|  |  |
| --- | --- |
|  | Chapter 14 |
| Caesar Cipher |

Topics Covered In This Chapter:

* Cryptography and ciphers
* Encrypting and decrypting
* Ciphertext, plaintext, keys, and symbols
* The Caesar Cipher
* ASCII ordinal values
* The chr() and ord() functions
* The isalpha() string method
* The isupper() and islower() string methods
* Cryptanalysis
* The brute force technique

The program in this chapter isn’t really a game, but it is a fun program. The program will convert normal English into a secret code. It can also convert secret codes back into regular English again. Only someone who is knowledgeable about secret codes will be able to understand our secret messages.

Because this program manipulates text to convert it into secret messages, you will learn several new functions and methods for manipulating strings. You will also learn how programs can do math with text strings just as it can with numbers.

## Cryptography

The science of writing secret codes is called cryptography. For thousands of years cryptography has made secret messages that only the sender and recipient could read, even if someone captured the messenger and read the coded message. A secret code system is called a cipher. The cipher used by the program in this chapter is called the Caesar cipher.

In cryptography, we call the message that we want to be secret the plaintext. The plaintext could look like this:

Hello there! The keys to the house are hidden under the flower pot.

Converting the plaintext into the encoded message is called encrypting the plaintext. The plaintext is encrypted into the ciphertext. The ciphertext looks like random letters, and we cannot understand what the original plaintext was just by looking at the ciphertext. Here is the previous example encrypted into ciphertext:

Yvccf kyviv! Kyv bvpj kf kyv yfljv riv yzuuve leuvi kyv wcfnvi gfk.

But if you know about the cipher used to encrypt the message, you can decrypt the ciphertext back to the plaintext. (Decryption is the opposite of encryption.)

Many ciphers also use keys. Keys are secret values that let you decrypt ciphertext that was encrypted using a specific cipher. Think of the cipher as being like a door lock. You can only unlock it with a particular key.

If you are interested in writing cryptography programs, you can read my other book, “Hacking Secret Ciphers with Python”. It is free to download from http://inventwithpython.com/hacking.

## The Caesar Cipher

The key for the Caesar Cipher will be a number from 1 to 26. Unless you know the key (that is, know the number used to encrypt the message), you won’t be able to decrypt the secret code.

The Caesar Cipher was one of the earliest ciphers ever invented. In this cipher, you encrypt a message by taking each letter in the message (in cryptography, these letters are called symbols because they can be letters, numbers, or any other sign) and replacing it with a “shifted” letter. If you shift the letter A by one space, you get the letter B. If you shift the letter A by two spaces, you get the letter C. Figure 14-1 is a picture of some letters shifted over by three spaces.

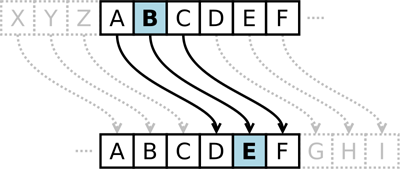


Figure 14-1: Shifting over letters by three spaces. Here, B becomes E.

To get each shifted letter, draw out a row of boxes with each letter of the alphabet. Then draw a second row of boxes under it, but start a certain number (this number is the key) of spaces over. After the letters at the end, wrap around back to the start of the boxes. Here is an example with the letters shifted by three spaces:

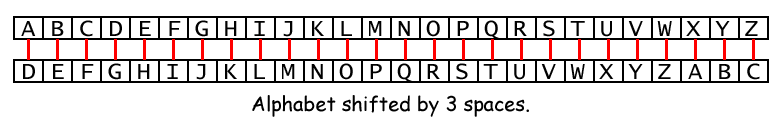


Figure 14-2: The entire alphabet shifted by three spaces.

The number of spaces you shift is the key in the Caesar Cipher. The example above shows the letter translations for the key 3.

If you encrypt the plaintext “Howdy” with a key of 3, then:

* The “H” becomes “K”.
* The letter “o” becomes “r”.
* The letter “w” becomes “z”.
* The letter “d” becomes “g”.
* The letter “y” becomes “b”.

