software has many constraints but endless possibilities.

The difference between programming and other creative activities is that when programming, you have all the raw materials you need in your computer; you don't need to buy any additional canvas, paint, film, yarn, LEGO bricks, or electronic components. A decade-old computer is more than powerful enough to write programs. Once your program is written, it can be copied perfectly an infinite number of times. A knit sweater can only be worn by one person at a time, but a useful program can easily be shared online with the entire world.

About This Book

The first part of this book covers basic Python programming concepts, and the second part covers various tasks you can have your computer automate. Each chapter in the second part has project programs for you to study. Here's a brief rundown of what you'll find in each chapter.

Part I: Python Programming Basics

Chapter 1: Python Basics Covers expressions, the most basic type of Python instruction, and how to use the Python interactive shell software to experiment with code.

Chapter 2: Flow Control Explains how to make programs decide which instructions to execute so your code can intelligently respond to different conditions.

Chapter 3: Functions Instructs you on how to define your own functions so that you can organize your code into more manageable chunks.

Chapter 4: Lists Introduces the list data type and explains how to organize data.

Chapter 5: Dictionaries and Structuring Data Introduces the dictionary data type and shows you more powerful ways to organize data.

Chapter 6: Manipulating Strings Covers working with text data (called *strings* in Python).

Part II: Automating Tasks

Chapter 7: Pattern Matching with Regular Expressions Covers how Python can manipulate strings and search for text patterns with regular expressions.

Chapter 8: Input Validation Explains how your program can verify the information a user gives it, ensuring that the user's data arrives in a format that won't cause errors in the rest of the program.

Chapter 9: Reading and Writing Files Explains how your program can read the contents of text files and save information to files on your hard drive.

Chapter 10: Organizing Files Shows how Python can copy, move, rename, and delete large numbers of files much faster than a human user can. Also explains compressing and decompressing files.

Chapter 11: Debugging Shows how to use Python's various bug-finding and bug-fixing tools.

Chapter 12: Web Scraping Shows how to write programs that can automatically download web pages and parse them for information. This is called *web scraping*.

Chapter 13: Working with Excel Spreadsheets Covers programmatically manipulating Excel spreadsheets so that you don't have to read them. This is helpful when the number of documents you have to analyze is in the hundreds or thousands.

Chapter 14: Working with Google Sheets Covers how to read and update Google Sheets, a popular web-based spreadsheet application, using Python.

Chapter 15: Working with PDF and Word Documents Covers programmatically reading Word and PDF documents.

Chapter 16: Working with CSV Files and JSON Data Continues to explain how to programmatically manipulate documents, now discussing CSV and JSON files.

Chapter 17: Keeping Time, Scheduling Tasks, and Launching Programs Explains how Python programs handle time and dates and how to schedule your computer to perform tasks at certain times. Also shows how your Python programs can launch non-Python programs.

Chapter 18: Sending Email and Text Messages Explains how to write programs that can send emails and text messages on your behalf.

Chapter 19: Manipulating Images Explains how to programmatically manipulate images such as JPEG or PNG files.

Chapter 20: Controlling the Keyboard and Mouse with GUI Automation Explains how to programmatically control the mouse and keyboard to automate clicks and keypresses.

Appendix A: Installing Third-Party Modules Shows you how to extend Python with useful additional modules.

Appendix B: Running Programs Shows you how to run your Python programs on Windows, macOS, and Linux from outside of the code editor.

Appendix C: Answers to the Practice Questions Provides answers and some additional context to the practice questions at the end of each chapter.

Downloading and Installing Python

You can download Python for Windows, macOS, and Ubuntu for free at *https://python.org/downloads/*. If you download the latest version from the website's download page, all of the programs in this book should work.

> **WARNING**
>
> *Be sure to download a version of Python 3 (such as 3.8.0). The programs in this book are written to run on Python 3 and may not run correctly, if at all, on Python 2.*

On the download page, you'll find Python installers for 64-bit and 32-bit computers for each operating system, so first figure out which installer you need. If you bought your computer in 2007 or later, it is most likely a 64-bit system. Otherwise, you have a 32-bit version, but here's how to find out for sure:

- On Windows, select **Start ▸ Control Panel ▸ System** and check whether System Type says 64-bit or 32-bit.
- On macOS, go the Apple menu, select **About This Mac ▸ More Info ▸ System Report ▸ Hardware**, and then look at the Processor Name field. If it says Intel Core Solo or Intel Core Duo, you have a 32-bit machine. If it says anything else (including Intel Core 2 Duo), you have a 64-bit machine.
- On Ubuntu Linux, open a Terminal and run the command `uname -m`. A response of `i686` means 32-bit, and `x86_64` means 64-bit.

