Cracking Codes with Python

An Introduction to Building and Breaking Ciphers

by Al Sweigart

Errata updated to print 9

Page	Error	Correction	Print corrected
xxi	Because messages encrypted with RSA are **impossible** to hack, . . .	Because messages encrypted with RSA are near **near impossible** to hack, . . .	Print 9
xxvi	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.**	If you're running Ubuntu, **Python is already installed, but you may have to install IDLE by following these steps:** **1. Open a Terminal window by pressing Ctrl-Shift-T.** **2. Run `sudo apt-get install idle3` (you will need the administrator password).**	Print 7

xxvi	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`
		It doesn't come with Python, so you'll need to **install it by entering the following in the interactive shell:**
		`>>> import subprocess, sys;`
		`subprocess.run([sys.executable, '-m', 'pip', 'install', 'pyperclip'])`
		If you have trouble installing pyperclip, consult *https://pypi.org/project/pyperclip/*. Meanwhile, you can replace any lines of code that contain `pyperclip.copy()` or `pyperclip.paste()` with the `pass` Python keyword.
			Print 7

xxvii	• 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.)**	• 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 3.11 64-bit)**. • On macOS, **open Spotlight by pressing COMMAND-spacebar and entering IDLE.** • On Ubuntu, **press the Win key and search for idle. Click the IDLE (using Python 3.10) item.**	Print 7
26	Notice that the expression `'Hello, world!'[7:13][2]` first evaluates ~~the list slice~~ to `'world!'[2]` and then further evaluates to `'r'`.	Notice that the expression `'Hello, world!'[7:13][2]` first evaluates to `'world!'[2]` and then further evaluates to `'r'`.	Print 9
57	Just as in the reverse cipher in Chapter 5, . . .	Just as in the reverse cipher in Chapter 4, . . .	Print 9

Page	Error	Correction	Print corrected
84	(All the variables in the reverse cipher and Caesar cipher programs in Chapters **5** and **6**, respectively, were global.)	(All the variables in the reverse cipher and Caesar cipher programs in Chapters **4** and **5**, respectively, were global.)	Print 9
96	. . . than the Caesar cipher program in Chapter **6**.	. . . than the Caesar cipher program in Chapter **5**.	Print 9
104	. . . and `math.ceil()` will instead **increment the ones place of the float and convert it to an integer to round up**.	. . . and `math.ceil()` will instead **round up any nonzero fractional part and convert it to an integer**.	Print 9
123	Although the key for the Caesar cipher could be an integer from 0 to 65 (the **length** of the symbol set), . . .	Although the key for the Caesar cipher could be an integer from 0 to 65 (the **range** of the symbol set), . . .	Print 9
123	The `for` loop on line 23 runs the test code with the keys `1` up to (but not including) the length of the message.	The `for` loop on line 23 runs the test code with the keys `1` up to (but not including) the length of the message **divided by two**.	
162, 163, 166, 195, 198, 199, 201	*The code and text repeatedly state that Python programs can be stopped at any time by pressing Ctrl-C (on Windows) or Ctrl-D (on macOS and Linux). In fact, all three operating systems use Ctrl-C.*		Print 9
175	In this example, the 8-square rod is the longest rod that can fit evenly into 24 and **32**.	In this example, the 8-square rod is the longest rod that can fit evenly into 24 and **16**.	Print 5
191	The condition on line 35 checks whether `keyA` is a negative number (that is, whether it's less than 0) *or* whether `keyB` is **greater** than 0 *or* **less** than the size of the symbol set minus one:	The condition on line 35 checks whether `keyA` is a negative number (that is, whether it's less than 0) *or* whether `keyB` is **less** than 0 *or* **greater** than the size of the symbol set minus one:	Print 9
202	Line 34 stores the tuple's first integer in `keyA` by placing the `[0]` after the **`hackAffine()`** function call.	Line 34 stores the tuple's first integer in `keyA` by placing the `[0]` after the **`getKeyParts()`** function call.	Print 9
209	`if keyIsValid(myKey):`	`if not keyIsValid(myKey):`	Print 2
212	`if keyIsValid(myKey):`	`if not keyIsValid(myKey):`	Print 2
236	~~`letterMapping1 =`~~ `simpleSubHacker.addLettersToMapping(letterMapping1, 'OLQIHXIRCKGNZ', candidates[0])`	`simpleSubHacker.addLettersToMapping(letterMapping1, 'OLQIHXIRCKGNZ', candidates[0])`	Print 7
237	~~`letterMapping1 =`~~ `simpleSubHacker.addLettersToMapping(letterMapping2, 'PLQRZKBZB', candidate)`	`simpleSubHacker.addLettersToMapping(letterMapping2, 'PLQRZKBZB', candidate)`	Print 7