The ciphertext of “Hello” with key 3 becomes “Krzgb”.

We will keep any non-letter characters the same. To decrypt “Krzgb” with the key 3, we go from the bottom boxes back to the top:

* The letter “K” becomes “H”.
* The letter “r” becomes “o”.
* The letter “z” becomes “w”.
* The letter “g” becomes “d”.
* The letter “b” becomes “y”.

## ASCII, and Using Numbers for Letters

How do we implement this shifting of the letters as code? We can do this by representing each letter as a number called an ordinal, and then adding or subtracting from this number to form a new ordinal (and a new letter). ASCII (pronounced “ask-ee” and stands for American Standard Code for Information Interchange) is a code that connects each character to a number between 32 and 126.

The capital letters “A” through “Z” have the ASCII numbers 65 through 90. The lowercase letters “a” through “z” have the ASCII numbers 97 through 122. The numeric digits “0” through “9” have the ASCII numbers 48 through 57. Table 14-1 shows all the ASCII characters and ordinals.

Modern computers use UTF-8 instead of ASCII. But UTF-8 is backwards compatible with ASCII, so the UTF-8 ordinals for ASCII characters are the same as ASCII’s ordinals.

Table 14-1: The ASCII Table

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 32 | (space) | 48 | 0 | 64 | @ | 80 | P | 96 | ` | 112 | p |
| 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 36 | $ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 39 | ' | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 40 | ( | 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 41 | ) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 42 | \* | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 43 | + | 59 | ; | 75 | K | 91 | [ | 107 | k | 123 | { |
| 44 | , | 60 | < | 76 | L | 92 | \ | 108 | l | 124 | | |
| 45 | - | 61 | = | 77 | M | 93 | ] | 109 | m | 125 | } |
| 46 | . | 62 | > | 78 | N | 94 | ^ | 110 | n | 126 | ~ |
| 47 | / | 63 | ? | 79 | O | 95 | \_ | 111 | o |  |  |

So if you wanted to shift “A” by three spaces, you would do the following:

* Convert “A” to an ordinal (65).
* Add 3 to 65, to get 68.
* Convert the ordinal 68 back to a letter (“D”).

The chr() and ord() functions can convert between characters and ordinals.

## The chr() and ord() Functions

The chr() function (pronounced “char”, short for “character”) takes an integer ordinal and returns a single-character string. The ord() function (short for “ordinal”) takes a single-character string, and returns the integer ordinal value. Try entering the following into the interactive shell:

>>> chr(65)

'A'

>>> ord('A')

65

>>> chr(65+8)

'I'

>>> chr(52)

'4'

>>> chr(ord('F'))

'F'

>>> ord(chr(68))

68

On the third line, chr(65+8) evaluates to chr(73). If you look at the ASCII table, you can see that 73 is the ordinal for the capital letter “I”.

On the fifth line, chr(ord('F')) evaluates to chr(70) which evaluates to 'F'. The ord() and chr() functions are the opposite of each other.

## Sample Run of Caesar Cipher

Here is a sample run of the Caesar Cipher program, encrypting a message:

Do you wish to encrypt or decrypt a message?

encrypt

Enter your message:

The sky above the port was the color of television, tuned to a dead channel.

Enter the key number (1-26)

13

Your translated text is:

Gur fxl nobir gur cbeg jnf gur pbybe bs gryrivfvba, gharq gb n qrnq punaary.

Now run the program and decrypt the text that you just encrypted.

Do you wish to encrypt or decrypt a message?

decrypt

Enter your message:

Gur fxl nobir gur cbeg jnf gur pbybe bs gryrivfvba, gharq gb n qrnq punaary.

Enter the key number (1-26)

13

Your translated text is:

The sky above the port was the color of television, tuned to a dead channel.