On Windows, download the Python installer (the filename will end with *.msi*) and double-click it. Follow the instructions the installer displays on the screen to install Python, as listed here:

1. Select **Install for All Users** and click **Next**.
2. Accept the default options for the next several windows by clicking **Next**.

On macOS, download the *.dmg* file that's right for your version of macOS and double-click it. Follow the instructions the installer displays on the screen to install Python, as listed here:

1. When the DMG package opens in a new window, double-click the *Python.mpkg* file. You may have to enter the administrator password.
2. Accept the default options for the next several windows by clicking **Continue** and click **Agree** to accept the license.
3. On the final window, click **Install**.

If you're running Ubuntu, you can install Python from the Terminal by following these steps:

1. Open the Terminal window.
2. Enter `sudo apt-get install python3`.
3. Enter `sudo apt-get install idle3`.
4. Enter `sudo apt-get install python3-pip`.

Downloading and Installing Mu

While the *Python interpreter* is the software that runs your Python programs, the *Mu editor software* is where you'll enter your programs, much the way you type in a word processor. You can download Mu from *https://codewith.mu/*.

On Windows and macOS, download the installer for your operating system and then run it by double-clicking the installer file. If you are on macOS, running the installer opens a window where you must drag the Mu icon to the Applications folder icon to continue the installation. If you are on Ubuntu, you'll need to install Mu as a Python package. In that case, click the Instructions button in the Python Package section of the download page.

Starting Mu

Once it's installed, let's start Mu.

- On Windows 7 or later, click the Start icon in the lower-left corner of your screen, enter `Mu` in the search box, and select it.
- On macOS, open the Finder window, click **Applications**, and then click **mu-editor**.
- On Ubuntu, select **Applications** ▸ **Accessories** ▸ **Terminal** and then enter `python3 -m mu`.

The first time Mu runs, a Select Mode window will appear with options Adafruit CircuitPython, BBC micro:bit, Pygame Zero, and Python 3. Select **Python 3**. You can always change the mode later by clicking the Mode button at the top of the editor window.

> **NOTE**
>
> *You'll need to download Mu version 1.10.0 or later in order to install the third-party modules featured in this book. As of this writing, 1.10.0 is an alpha release and is listed on the download page as a separate link from the main download links.*

Starting IDLE

This book uses Mu as an editor and interactive shell. However, you can use any number of editors for writing Python code. The *Integrated Development and Learning Environment (IDLE)* software installs along with Python, and it can serve as a second editor if for some reason you can't get Mu installed or working. Let's start IDLE now.

- On Windows 7 or later, click the Start icon in the lower-left corner of your screen, enter `IDLE` in the search box, and select **IDLE (Python GUI)**.
- On macOS, open the Finder window, click **Applications**, click **Python 3.8**, and then click the IDLE icon.
- On Ubuntu, select **Applications** ▸ **Accessories** ▸ **Terminal** and then enter `idle3`. (You may also be able to click **Applications** at the top of the screen, select **Programming**, and then click **IDLE 3**.)

The Interactive Shell

When you run Mu, the window that appears is called the *file editor* window. You can open the *interactive shell* by clicking the REPL button. A shell is a program that lets you type instructions into the computer, much like the Terminal or Command Prompt on macOS and Windows, respectively. Python's interactive shell lets you enter instructions for the Python interpreter software to run. The computer reads the instructions you enter and runs them immediately.

In Mu, the interactive shell is a pane in the lower half of the window with the following text:

```
Jupyter QtConsole 4.3.1
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit
(AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

If you run IDLE, the interactive shell is the window that first appears. It should be mostly blank except for text that looks something like this:

```
Python 3.8.0b1 (tags/v3.8.0b1:3b5deb0116, Jun 4 2019, 19:52:55) [MSC v.1916
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

`In [1]:` and `>>>` are called *prompts*. The examples in this book will use the `>>>` prompt for the interactive shell since it's more common. If you run Python from the Terminal or Command Prompt, they'll use the `>>>` prompt, as well. The `In [1]:` prompt was invented by Jupyter Notebook, another popular Python editor.

