CRACKING CODES WITH PYTHON

AN INTRODUCTION TO
BUILDING AND *BREAKING CIPHERS*

AL SWEIGART

no starch press

CRACKING CODES WITH PYTHON

CRACKING CODES WITH PYTHON

An Introduction to Building and Breaking Ciphers

by Al Sweigart

no starch press

San Francisco

CRACKING CODES WITH PYTHON. Copyright © 2018 by Al Sweigart.

Some rights reserved. This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 United States License. To view a copy of this license, visit *http://creativecommons.org/licenses/by-nc-sa/3.0/us/* or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

ISBN-10: 1-59327-822-5
ISBN-13: 978-1-59327-822-9

Publisher: William Pollock
Production Editor: Riley Hoffman
Cover Illustration: Josh Ellingson
Interior Design: Octopod Studios
Developmental Editors: Jan Cash and Annie Choi
Technical Reviewers: Ari Lacenski and Jean-Philippe Aumasson
Copyeditor: Anne Marie Walker
Compositors: Riley Hoffman and Meg Sneeringer
Proofreader: Paula L. Fleming

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

Library of Congress Cataloging-in-Publication Data

```
Names: Sweigart, Al, author.
Title: Cracking codes with Python : an introduction to building and breaking
   ciphers / Al Sweigart.
Description: San Francisco : No Starch Press,Inc., [2018]
Identifiers: LCCN 2017035704 (print) | LCCN 2017047589 (ebook) | ISBN
   9781593278694 (epub) | ISBN 1593278691 (epub) | ISBN 9781593278229 (pbk.)
   | ISBN 1593278225 (pbk.)
Subjects: LCSH: Data encryption (Computer science) | Python (Computer program
   language) | Computer security. | Hacking.
Classification: LCC QA76.9.A25 (ebook) | LCC QA76.9.A25 S9317 2018 (print) |
   DDC 005.8/7--dc23
LC record available at https://lccn.loc.gov/2017035704
```

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

Dedicated to Aaron Swartz, 1986–2013

"Aaron was part of an army of citizens that believes democracy only works when the citizenry are informed, when we know about our rights—and our obligations. An army that believes we must make justice and knowledge available to all—not just the well born or those that have grabbed the reins of power—so that we may govern ourselves more wisely. When I see our army, I see Aaron Swartz and my heart is broken. We have truly lost one of our better angels."
—Carl Malamud

About the Author

Al Sweigart is a software developer and tech book author living in San Francisco. 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 weighs 12 pounds.

About the Technical Reviewers

Ari Lacenski creates mobile apps and Python software. She lives in Seattle.

Jean-Philippe Aumasson (Chapters 22–24) is Principal Research Engineer at Kudelski Security, Switzerland. He speaks regularly at information security conferences such as Black Hat, DEF CON, Troopers, and Infiltrate. He is the author of *Serious Cryptography* (No Starch Press, 2017).

BRIEF CONTENTS

Acknowledgments xix

Introduction xxi

Chapter 1: Making Paper Cryptography Tools. 1

Chapter 2: Programming in the Interactive Shell 11

Chapter 3: Strings and Writing Programs 21

Chapter 4: The Reverse Cipher 39

Chapter 5: The Caesar Cipher. 53

Chapter 6: Hacking the Caesar Cipher with Brute-Force 69

Chapter 7: Encrypting with the Transposition Cipher 77

Chapter 8: Decrypting with the Transposition Cipher 99

Chapter 9: Programming a Program to Test Your Program 113

Chapter 10: Encrypting and Decrypting Files. 127

Chapter 11: Detecting English Programmatically 141

Chapter 12: Hacking the Transposition Cipher 161

Chapter 13: A Modular Arithmetic Module for the Affine Cipher 171

Chapter 14: Programming the Affine Cipher 185

Chapter 15: Hacking the Affine Cipher 197

Chapter 16: Programming the Simple Substitution Cipher 207

Chapter 17: Hacking the Simple Substitution Cipher 221

Chapter 18: Programming the Vigenère Cipher 247

Chapter 19: Frequency Analysis 259

Chapter 20: Hacking the Vigenère Cipher 279

Chapter 21: The One-Time Pad Cipher 315
Chapter 22: Finding and Generating Prime Numbers 321
Chapter 23: Generating Keys for the Public Key Cipher 335
Chapter 24: Programming the Public Key Cipher 349
Appendix: Debugging Python Code 375
Index 381