If you do not decrypt with the correct key, the decrypted text will be garbage data:

Do you wish to encrypt or decrypt a message?

decrypt

Enter your message:

Gur fxl nobir gur cbeg jnf gur pbybe bs gryrivfvba, gharq gb n qrnq punaary.

Enter the key number (1-26)

15

Your translated text is:

Rfc qiw yzmtc rfc nmpr uyq rfc amjmp md rcjctgqgml, rslcb rm y bcyb afyllcj.

## Source Code of Caesar Cipher

Here is the source code for the Caesar Cipher program. After you type this code in, save the file as cipher.py. If you get errors after typing this code in, compare the code you typed to the book’s code with the online diff tool at http://invpy.com/diff/cipher.

caesar.py

1. # Caesar Cipher

2.

3. MAX\_KEY\_SIZE = 26

4.

5. def getMode():

6. while True:

7. print('Do you wish to encrypt or decrypt a message?')

8. mode = input().lower()

9. if mode in 'encrypt e decrypt d'.split():

10. return mode

11. else:

12. print('Enter either "encrypt" or "e" or "decrypt" or "d".')

13.

14. def getMessage():

15. print('Enter your message:')

16. return input()

17.

18. def getKey():

19. key = 0

20. while True:

21. print('Enter the key number (1-%s)' % (MAX\_KEY\_SIZE))

22. key = int(input())

23. if (key >= 1 and key <= MAX\_KEY\_SIZE):

24. return key

25.

26. def getTranslatedMessage(mode, message, key):

27. if mode[0] == 'd':

28. key = -key

29. translated = ''

30.

31. for symbol in message:

32. if symbol.isalpha():

33. num = ord(symbol)

34. num += key

35.

36. if symbol.isupper():

37. if num > ord('Z'):

38. num -= 26

39. elif num < ord('A'):

40. num += 26

41. elif symbol.islower():

42. if num > ord('z'):

43. num -= 26

44. elif num < ord('a'):

45. num += 26

46.

47. translated += chr(num)

48. else:

49. translated += symbol

50. return translated

51.

52. mode = getMode()

53. message = getMessage()

54. key = getKey()

55.

56. print('Your translated text is:')

57. print(getTranslatedMessage(mode, message, key))

## How the Code Works

The encryption and decryption processes are the reverse of the other, and even then they still share much of the same code. Let’s look at how each line works.

1. # Caesar Cipher  
 2.

3. MAX\_KEY\_SIZE = 26

The first line is just a comment. MAX\_KEY\_SIZE is a constant that stores the integer 26 in it. MAX\_KEY\_SIZE reminds us that in this program, the key used in the cipher should be between 1 and 26.

Deciding to Encrypt or Decrypt

5. def getMode():

6. while True:

7. print('Do you wish to encrypt or decrypt a message?')

8. mode = input().lower()

9. if mode in 'encrypt e decrypt d'.split():

10. return mode

11. else:

12. print('Enter either "encrypt" or "e" or "decrypt" or "d".')

The getMode() function will let the user type in if they want encryption or decryption mode for the program. The value returned from input() and lower() is stored in mode. The if statement’s condition checks if the string stored in mode exists in the list returned by 'encrypt e decrypt d'.split().

This list is ['encrypt', 'e', 'decrypt', 'd'], but it is easier for the programmer to type 'encrypt e decrypt d'.split() and not type in all those quotes and commas. Use whichever is easiest for you; they both evaluate to the same list value.

This function will return the string in mode as long as mode is equal to 'encrypt', 'e', 'decrypt', or 'd'. Therefore, getMode() will return the string 'e' or the string 'd' (but the user can type in either “e”, “encrypt”, “d”, or “decrypt”.)

Getting the Message from the Player

14. def getMessage():

15. print('Enter your message:')

16. return input()

The getMessage() function simply gets the message to encrypt or decrypt from the user and returns it.

