Chapter 0 – Introduction

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 its competitor. 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 around them.

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, and 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 the following:

- Moving and renaming thousands of files and sorting them into folders
- Filling out online forms, no typing required
- Downloading files or copy 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, 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.

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 that number of 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 refers to the Python 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. The Python interpreter is free to download from *http://python.org/*, and there are versions for Linux, OS X, 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 that people think 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. You find a solution by applying deduction and logic from the starting numbers. For example, since 5 appears in the top left of the Sudoku puzzle shown in Figure 1, it cannot appear elsewhere in the top row, in the leftmost column, or in the top-left 3×3 square. Solving one row, column, or square at a time will provide more number clues for the rest of the puzzle.

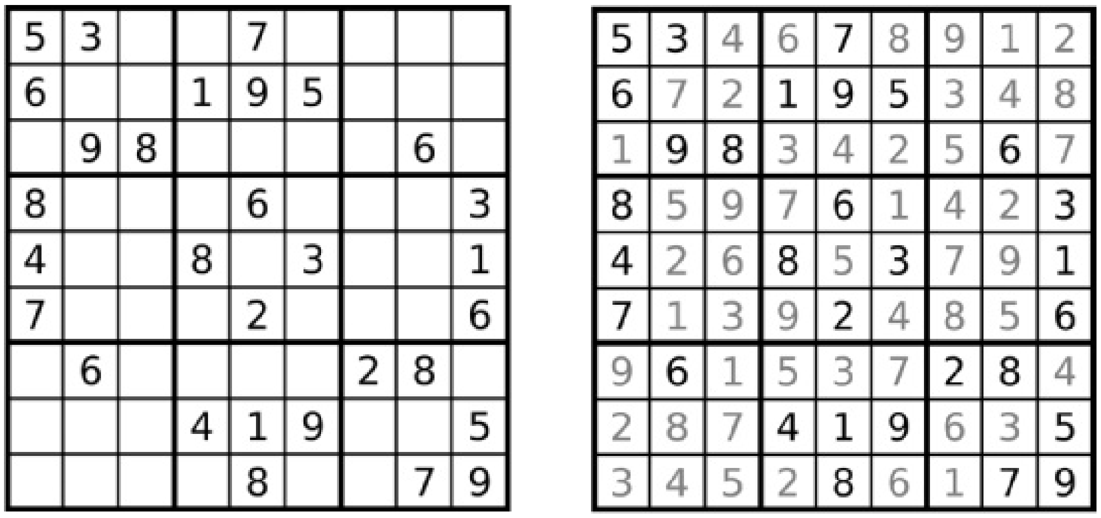


Figure 1. A new Sudoku puzzle (left) and its solution (right). Despite using numbers, Sudoku doesn't involve much math. (Images © Wikimedia Commons)

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.

Programming Is a Creative Activity

Programming is a creative task, somewhat like constructing a castle out of LEGO bricks. You start with a basic idea of what you want your castle to look like and inventory your available blocks. Then you start building. Once you've finished building your program, you can pretty up your code just like you would your castle.

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. When your program is written, it can easily be shared online with the entire world. And though you'll make mistakes when programming, the activity is still a lot of fun.

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

- **Chapter 1**. Covers expressions, the most basic type of Python instruction, and how to use the Python interactive shell software to experiment with code.
- **Chapter 2**. Explains how to make programs decide which instructions to execute so your code can intelligently respond to different conditions.
- **Chapter 3**. Instructs you on how to define your own functions so that you can organize your code into more manageable chunks.
- **Chapter 4**. Introduces the list data type and explains how to organize data.
- **Chapter 5**. Introduces the dictionary data type and shows you more powerful ways to organize data.
- **Chapter 6**. Covers working with text data (called *strings* in Python).

Part II

- **Chapter 7**. Covers how Python can manipulate strings and search for text patterns with regular expressions.
- **Chapter 8**. Explains how your programs can read the contents of text files and save information to files on your hard drive.
- **Chapter 9**. Shows how Python can copy, move, rename, and delete large numbers of files much faster than a human user can. It also explains

compressing and decompressing files.

- **Chapter 10**. Shows how to use Python's various bug-finding and bug-fixing tools.
- **Chapter 11**. Shows how to write programs that can automatically download web pages and parse them for information. This is called *web scraping*.
- **Chapter 12**. 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 13**. Covers programmatically reading Word and PDF documents.
- **Chapter 14**. Continues to explain how to programmatically manipulate documents with CSV and JSON files.
- **Chapter 15**. Explains how time and dates are handled by Python programs and how to schedule your computer to perform tasks at certain times. This chapter also shows how your Python programs can launch non-Python programs.
- **Chapter 16**. Explains how to write programs that can send emails and text messages on your behalf.
- **Chapter 17**. Explains how to programmatically manipulate images such as JPEG or PNG files.
- **Chapter 18**. Explains how to programmatically control the mouse and keyboard to automate clicks and keypresses.