CONTENTS IN DETAIL

ACKNOWLEDGMENTS **xix**

INTRODUCTION **xxi**

Who Should Read This Book? xxii
What's in This Book? xxiii
How to Use This Book xxiv
    Typing Source Code xxiv
    Checking for Typos xxv
    Coding Conventions in This Book xxv
    Online Resources xxv
Downloading and Installing Python xxv
    Windows Instructions xxvi
    macOS Instructions xxvi
    Ubuntu Instructions xxvi
Downloading pyperclip.py xxvi
Starting IDLE xxvii
Summary xxviii

1
MAKING PAPER CRYPTOGRAPHY TOOLS **1**

What Is Cryptography? 2
Codes vs. Ciphers 3
The Caesar Cipher 4
    The Cipher Wheel 4
    Encrypting with the Cipher Wheel 5
    Decrypting with the Cipher Wheel 6
    Encrypting and Decrypting with Arithmetic 7
Why Double Encryption Doesn't Work 8
Summary 8
Practice Questions 9

2
PROGRAMMING IN THE INTERACTIVE SHELL **11**

Some Simple Math Expressions 12
    Integers and Floating-Point Values 13
    Expressions 13
    Order of Operations 14
    Evaluating Expressions 14
Storing Values with Variables 15
    Overwriting Variables 17
    Variable Names 18
Summary 18
Practice Questions 19

3
STRINGS AND WRITING PROGRAMS 21
Working with Text Using String Values . . . 22
String Concatenation with the + Operator . . . 23
String Replication with the * Operator . . . 24
Getting Characters from Strings Using Indexes . . . 24
Printing Values with the print() Function . . . 27
Printing Escape Characters . . . 28
Quotes and Double Quotes . . . 29
Writing Programs in IDLE's File Editor . . . 30
Source Code for the "Hello, World!" Program . . . 31
Checking Your Source Code with the Online Diff Tool . . . 31
Using IDLE to Access Your Program Later . . . 32
Saving Your Program . . . 32
Running Your Program . . . 33
Opening the Programs You've Saved . . . 34
How the "Hello, World!" Program Works . . . 34
Comments . . . 34
Printing Directions to the User . . . 34
Taking a User's Input . . . 35
Ending the Program . . . 35
Summary . . . 36
Practice Questions . . . 37

4
THE REVERSE CIPHER 39
Source Code for the Reverse Cipher Program . . . 40
Sample Run of the Reverse Cipher Program . . . 40
Setting Up Comments and Variables . . . 41
Finding the Length of a String . . . 41
Introducing the while Loop . . . 42
The Boolean Data Type . . . 43
Comparison Operators . . . 43
Blocks . . . 45
The while Loop Statement . . . 46
"Growing" a String . . . 47
Improving the Program with an input() Prompt . . . 50
Summary . . . 50
Practice Questions . . . 51

5
THE CAESAR CIPHER 53
Source Code for the Caesar Cipher Program . . . 54
Sample Run of the Caesar Cipher Program . . . 55
Importing Modules and Setting Up Variables . . . 56
Constants and Variables . . . 57
The for Loop Statement . . . 58
An Example for Loop . . . 58
A while Loop Equivalent of a for Loop . . . 59

The if Statement . . . 59
    An Example if Statement . . . 60
    The else Statement . . . 60
    The elif Statement . . . 61
The in and not in Operators . . . 61
The find() String Method . . . 62
Encrypting and Decrypting Symbols . . . 63
    Handling Wraparound . . . 64
    Handling Symbols Outside of the Symbol Set . . . 65
Displaying and Copying the Translated String . . . 65
Encrypting Other Symbols . . . 66
Summary . . . 66
Practice Questions . . . 67

6
HACKING THE CAESAR CIPHER WITH BRUTE-FORCE 69

Source Code for the Caesar Cipher Hacker Program . . . 70
Sample Run of the Caesar Cipher Hacker Program . . . 71
Setting Up Variables . . . 72
Looping with the range() Function . . . 72
Decrypting the Message . . . 73
Using String Formatting to Display the Key and Decrypted Messages . . . 75
Summary . . . 76
Practice Question . . . 76