Getting the Key from the Player

18. def getKey():

19. key = 0

20. while True:

21. print('Enter the key number (1-%s)' % (MAX\_KEY\_SIZE))

22. key = int(input())

23. if (key >= 1 and key <= MAX\_KEY\_SIZE):

24. return key

The getKey() function lets the player type in the key they will use to encrypt or decrypt the message. The while loop ensures that the function keeps looping until the user enters a valid key.

A valid key here is one that is between the integer values 1 and 26 (remember that MAX\_KEY\_SIZE will only ever have the value 26 because it is constant). It then returns this key. Line 22 sets key to the integer version of what the user typed in, so getKey() returns an integer.

Encrypt or Decrypt the Message with the Given Key

26. def getTranslatedMessage(mode, message, key):

27. if mode[0] == 'd':

28. key = -key

29. translated = ''

getTranslatedMessage() does the encrypting and decrypting. It has three parameters:

* mode sets the function to encryption mode or decryption mode.
* message is the plaintext (or ciphertext) to be encrypted (or decrypted).
* key is the key that is used in this cipher.

Line 27 checks if the first letter in the mode variable is the string 'd'. If so, then the program is in decryption mode. The only difference between the decryption and encryption mode is that in decryption mode the key is set to the negative version of itself. If key was the integer 22, then in decryption mode set it to -22. The reason why will be explained later.

translated is the string of the result: either the ciphertext (if you are encrypting) or the plaintext (if you are decrypting). It starts as the blank string and has encrypted or decrypted characters concatenated to the end of it.

## The isalpha() String Method

The isalpha() string method will return True if the string is an uppercase or lowercase letter from A to Z. If the string contains any non-letter characters, then isalpha() will return False. Try entering the following into the interactive shell:

>>> 'Hello'.isalpha()

True

>>> 'Forty two'.isalpha()

False

>>> 'Fortytwo'.isalpha()

True

>>> '42'.isalpha()

False

>>> ''.isalpha()

False

As you can see, 'Forty two'.isalpha() will return False because 'Forty two' has a space in it, which is a non-letter character. 'Fortytwo'.isalpha() returns True because it doesn’t have this space.

'42'.isalpha() returns False because both '4' and '2' are non-letter characters. isalpha() only returns True if the string has only letter characters and isn’t blank.

The isalpha() method is used in the next few lines of the program.

31. for symbol in message:

32. if symbol.isalpha():

33. num = ord(symbol)

34. num += key

Line 31’s for loop iterates over each letter (in cryptography they are called symbols) in the message string. On each iteration through this loop, symbol will have the value of a letter in message.

Line 32 is there because only letters will be encrypted or decrypted. Numbers, punctuation marks, and everything else will stay in their original form. The num variable will hold the integer ordinal value of the letter stored in symbol. Line 34 then “shifts” the value in num by the value in key.

## The isupper() and islower() String Methods

The isupper() and islower() string methods (which are on line 36 and 41) work in a way that is similar to the isdigit() and isalpha() methods.

isupper() will return True if the string it is called on contains at least one uppercase letter and no lowercase letters. islower() returns True if the string it is called on contains at least one lowercase letter and no uppercase letters. Otherwise these methods return False.

Try entering the following into the interactive shell:

>>> 'HELLO'.isupper()

True

>>> 'hello'.isupper()

False

>>> 'hello'.islower()

True

>>> 'Hello'.islower()

False

>>> 'LOOK OUT BEHIND YOU!'.isupper()

True

>>> '42'.isupper()

False

>>> '42'.islower()

False

>>> ''.isupper()

False

>>> ''.islower()

False

Encrypting or Decrypting Each Letter

36. if symbol.isupper():

37. if num > ord('Z'):

38. num -= 26

39. elif num < ord('A'):

40. num += 26

Line 36 checks if the symbol is an uppercase letter. If so, there are two special cases to worry about. What if symbol was 'Z' and key was 4? If that were the case, the value of num here would be the character '^' (The ordinal of '^' is 94). But ^ isn’t a letter at all. You want the ciphertext to “wrap around” to the beginning of the alphabet.