For example, enter the following into the interactive shell next to the prompt:

```
>>> print('Hello, world!')
```

After you type that line and press ENTER, the interactive shell should display this in response:

```
>>> print('Hello, world!')
Hello, world!
```

You've just given the computer an instruction, and it did what you told it to do!

Installing Third-Party Modules

Some Python code requires your program to import modules. Some of these modules come with Python, but others are third-party modules created by developers outside of the Python core dev team. Appendix A has detailed instructions on how to use the `pip` program (on Windows) or `pip3` program (on macOS and Linux) to install third-party modules. Consult Appendix A when this book instructs you to install a particular third-party module.

How to Find Help

Programmers tend to learn by searching the internet for answers to their questions. This is quite different from the way many people are accustomed to learning—through an in-person teacher who lectures and can answer questions. What's great about using the internet as a schoolroom is that there are whole communities of folks who can answer your questions. Indeed, your questions have probably already been answered, and the answers are waiting online for you to find them. If you encounter an error message or have trouble making your code work, you won't be the first person to have your problem, and finding a solution is easier than you might think.

For example, let's cause an error on purpose: enter `'42' + 3` into the interactive shell. You don't need to know what this instruction means right now, but the result should look like this:

```
>>> '42' + 3
❶ Traceback (most recent call last):
    File "<pyshell#0>", line 1, in <module>
      '42' + 3
❷ TypeError: Can't convert 'int' object to str implicitly
>>>
```

The error message ❷ appears because Python couldn't understand your instruction. The traceback part ❶ of the error message shows the specific instruction and line number that Python had trouble with. If you're not sure what to make of a particular error message, search for it online. Enter **"TypeError: Can't convert 'int' object to str implicitly"** (including the quotes) into your favorite search engine, and you should see tons of links explaining what the error message means and what causes it, as shown in Figure 0-2.