Downloading and Installing Python

You can download Python for Windows, OS X, and Ubuntu for free from *http://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 3.4.0). The programs in this book are written to run on Python 3 and may not run correctly, if at all, on Python 2.

You'll find Python installers for 64-bit and 32-bit computers for each operating system on the download page, 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 OS X, 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 then click **Next**.
2. Install to the *C:\Python34* folder by clicking **Next**.
3. Click **Next** again to skip the Customize Python section.

On Mac OS X, download the *.dmg* file that's right for your version of OS X 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. Click **Continue** through the Welcome section and click **Agree** to accept the license.
3. Select **HD Macintosh** (or whatever name your hard drive has) and 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`.

Starting IDLE

While the *Python interpreter* is the software that runs your Python programs, the *interactive development environment (IDLE)* software is where you'll enter your programs, much like a word processor. Let's start IDLE now.

- On Windows 7 or newer, click the Start icon in the lower-left corner of your screen, enter `IDLE` in the search box, and select **IDLE (Python GUI)**.
- On Windows XP, click the **Start** button and then select **Programs ▸ Python 3.4 ▸ IDLE (Python GUI)**.
- On Mac OS X, open the Finder window, click **Applications**, click **Python 3.4**, 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

No matter which operating system you're running, the IDLE window that first appears should be mostly blank except for text that looks something like this:

```
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64
bit (AMD64)] on win32Type "copyright", "credits" or "license()" for more
information.
>>>
```

This window is called the *interactive shell*. A shell is a program that lets you type instructions into the computer, much like the Terminal or Command Prompt on OS X 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.

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:

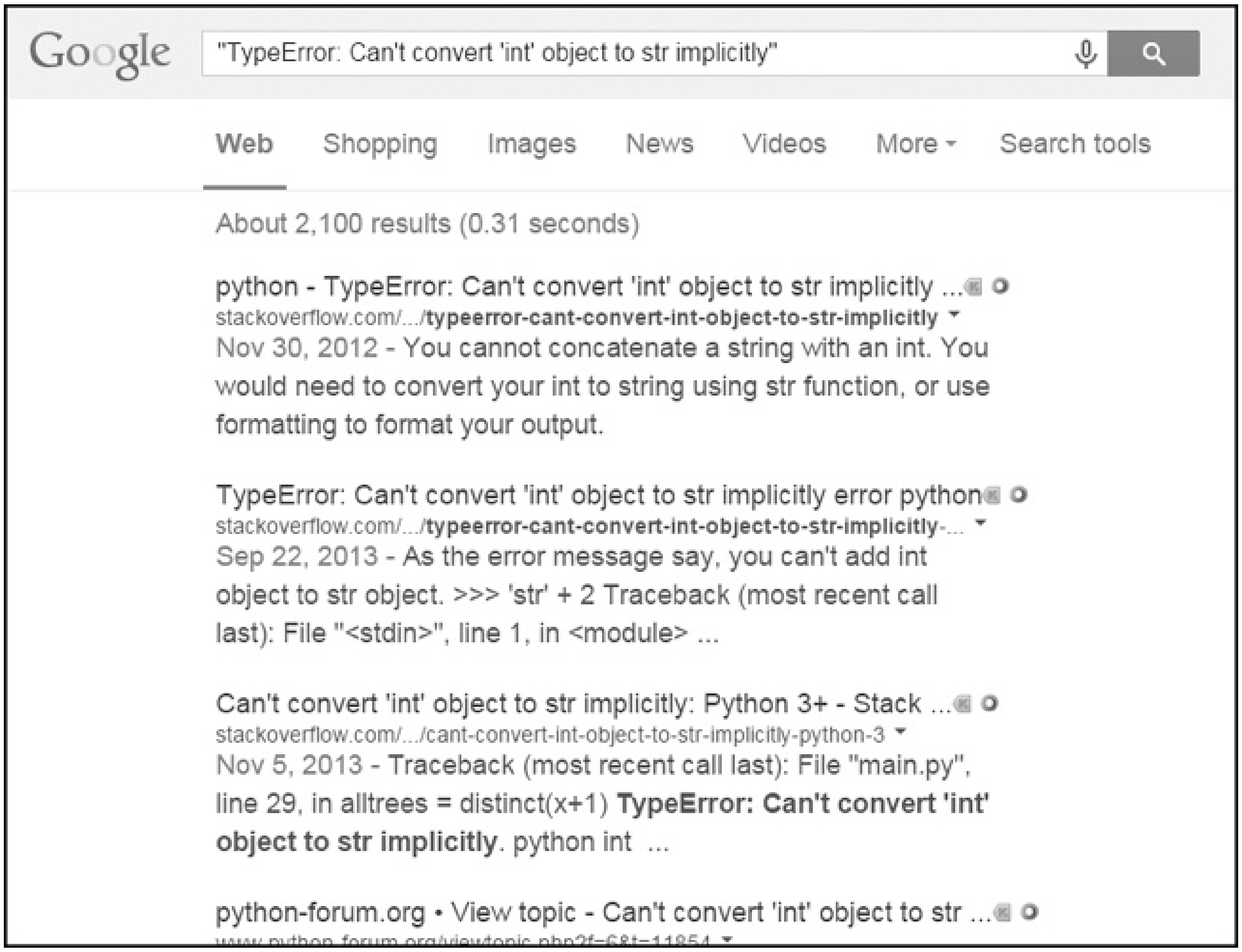
```
>>> print('Hello world!')
Hello world!
```