Check if num has a value larger than the ordinal value for “Z”. If so, then **subtract** 26 (because there are 26 letters in total) from num. After doing this, the value of num is 68. 68 is the correct ordinal value for 'D'.

41. elif symbol.islower():

42. if num > ord('z'):

43. num -= 26

44. elif num < ord('a'):

45. num += 26

If the symbol is a lowercase letter, the program runs code that is similar to lines 36 through 40. The only difference is that it uses ord('z') and ord('a') instead of ord('Z') and ord('A').

In decrypting mode, then key would be negative. The special case would be where num -= 26 is less than the ASCII value for “a”. In that case, **add** 26 to num to have it “wrap around” to the end of the alphabet.

47. translated += chr(num)

48. else:

49. translated += symbol

Line 47 concatenates the encrypted/decrypted character to the translated string.

If the symbol was not an uppercase or lowercase letter, then line 48 concatenates the original symbol to the translated string. Therefore, spaces, numbers, punctuation marks, and other characters won’t be encrypted or decrypted.

50. return translated

The last line in the getTranslatedMessage() function returns the translated string.

The Start of the Program

52. mode = getMode()

53. message = getMessage()

54. key = getKey()

55. print('Your translated text is:')

56. print(getTranslatedMessage(mode, message, key))

The start of the program calls each of the three functions defined previously to get the mode, message, and key from the user. These three values are passed to getTranslatedMessage() whose return value (the translated string) is printed to the user.

## Brute Force

That’s the entire Caesar Cipher. However, while this cipher may fool some people who don’t understand cryptography, it won’t keep a message secret from someone who knows cryptanalysis. While cryptography is the science of making codes, cryptanalysis is the science of breaking codes.

Do you wish to encrypt or decrypt a message?

encrypt

Enter your message:

Doubts may not be pleasant, but certainty is absurd.

Enter the key number (1-26)

8

Your translated text is:

Lwcjba uig vwb jm xtmiaivb, jcb kmzbiqvbg qa ijaczl.

The whole point of cryptography is that so if someone else gets their hands on the encrypted message, they cannot figure out the original unencrypted message from it. Let’s pretend we are the code breaker and all we have is the encrypted text:

Lwcjba uig vwb jm xtmiaivb, jcb kmzbiqvbg qa ijaczl.

Brute force is the technique of trying every possible key until you find the correct one. Because there are only 26 possible keys, it would be easy for a cryptanalyst to write a hacking program than decrypts with every possible key. Then they could look for the key that decrypts to plain English. Let’s add a brute force feature to the program.

Adding the Brute Force Mode

First, change lines 7, 9, and 12 (which are in the getMode() function) to look like the following (the changes are in bold):

5. def getMode():

6. while True:

7. print('Do you wish to encrypt or decrypt **or brute force** a message?')

8. mode = input().lower()

9. if mode in 'encrypt e decrypt d **brute b**'.split():

10. return mode[0]

11. else:

12. print('Enter either "encrypt" or "e" or "decrypt" or "d" **or "brute" or "b"**.')

This code will let the user select “brute force” as a mode. Modify and add the following changes to the main part of the program:

52. mode = getMode()

53. message = getMessage()

54. **if mode[0] != 'b':**

55. key = getKey()

56.

57. print('Your translated text is:')

58. **if mode[0] != 'b':**

59. print(getTranslatedMessage(mode, message, key))

60. **else:**

61. **for key in range(1, MAX\_KEY\_SIZE + 1):**

62. **print(key, getTranslatedMessage('decrypt', message, key))**

These changes ask the user for a key if they are not in “brute force” mode. The original getTranslatedMessage() call is made and the translated string is printed.