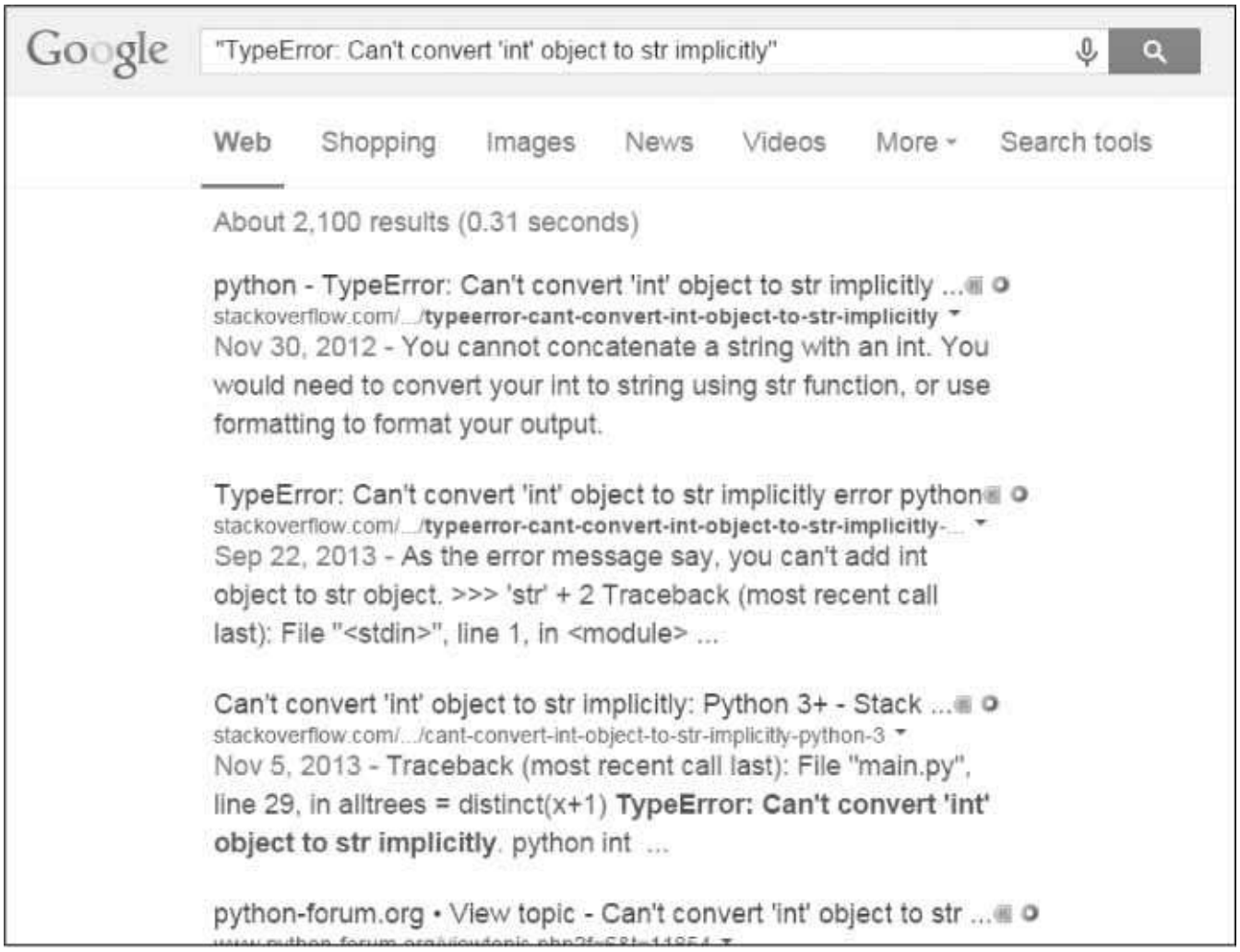

Figure 0-2: The Google results for an error message can be very helpful.

You'll often find that someone else had the same question as you and that some other helpful person has already answered it. No one person can know everything about programming, so an everyday part of any software developer's job is looking up answers to technical questions.

Asking Smart Programming Questions

If you can't find the answer by searching online, try asking people in a web forum such as Stack Overflow (*https://stackoverflow.com/*) or the "learn programming" subreddit at *https://reddit.com/r/learnprogramming/*. But keep in mind there are smart ways to ask

programming questions that help others help you. To begin with, be sure to read the FAQ sections at these websites about the proper way to post questions.

When asking programming questions, remember to do the following:

- Explain what you are trying to do, not just what you did. This lets your helper know if you are on the wrong track.
- Specify the point at which the error happens. Does it occur at the very start of the program or only after you do a certain action?
- Copy and paste the *entire* error message and your code to *https://pastebin.com/* or *https://gist.github.com/*.

  These websites make it easy to share large amounts of code with people online, without losing any text formatting. You can then put the URL of the posted code in your email or forum post. For example, here some pieces of code I've posted: *https://pastebin.com/SzP2DbFx/* and *https://gist.github.com/asweigart/6912168/*.
- Explain what you've already tried to do to solve your problem. This tells people you've already put in some work to figure things out on your own.
- List the version of Python you're using. (There are some key differences between version 2 Python interpreters and version 3 Python interpreters.) Also, say which operating system and version you're running.
- If the error came up after you made a change to your code, explain exactly what you changed.
- Say whether you're able to reproduce the error every time you run the program or whether it happens only after you perform certain actions. If the latter, then explain what those actions are.

Always follow good online etiquette as well. For example, don't post your questions in all caps or make unreasonable demands of the people trying to help you.

You can find more information on how to ask for programming help in the blog post at *https://autbor.com/help/*. You can find a list of frequently asked questions about programming at *https://www.reddit.com/r/learnprogramming/wiki/faq/* and a similar list about getting a job in software development at *https://www.reddit.com/r/cscareerquestions/wiki/index/*.

I love helping people discover Python. I write programming tutorials on my blog at *https://inventwithpython.com/blog/*, and you can contact me with questions at *al@inventwithpython.com*. Although, you may get a faster response by posting your questions to *https://reddit.com/r/inventwithpython/*.

SUMMARY

For most people, their computer is just an appliance instead of a tool. But by learning how to program, you'll gain access to one of the most powerful tools of the modern world, and you'll have fun along the way. Programming isn't brain surgery—it's fine for amateurs to experiment and make mistakes.

This book assumes you have zero programming knowledge and will teach you quite a bit, but you may have questions beyond its scope. Remember that asking effective questions and knowing how to find answers are invaluable tools on your programming journey.

Let's begin!