How to Find Help

Solving programming problems on your own is easier than you might think. If you're not convinced, then 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 appeared here 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 online for the exact error message. 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 2.



About 2,100 results (0.31 seconds)

python - TypeError: Can't convert 'int' object to str implicitly ...
stackoverflow.com/.../**typeerror-cant-convert-int-object-to-str-implicitly**
Nov 30, 2012 - You cannot concatenate a string with an int. You would need to convert your int to string using str function, or use formatting to format your output.

TypeError: Can't convert 'int' object to str implicitly error python
stackoverflow.com/.../**typeerror-cant-convert-int-object-to-str-implicitly-**...
Sep 22, 2013 - As the error message say, you can't add int object to str object. >>> 'str' + 2 Traceback (most recent call last): File "<stdin>", line 1, in <module> ...

Can't convert 'int' object to str implicitly: Python 3+ - Stack ...
stackoverflow.com/.../cant-convert-int-object-to-str-implicitly-python-3
Nov 5, 2013 - Traceback (most recent call last): File "main.py", line 29, in alltrees = distinct(x+1) **TypeError: Can't convert 'int' object to str implicitly**. python int ...

python-forum.org • View topic - Can't convert 'int' object to str ...

Figure 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 Overlow (*http://stackoverflow.com/*) or the "learn programming" subreddit at *http://reddit.com/r/learnprogramming/*. But keep in mind there are smart ways to ask programming questions that help others help you. Be sure to read the Frequently Asked Questions sections these websites have 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 *http://pastebin.com/* or *http://gist.github.com/*.

  These websites make it easy to share large amounts of code with people over the Web, without the risk of 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: *http://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. Explain what those actions are, if so.

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.

Summary

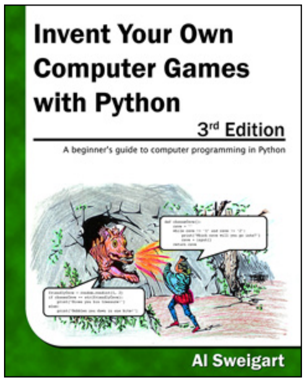
For most people, their computer is just an appliance instead of a tool. But by learning how to program, you'll gain access to one of the most powerful tools of the modern world, and you'll have fun along the way. Programming isn't brain surgery—it's fine for amateurs to experiment and make mistakes.

I love helping people discover Python. I write programming tutorials on my blog at *http://inventwithpython.com/blog/*, and you can contact me with questions at *al@inventwithpython.com*.

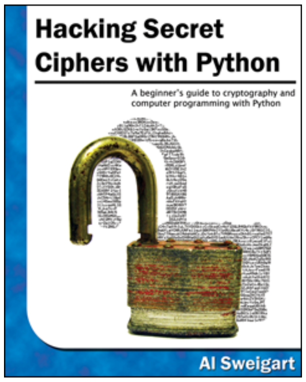
This book will start you off from zero programming knowledge, but you may have questions beyond its scope. Remember that asking effective questions and knowing how to find answers are invaluable tools on your programming journey.

Let's begin!

ToC →


Invent Your Own Computer Games with Python
3rd Edition
A beginner's guide to computer programming in Python
Al Sweigart

Making Games with Python & Pygame
A guide to programming with graphics, animation, and sound
Al Sweigart


Hacking Secret Ciphers with Python
A beginner's guide to cryptography and computer programming with Python
Al Sweigart