However, if the user is in “brute force” mode then getTranslatedMessage() loop that iterates from 1 all the way up to MAX\_KEY\_SIZE (which is 26). Remember that when the range() function returns a list of integers up to, but not including, the second parameter, which is why you have + 1. This program will print every possible translation of the message (including the key number used in the translation). Here is a sample run of this modified program:

Do you wish to encrypt or decrypt or brute force a message?

brute

Enter your message:

Lwcjba uig vwb jm xtmiaivb, jcb kmzbiqvbg qa ijaczl.

Your translated text is:

1 Kvbiaz thf uva il wslhzhua, iba jlyahpuaf pz hizbyk.

2 Juahzy sge tuz hk vrkgygtz, haz ikxzgotze oy ghyaxj.

3 Itzgyx rfd sty gj uqjfxfsy, gzy hjwyfnsyd nx fgxzwi.

4 Hsyfxw qec rsx fi tpiewerx, fyx givxemrxc mw efwyvh.

5 Grxewv pdb qrw eh sohdvdqw, exw fhuwdlqwb lv devxug.

6 Fqwdvu oca pqv dg rngcucpv, dwv egtvckpva ku cduwtf.

7 Epvcut nbz opu cf qmfbtbou, cvu dfsubjouz jt bctvse.

8 Doubts may not be pleasant, but certainty is absurd.

9 Cntasr lzx mns ad okdzrzms, ats bdqszhmsx hr zartqc.

10 Bmszrq kyw lmr zc njcyqylr, zsr acpryglrw gq yzqspb.

11 Alryqp jxv klq yb mibxpxkq, yrq zboqxfkqv fp xyproa.

12 Zkqxpo iwu jkp xa lhawowjp, xqp yanpwejpu eo wxoqnz.

13 Yjpwon hvt ijo wz kgzvnvio, wpo xzmovdiot dn vwnpmy.

14 Xiovnm gus hin vy jfyumuhn, von wylnuchns cm uvmolx.

15 Whnuml ftr ghm ux iextltgm, unm vxkmtbgmr bl tulnkw.

16 Vgmtlk esq fgl tw hdwsksfl, tml uwjlsaflq ak stkmjv.

17 Uflskj drp efk sv gcvrjrek, slk tvikrzekp zj rsjliu.

18 Tekrji cqo dej ru fbuqiqdj, rkj suhjqydjo yi qrikht.

19 Sdjqih bpn cdi qt eatphpci, qji rtgipxcin xh pqhjgs.

20 Rciphg aom bch ps dzsogobh, pih qsfhowbhm wg opgifr.

21 Qbhogf znl abg or cyrnfnag, ohg pregnvagl vf nofheq.

22 Pagnfe ymk zaf nq bxqmemzf, ngf oqdfmuzfk ue mnegdp.

23 Ozfmed xlj yze mp awpldlye, mfe npceltyej td lmdfco.

24 Nyeldc wki xyd lo zvokckxd, led mobdksxdi sc klcebn.

25 Mxdkcb vjh wxc kn yunjbjwc, kdc lnacjrwch rb jkbdam.

26 Lwcjba uig vwb jm xtmiaivb, jcb kmzbiqvbg qa ijaczl.

After looking over each row, you can see that the 8th message isn’t garbage, but plain English! The cryptanalyst can deduce that the original key for this encrypted text must have been 8. This brute force would have been difficult to do back in the days of Caesars and the Roman Empire, but today we have computers that can quickly go through millions or even billions of keys in a short time.

Summary

Computers are good at doing mathematics. When we create a system to translate some piece of information into numbers (such as we do with text and ordinals or with space and coordinate systems), computer programs can process these numbers quickly and efficiently.

But while our Caesar cipher program here can encrypt messages that will keep them secret from people who have to figure it out with pencil and paper, it won’t keep it secret from people who know how to get computers to process information for them. (Our brute force mode proves this.)

A large part of figuring out how to write a program is figuring out how to represent the information you want to manipulate as values that Python can understand.

The next chapter will present Reversi (also known as Othello). The AI that plays this game will be much more advanced than the AI that played Tic Tac Toe in chapter 9. In fact, the AI is so good most of the time you’ll be unable to beat it!