Page	Error	Correction	Print corrected
237	~~`letterMapping2 =`~~ `simpleSubHacker.addLettersToMapping(letterMapping2, 'PLQRZKBZB', candidate)`	`simpleSubHacker.addLettersToMapping(letterMapping2, 'PLQRZKBZB', candidate)`	Print 7
238	~~`letterMapping3 =`~~ `simpleSubHacker.addLettersToMapping(letterMapping3, 'MPBKSSIPLC', candidates[i])`	`simpleSubHacker.addLettersToMapping(letterMapping3, 'MPBKSSIPLC', candidates[i])`	Print 7
243	To decrypt our message, we'll use the function `simpleSubstitutionCipher.decryptMessage()` that we already programmed in *simpleSubstitutionCipher.py*. But `simpleSubstitutionCipher.decryptMessage()` decrypts using keys only, not letter mappings, so we can't use the function directly. To address this issue, we'll create a `decryptWithCipherletterMapping()` function that takes a letter mapping, converts the mapping into a key, and then passes the key and message to `simpleSubstitutionCipher.decryptMessage()`.	To decrypt our message, we'll use the function `simpleSubCipher.decryptMessage()` that we already programmed in *simpleSubCipher.py*. But `simpleSubCipher.decryptMessage()` decrypts using keys only, not letter mappings, so we can't use the function directly. To address this issue, we'll create a `decryptWithCipherletterMapping()` function that takes a letter mapping, converts the mapping into a key, and then passes the key and message to `simpleSubCipher.decryptMessage()`.	Print 9
250	There are 95,428,956,661,682,176 possible twelve-letter keys, but there are only about 1800 twelve-letter words in our dictionary file.	There are 95,428,956,661,682,176 possible twelve-letter keys, but there are only about 1,800 twelve-letter words in our dictionary file.	Print 7
264	. . . 26 + 26 + 26 + 26 + 26, or **156**, decryptions for a five-letter key.	. . . 26 + 26 + 26 + 26 + 26, or **130**, decryptions for a five-letter key.	Print 9
281	`for word in lines:`	`for word in words:`	Print 2
282	His method was later published by Friedrich Kasiski, **an** ~~**early**~~ 20th-century mathematician . . .	His method was later published by Friedrich Kasiski, **a** 20th-century mathematician . . .	Print 9
305	After sorting the tuples in `freqScores` in reverse order, line 176 appends a list containing only the first **three** tuples, or the tuples with the **three** highest English frequency match scores, to `allFreqScores`.	After sorting the tuples in `freqScores` in reverse order, line 176 appends a list containing only the first **four** tuples, or the tuples with the **four** highest English frequency match scores, to `allFreqScores`.	Print 9
325	83. 84. `# See if any of the low prime numbers can divide num:` 85. `for prime in LOW_PRIMES:` 86. `if (num % prime == 0):` 87. `return False` 88.	83. `# See if any of the low prime numbers can divide num:` 84. `for prime in LOW_PRIMES:` 85. `if (num == prime):` 86. `return True` 87. `if (num % prime == 0):` 88. `return False`	Print 2

Page	Error	Correction	Print corrected

333	Line 85 loops through each of the prime numbers in the LOW_PRIMES list:
	
	`84.  # See if any of the low prime numbers can divide num:`
	`85.  for prime in LOW_PRIMES:`
	`86.  if (num % prime == 0):`
	`87.  return False`
	
	The integer in `num` is modded by each prime number using the mod operator on line 86, and if the result evaluates to `0`, we know that `prime` divides `num` so `num` is not prime. In that case, line 87 returns `False`.
	Those are the **two** quick tests we'll perform to determine whether a number is prime. If the execution continues past line 87, the `rabinMiller()` function checks `num`'s primality.
		Line 84 loops through each of the prime numbers in the LOW_PRIMES list:
		
		`83.  # See if any of the low prime numbers can divide num:`
		`84.  for prime in LOW_PRIMES:`
		`85.  if (num == prime):`
		`86.  return True`
		`87.  if (num % prime == 0):`
		`88.  return False`
		
		If the integer in `num` is the same as `prime`, then obviously `num` must be a prime number and line 86 returns `True`.
		The integer in `num` is modded by each prime number using the mod operator on line 87, and if the result evaluates to `0`, we know that `prime` divides `num` so `num` is not prime. In that case, line 88 returns `False`.
		Those are the **three** quick tests we'll perform to determine whether a number is prime. If the execution continues past line 88, the `rabinMiller()` function checks `num`'s primality.
			Print 2

341, 345	`45. def makeKeyFiles(name, keySize):`
	`46.  # Creates two files 'x_pubkey.txt' and 'x_privkey.txt' (where x`
	`47.  # is the value in name) with the n,e and d,e integers written in`
	`48.  # them, delimited by a comma.`
		`45. def makeKeyFiles(name, keySize):`
		`46.  # Creates two files 'x_pubkey.txt' and 'x_privkey.txt' (where x`
		`47.  # is the value in name) with the n,e and n,d integers written in`
		`48.  # them, delimited by a comma.`
			Print 9

341, 347	`64. print('The private key is a %s and a %s digit number.' %` `(len(str(publicKey[0])), len(str(publicKey[1]))))`	`64. print('The private key is a %s and a %s digit number.' %` `(len(str(privateKey[0])), len(str(privateKey[1]))))`	Print 9
363	Let's look at how the program converts a message string into **128-byte** blocks.	Let's look at how the program converts a message string into **169-character** blocks.	Print 9