7
ENCRYPTING WITH THE TRANSPOSITION CIPHER 77

How the Transposition Cipher Works . . . 78
    Encrypting a Message by Hand . . . 79
    Creating the Encryption Program . . . 80
Source Code for the Transposition Cipher Encryption Program . . . 81
Sample Run of the Transposition Cipher Encryption Program . . . 82
Creating Your Own Functions with def Statements . . . 82
    Defining a Function that Takes Arguments with Parameters . . . 83
    Changes to Parameters Exist Only Inside the Function . . . 84
    Defining the main() Function . . . 85
Passing the Key and Message As Arguments . . . 86
The List Data Type . . . 86
    Reassigning the Items in Lists . . . 87
    Lists of Lists . . . 88
    Using len() and the in Operator with Lists . . . 89
    List Concatenation and Replication with the + and * Operators . . . 89
The Transposition Encryption Algorithm . . . 90
Augmented Assignment Operators . . . 91
Moving currentIndex Through the Message . . . 92
The join() String Method . . . 93
Return Values and return Statements . . . 94
    A return Statement Example . . . 94
    Returning the Encrypted Ciphertext . . . 95
The __name__ Variable . . . 95
Summary . . . 96
Practice Questions . . . 97

8
DECRYPTING WITH THE TRANSPOSITION CIPHER 99

How to Decrypt with the Transposition Cipher on Paper 100
Source Code for the Transposition Cipher Decryption Program 101
Sample Run of the Transposition Cipher Decryption Program 102
Importing Modules and Setting Up the main() Function 102
Decrypting the Message with the Key 103
    The round(), math.ceil(), and math.floor() Functions 103
    The decryptMessage() Function 104
    Boolean Operators 106
    Adjusting the column and row Variables 109
Calling the main() Function 110
Summary 110
Practice Questions 111

9
PROGRAMMING A PROGRAM TO TEST YOUR PROGRAM 113

Source Code for the Transposition Cipher Tester Program 114
Sample Run of the Transposition Cipher Tester Program 115
Importing the Modules 116
Creating Pseudorandom Numbers 116
Creating a Random String 118
    Duplicating a String a Random Number of Times 118
    List Variables Use References 119
    Passing References 121
    Using copy.deepcopy() to Duplicate a List 122
    The random.shuffle() Function 122
    Randomly Scrambling a String 123
Testing Each Message 123
Checking Whether the Cipher Worked and Ending the Program 124
Calling the main() Function 124
Testing the Test Program 125
Summary 125
Practice Questions 126

10
ENCRYPTING AND DECRYPTING FILES 127

Plain Text Files 128
Source Code for the Transposition File Cipher Program 128
Sample Run of the Transposition File Cipher Program 130
Working with Files 130
    Opening Files 131
    Writing to and Closing Files 131
    Reading from a File 132
Setting Up the main() Function 132
Checking Whether a File Exists 133
    The os.path.exists() Function 133
    Checking Whether the Input File Exists with the os.path.exists() Function 134
Using String Methods to Make User Input More Flexible 134
    The upper(), lower(), and title() String Methods 134
    The startswith() and endswith() String Methods 135
    Using These String Methods in the Program 135

Reading the Input File 136
Measuring the Time It Took to Encrypt or Decrypt 136
    The time Module and time.time() Function 136
    Using the time.time() Function in the Program 137
Writing the Output File 137
Calling the main() Function 138
Summary 138
Practice Questions 139

11
DETECTING ENGLISH PROGRAMMATICALLY 141

How Can a Computer Understand English? 142
Source Code for the Detect English Module 143
Sample Run of the Detect English Module 145
Instructions and Setting Up Constants 145
The Dictionary Data Type 146
    The Difference Between Dictionaries and Lists 147
    Adding or Changing Items in a Dictionary 147
    Using the len() Function with Dictionaries 148
    Using the in Operator with Dictionaries 148
    Finding Items Is Faster with Dictionaries than with Lists 149
    Using for Loops with Dictionaries 149
Implementing the Dictionary File 150
    The split() Method 150
    Splitting the Dictionary File into Individual Words 151
    Returning the Dictionary Data 151
Counting the Number of English Words in message 152
    Divide-by-Zero Errors 152
    Counting the English Word Matches 153
    The float(), int(), and str() Functions and Integer Division 154
    Finding the Ratio of English Words in the Message 154
Removing Non-Letter Characters 155
    The append() List Method 155
    Creating a String of Letters 156
Detecting English Words 156
    Using Default Arguments 157
    Calculating Percentages 157
Summary 159
Practice Questions 160

12
HACKING THE TRANSPOSITION CIPHER 161

Source Code of the Transposition Cipher Hacker Program 162
Sample Run of the Transposition Cipher Hacker Program 163
Importing the Modules 164
Multiline Strings with Triple Quotes 164
Displaying the Results of Hacking the Message 165
Getting the Hacked Message 166
    The strip() String Method 167
    Applying the strip() String Method 168
    Failing to Hack the Message 168

Calling the main() Function 169
Summary 169
Practice Questions 169

13
A MODULAR ARITHMETIC MODULE FOR THE AFFINE CIPHER 171

Modular Arithmetic. 172
The Modulo Operator. 173
Finding Factors to Calculate the Greatest Common Divisor 173
Multiple Assignment 175
Euclid's Algorithm for Finding the GCD. 176
Understanding How the Multiplicative and Affine Ciphers Work 177
Choosing Valid Multiplicative Keys 178
Encrypting with the Affine Cipher 179
Decrypting with the Affine Cipher 179
Finding Modular Inverses 181
The Integer Division Operator 181
Source Code for the Cryptomath Module 182
Summary 183
Practice Questions 183

14
PROGRAMMING THE AFFINE CIPHER 185

Source Code for the Affine Cipher Program. 186
Sample Run of the Affine Cipher Program 188
Setting Up Modules, Constants, and the main() Function 188
Calculating and Validating the Keys. 189
The Tuple Data Type 190
Checking for Weak Keys 190
How Many Keys Can the Affine Cipher Have? 191
Writing the Encryption Function 193
Writing the Decryption Function. 194
Generating Random Keys 195
Calling the main() Function 196
Summary 196
Practice Questions 196

15
HACKING THE AFFINE CIPHER 197

Source Code for the Affine Cipher Hacker Program 198
Sample Run of the Affine Cipher Hacker Program 199
Setting Up Modules, Constants, and the main() Function 200
The Affine Cipher Hacking Function 201
The Exponent Operator 201
Calculating the Total Number of Possible Keys 201
The continue Statement 202
Using continue to Skip Code 203
Calling the main() Function 204
Summary 205
Practice Questions 205

16
PROGRAMMING THE SIMPLE SUBSTITUTION CIPHER 207

How the Simple Substitution Cipher Works 208
Source Code for the Simple Substitution Cipher Program 209
Sample Run of the Simple Substitution Cipher Program 210
Setting Up Modules, Constants, and the main() Function 211
The sort() List Method 212
Wrapper Functions. 213
The translateMessage() Function. 215
    The isupper() and islower() String Methods 216
    Preserving Cases with isupper(). 217
Generating a Random Key 218
Calling the main() Function 219
Summary 219
Practice Questions 219

17
HACKING THE SIMPLE SUBSTITUTION CIPHER 221

Using Word Patterns to Decrypt. 222
    Finding Word Patterns. 222
    Finding Potential Decryption Letters 223
Overview of the Hacking Process. 225
The Word Pattern Modules 225
Source Code for the Simple Substitution Hacking Program 226
Sample Run of the Simple Substitution Hacking Program 229
Setting Up Modules and Constants. 230
Finding Characters with Regular Expressions. 230
Setting Up the main() Function 231
Displaying Hacking Results to the User 232
Creating a Cipherletter Mapping 232
    Creating a Blank Mapping. 232
    Adding Letters to a Mapping 233
    Intersecting Two Mappings. 234
    How the Letter-Mapping Helper Functions Work 235
    Identifying Solved Letters in Mappings 238
    Testing the removeSolvedLetterFromMapping() Function 240
The hackSimpleSub() Function 241
    The replace() String Method 243
    Decrypting the Message. 243
    Decrypting in the Interactive Shell 244
Calling the main() Function 245
Summary 246
Practice Questions 246

18
PROGRAMMING THE VIGENÈRE CIPHER 247

Using Multiple Letter Keys in the Vigenère Cipher. 248
    Longer Vigenère Keys Are More Secure. 249
    Choosing a Key That Prevents Dictionary Attacks 250
Source Code for the Vigenère Cipher Program 251
Sample Run of the Vigenère Cipher Program. 252

Setting Up Modules, Constants, and the main() Function 252
Building Strings with the List-Append-Join Process 253
Encrypting and Decrypting the Message 255
Calling the main() Function 257
Summary 257
Practice Questions 258

19
FREQUENCY ANALYSIS 259

Analyzing the Frequency of Letters in Text 260
Matching Letter Frequencies 262
Calculating the Frequency Match Score for the Simple Substitution Cipher . . . 262
Calculating the Frequency Match Score for the Transposition Cipher 263
Using Frequency Analysis on the Vigenère Cipher. 264
Source Code for Matching Letter Frequencies 265
Storing the Letters in ETAOIN Order. 266
Counting the Letters in a Message 267
Getting the First Member of a Tuple 268
Ordering the Letters in the Message by Frequency 268
Counting the Letters with getLetterCount() 269
Creating a Dictionary of Frequency Counts and Letter Lists 269
Sorting the Letter Lists in Reverse ETAOIN Order 270
Sorting the Dictionary Lists by Frequency 274
Creating a List of the Sorted Letters 276
Calculating the Frequency Match Score of the Message 276
Summary 277
Practice Questions 278

20
HACKING THE VIGENÈRE CIPHER 279

Using a Dictionary Attack to Brute-Force the Vigenère Cipher 280
Source Code for the Vigenère Dictionary Hacker Program 280
Sample Run of the Vigenère Dictionary Hacker Program 281
About the Vigenère Dictionary Hacker Program 281
Using Kasiski Examination to Find the Key's Length 282
Finding Repeated Sequences 282
Getting Factors of Spacings 283
Getting Every Nth Letters from a String. 284
Using Frequency Analysis to Break Each Subkey. 285
Brute-Forcing Through the Possible Keys. 287
Source Code for the Vigenère Hacking Program 287
Sample Run of the Vigenère Hacking Program. 293
Importing Modules and Setting Up the main() Function 294
Finding Repeated Sequences. 294
Calculating the Factors of the Spacings. 297
Removing Duplicates with the set() Function 298
Removing Duplicate Factors and Sorting the List 298
Finding the Most Common Factors 298
Finding the Most Likely Key Lengths 300
The extend() List Method 301
Extending the repeatedSeqSpacings Dictionary 301
Getting the Factors from factorsByCount. 302

Getting Letters Encrypted with the Same Subkey 302
Attempting Decryption with a Likely Key Length 303
    The end Keyword Argument for print() 306
    Running the Program in Silent Mode or Printing Information to the User 306
    Finding Possible Combinations of Subkeys 306
    Printing the Decrypted Text with the Correct Casing 310
Returning the Hacked Message 311
    Breaking Out of the Loop When a Potential Key Is Found 311
    Brute-Forcing All Other Key Lengths 312
Calling the main() Function 313
Modifying the Constants of the Hacking Program 313
Summary 314
Practice Questions 314

21
THE ONE-TIME PAD CIPHER 315

The Unbreakable One-Time Pad Cipher 316
    Making Key Length Equal Message Length 316
    Making the Key Truly Random 318
    Avoiding the Two-Time Pad 319
    Why the Two-Time Pad Is the Vigenère Cipher 319
Summary 320
Practice Questions 320

22
FINDING AND GENERATING PRIME NUMBERS 321

What Is a Prime Number? 322
Source Code for the Prime Numbers Module 324
Sample Run of the Prime Numbers Module 326
How the Trial Division Algorithm Works 326
Implementing the Trial Division Algorithm Test 328
The Sieve of Eratosthenes 328
Generating Prime Numbers with the Sieve of Eratosthenes 330
The Rabin-Miller Primality Algorithm 331
Finding Large Prime Numbers 332
Generating Large Prime Numbers 333
Summary 334
Practice Questions 334

23
GENERATING KEYS FOR THE PUBLIC KEY CIPHER 335

Public Key Cryptography 336
The Problem with Authentication 337
    Digital Signatures 338
    Beware the MITM Attack 339
Steps for Generating Public and Private Keys 340
Source Code for the Public Key Generation Program 340
Sample Run of the Public Key Generation Program 342
Creating the main() Function 343

Generating Keys with the generateKey() Function 343
    Calculating an e Value 344
    Calculating a d Value 344
    Returning the Keys 345
Creating Key Files with the makeKeyFiles() Function 345
Calling the main() Function 347
Hybrid Cryptosystems 347
Summary 348
Practice Questions 348

24
PROGRAMMING THE PUBLIC KEY CIPHER 349

How the Public Key Cipher Works 350
    Creating Blocks 350
    Converting a String into a Block 350
    The Mathematics of Public Key Cipher Encryption and Decryption 353
    Converting a Block to a String 354
    Why We Can't Hack the Public Key Cipher 355
Source Code for the Public Key Cipher Program 357
Sample Run of the Public Key Cipher Program 360
Setting Up the Program 362
How the Program Determines Whether to Encrypt or Decrypt 362
Converting Strings to Blocks with getBlocksFromText() 363
    The min() and max() Functions 364
    Storing Blocks in blockInt 364
Using getTextFromBlocks() to Decrypt 366
    Using the insert() List Method 367
    Merging the Message List into One String 367
Writing the encryptMessage() Function 367
Writing the decryptMessage() Function 368
Reading in the Public and Private Keys from Their Key Files 369
Writing the Encryption to a File 369
Decrypting from a File 371
Calling the main() Function 373
Summary 373

APPENDIX
DEBUGGING PYTHON CODE 375

How the Debugger Works 375
Debugging the Reverse Cipher Program 377
Setting Breakpoints 379
Summary 380

INDEX 381

ACKNOWLEDGMENTS

This book would not have been possible without the exceptional work of the No Starch Press team. Thanks to my publisher, Bill Pollock; thanks to my editors, Riley Hoffman, Jan Cash, Annie Choi, Anne Marie Walker, and Laurel Chun, for their incredible help throughout the process; thanks to my technical editor, Ari Lacenski, for her help in this edition and back when it was just a stack of printouts I showed her at Shotwell's; thanks to JP Aumasson for lending his expertise in the public key chapters; and thanks to Josh Ellingson for a great cover.

INTRODUCTION

"I couldn't help but overhear,
probably because I was eavesdropping."
—Anonymous

If you could travel back to the early 1990s with this book, the contents of Chapter 23 that implement part of the RSA cipher would be illegal to export out of the United States. Because messages encrypted with RSA are impossible to hack, the export of encryption software like RSA was deemed a matter of national security and required State Department approval. In fact, strong cryptography was regulated at the same level as tanks, missiles, and flamethrowers.

In 1990, Daniel J. Bernstein, a student at the University of California, Berkeley, wanted to publish an academic paper that featured source code of his Snuffle encryption system. The US government informed him that he would need to become a licensed arms dealer before he could post his source code on the internet. The government also told him that it would deny him an export license if he applied for one because his technology was too secure.

The Electronic Frontier Foundation, a young digital civil liberties organization, represented Bernstein in *Bernstein v. United States*. For the first time ever, the courts ruled that written software code was speech protected by the First Amendment and that the export control laws on encryption violated Bernstein's First Amendment rights.

Now, strong cryptography is at the foundation of a large part of the global economy, safeguarding businesses and e-commerce sites used by millions of internet shoppers every day. The intelligence community's predictions that encryption software would become a grave national security threat were unfounded.

But as recently as the 1990s, spreading this knowledge freely (as this book does) would have landed you in prison for arms trafficking. For a more detailed history of the legal battle for freedom of cryptography, read Steven Levy's book *Crypto: How the Code Rebels Beat the Government, Saving Privacy in the Digital Age* (Penguin, 2001).

Who Should Read This Book?

Many books teach beginners how to write secret messages using ciphers. A couple of books teach beginners how to hack ciphers. But no books teach beginners how to program computers to hack ciphers. This book fills that gap.

This book is for those who are curious about encryption, hacking, or cryptography. The ciphers in this book (except for the public key cipher in Chapters 23 and 24) are all centuries old, but any laptop has the computational power to hack them. No modern organizations or individuals use these ciphers anymore, but by learning them, you'll learn the foundations cryptography was built on and how hackers can break weak encryption.

NOTE *The ciphers you'll learn in this book are fun to play with, but they don't provide true security. Don't use any of the encryption programs in this book to secure your actual files. As a general rule, you shouldn't trust the ciphers that you create. Real-world ciphers are subject to years of analysis by professional cryptographers before being put into use.*

This book is also for people who have never programmed before. It teaches basic programming concepts using the Python programming language, which is one of the best languages for beginners. It has a gentle learning curve that novices of all ages can master, yet it's also a powerful language used by professional software developers. Python runs on Windows, macOS, Linux, and even the Raspberry Pi, and it's free to download and use. (See "Downloading and Installing Python" on page xxv for instructions.)

In this book, I'll use the term *hacker* often. The word has two definitions. A hacker can be a person who studies a system (such as the rules of a cipher or a piece of software) to understand it so well that they're not limited by that system's original rules and can modify it in creative ways.

A hacker can also be a criminal who breaks into computer systems, violates people's privacy, and causes damage. This book uses the term in the first sense. Hackers are cool. Criminals are just people who think they're being clever by breaking stuff.

What's in This Book?

The first few chapters introduce basic Python and cryptography concepts. Thereafter, chapters generally alternate between explaining a program for a cipher and then explaining a program that hacks that cipher. Each chapter also includes practice questions to help you review what you've learned.

- **Chapter 1: Making Paper Cryptography Tools** covers some simple paper tools, showing how encryption was done before computers.
- **Chapter 2: Programming in the Interactive Shell** explains how to use Python's interactive shell to play around with code one line at a time.
- **Chapter 3: Strings and Writing Programs** covers writing full programs and introduces the string data type used in all programs in this book.
- **Chapter 4: The Reverse Cipher** explains how to write a simple program for your first cipher.
- **Chapter 5: The Caesar Cipher** covers a basic cipher first invented thousands of years ago.
- **Chapter 6: Hacking the Caesar Cipher with Brute-Force** explains the brute-force hacking technique and how to use it to decrypt messages without the encryption key.
- **Chapter 7: Encrypting with the Transposition Cipher** introduces the transposition cipher and a program that encrypts messages with it.
- **Chapter 8: Decrypting with the Transposition Cipher** covers the second half of the transposition cipher: being able to decrypt messages with a key.
- **Chapter 9: Programming a Program to Test Your Program** introduces the programming technique of testing programs with other programs.
- **Chapter 10: Encrypting and Decrypting Files** explains how to write programs that read files from and write files to the hard drive.
- **Chapter 11: Detecting English Programmatically** describes how to make the computer detect English sentences.
- **Chapter 12: Hacking the Transposition Cipher** combines the concepts from previous chapters to hack the transposition cipher.
- **Chapter 13: A Modular Arithmetic Module for the Affine Cipher** explains the math concepts behind the affine cipher.
- **Chapter 14: Programming the Affine Cipher** covers writing an affine cipher encryption program.
- **Chapter 15: Hacking the Affine Cipher** explains how to write a program to hack the affine cipher.

- **Chapter 16: Programming the Simple Substitution Cipher** covers writing a simple substitution cipher encryption program.
- **Chapter 17: Hacking the Simple Substitution Cipher** explains how to write a program to hack the simple substitution cipher.
- **Chapter 18: Programming the Vigenère Cipher** explains a program for the Vigenère cipher, a more complex substitution cipher.
- **Chapter 19: Frequency Analysis** explores the structure of English words and how to use it to hack the Vigenère cipher.
- **Chapter 20: Hacking the Vigenère Cipher** covers a program for hacking the Vigenère cipher.
- **Chapter 21: The One-Time Pad Cipher** explains the one-time pad cipher and why it's mathematically impossible to hack.
- **Chapter 22: Finding and Generating Prime Numbers** covers how to write a program that quickly determines whether a number is prime.
- **Chapter 23: Generating Keys for the Public Key Cipher** describes public key cryptography and how to write a program that generates public and private keys.
- **Chapter 24: Programming the Public Key Cipher** explains how to write a program for a public key cipher, which you can't hack using a mere laptop.
- The appendix, **Debugging Python Code**, shows you how to use IDLE's debugger to find and fix bugs in your programs.

How to Use This Book

Cracking Codes with Python is different from other programming books because it focuses on the source code of complete programs. Instead of teaching you programming concepts and leaving it up to you to figure out how to make your own programs, this book shows you complete programs and explains how they work.

In general, you should read the chapters in this book in order. The programming concepts build on those in the previous chapters. However, Python is such a readable language that after the first few chapters, you can probably jump ahead to later chapters and piece together what the code does. If you jump ahead and feel lost, return to earlier chapters.

Typing Source Code

As you read through this book, I encourage you to *manually type the source code from this book into Python*. Doing so will definitely help you understand the code better.

When typing the source code, don't include the line numbers that appear at the beginning of each line. These numbers are not part of the actual programs, and we use them only to refer to specific lines in the code. But aside from the line numbers, be sure to enter the code exactly as it appears, including the uppercase and lowercase letters.

You'll also notice that some of the lines don't begin at the leftmost edge of the page but are indented by four, eight, or more spaces. Be sure to enter the correct number of spaces at the beginning of each line to avoid errors.

But if you would rather not type the code, you can download the source code files from this book's website at *https://www.nostarch.com/crackingcodes/*.

Checking for Typos

Although manually entering the source code for the programs is helpful for learning Python, you might occasionally make typos that cause errors. These typos can be difficult to spot, especially when your source code is very long.

To quickly and easily check for mistakes in your typed source code, you can copy and paste the text into the online diff tool on the book's website at *https://www.nostarch.com/crackingcodes/*. The diff tool shows any differences between the source code in the book and yours.

Coding Conventions in This Book

This book is not designed to be a reference manual; it's a hands-on guide for beginners. For this reason, the coding style sometimes goes against best practices, but that's a conscious decision to make the code easier to learn. This book also skips theoretical computer science concepts.

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.

Online Resources

This book's website (*https://www.nostarch.com/crackingcodes/*) includes many useful resources, including downloadable files of the programs and sample solutions to the practice questions. This book covers classical ciphers thoroughly, but because there is always more to learn, I've also included suggestions for further reading on many of the topics introduced in this book.

Downloading and Installing Python

Before you can begin programming, you'll need to install the *Python interpreter,* which is software that executes the instructions you'll write in the Python language. I'll refer to "the Python interpreter" as "Python" from now on.

Download Python for Windows, macOS, and Ubuntu for free from *https://www.python.org/downloads/*. If you download the latest version, all of the programs in this book should work.

NOTE *Be sure to download a version of Python 3 (such as 3.6). The programs in this book are written to run on Python 3 and may not run correctly, if at all, on Python 2.*

Windows Instructions

On Windows, download the Python installer, which should have a filename ending with *.msi,* and double-click it. Follow the instructions the installer displays on the screen to install Python, as listed here:

1. Select **Install Now** to begin the installation.
2. When the installation is finished, click **Close**.

macOS Instructions

On macOS, download the *.dmg* file for your version of macOS from the website 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 your computer's administrator password.
2. Click **Continue** through the Welcome section and click **Agree** to accept the license.
3. Select **HD Macintosh** (or the name of your hard drive) and click **Install**.

Ubuntu Instructions

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

1. Open the Ubuntu Software Center.
2. Type `Python` in the search box in the top-right corner of the window.
3. Select **IDLE (using Python 3.6)**, or whatever is the latest version.
4. Click **Install**.

You may have to enter the administrator password to complete the installation.

Downloading pyperclip.py

Almost every program in this book uses a custom module I wrote called *pyperclip.py*. This module provides functions that let your programs copy and paste text to the clipboard. It doesn't come with Python, so you'll need to download it from *https://www.nostarch.com/crackingcodes/*.

This file must be in the same folder (also called *directory*) as the Python program files you write. Otherwise you'll see the following error message when you try to run your programs:

```
ImportError: No module named pyperclip
```

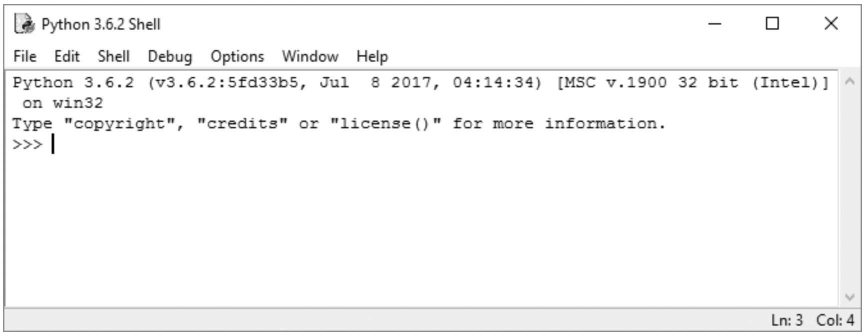
Now that you've downloaded and installed the Python interpreter and the *pyperclip.py* module, let's look at where you'll be writing your programs.

Starting IDLE

While the Python interpreter is the software that runs your Python programs, the *interactive development environment (IDLE)* software is where you'll write your programs, much like a word processor. IDLE is installed when you install Python. To start IDLE, follow these steps:

- 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 3.6 64-bit)**.
- On macOS, open Finder, click **Applications**, click **Python 3.6**, 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**.)

No matter which operating system you're running, the IDLE window should look something like Figure 1. The header text may be slightly different depending on your specific version of Python.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
```

Figure 1: The IDLE window

This window is called the *interactive shell*. A shell is a program that lets you type instructions into the computer, much like the Terminal on macOS or the Windows Command Prompt. Sometimes you'll want to run short snippets of code instead of writing a full program. Python's interactive shell lets you enter instructions for the Python interpreter software, which the computer reads and runs immediately.

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

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

Press ENTER, and the interactive shell should display this in response:

```
Hello, world!
```

Summary

Before the introduction of computers ushered in modern cryptography, breaking many codes was impossible using just pencil and paper. Although computing made many of the old, classical ciphers vulnerable to attack, they're still fun to learn about. Writing cryptanalysis programs that crack these ciphers is a great way to learn how to program.

In Chapter 1, we'll start with some basic cryptography tools to encrypt and decrypt messages without the aid of computers.

Let's get hacking.