**Caesar’s cipher**

In cryptography, the original message is called the **plaintext** and the encrypted message is called the **ciphertext**.

One of the oldest and simplest ways to encrypt a message is to use a method called *Caesar’s cipher*, where **each letter in the message is replaced by the one that comes *k* positions after it in the alphabet**. It is known as *k* because it is a *key* ( you need to know *k* to ‘unlock’ the message and decrypt it).

Non-letter characters (spaces, numbers, symbols, etc.) can simply be copied over from the plaintext to the ciphertext.

**Example encryption: simple version**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| plaintext |  | i |  | l | o | v | e |  | c | o | m | p | u | t | i | n | g |
| key = +2 |  | ↓ |  | ↓ | ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| ciphertext |  | k |  | n | q | x | g |   | e | q | o | r | w | v | k | p | i |

This is easier if you also write down the position of each letter in the alphabet. Moving *k* letters down the alphabet is the same as adding the number *k* to a letter’s position.

**Example encryption: with the positions of the letters in the alphabet**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| plaintext |  | i |  | l | o | v | e |  | c | o | m | p | u | t | i | n | g |
| positions |  | 9 |  | 12 | 15 | 22 | 5 |  | 3 | 15 | 13 | 16 | 21 | 20 | 9 | 14 | 7 |
| key = +2 |  | ↓ |  | ↓ | ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| positions |  | 11 |  | 14 | 17 | 24 | 7 |  | 5 | 17 | 15 | 18 | 23 | 22 | 11 | 16 | 9 |
| ciphertext |  | k |  | n | q | x | g |   | e | q | o | r | w | v | k | p | i |

However, what happens when moving *k* letters down the alphabet takes you past the end of the alphabet? In that case, you can *circle* right back to the beginning of the alphabet, as you can see in the example below:

**Example encryption: some letters have been ‘shifted’ past the end of the alphabet**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| plaintext |  | i |  | l | o | v | e |  | c | o | m | p | u | t | i | n | g |
| positions |  | 9 |  | 12 | 15 | 22 | 5 |  | 3 | 15 | 13 | 16 | 21 | 20 | 9 | 14 | 7 |
| key = +10 |  | ↓ |  | ↓ | ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
|  |  |  |  |  |  | **32** |  |  |  |  |  |  | **31** | **30** |  |  |  |
| positions |  | 19 |  | 22 | 25 | 6 | 15 |  | 13 | 25 | 23 | 26 | 5 | 4 | 19 | 24 | 17 |
| ciphertext |  | s |  | v | y | f | o |   | m | y | w | z | e | d | s | x | q |

**If you get stuck with your Python try looking at the last page for some clues.**

 Task 1.

Create a Python program that prompts the user to enter the plaintext (the original message) and the key (the number of positions to shift letters by), and displays the ciphertext (the encrypted message).

|  |
| --- |
| **Example**  |
| Note: Use this example to check your program. This is the output your program should produce for the given input. |
| The program displays a prompt and waits for keyboard input | Enter the plaintext: |
| The user types in the original message | i love computing |
| The program displays a prompt and waits for keyboard input | Enter the key: |
| The user types in the number of positions to shift by | 2 |
| The program displays the encrypted message | k nqxg eqorwvkpi |

|  |
| --- |
| **Example**  |
| Note: Use this example to check your program. This is the output your program should produce for the given input. |
| The program displays a prompt and waits for keyboard input | Enter the plaintext: |
| The user types in the original message | i love computing |
| The program displays a prompt and waits for keyboard input | Enter the key: |
| The user types in the number of positions to shift by | 10 |
| The program displays the encrypted message | s vyfo mywzedsxq |

**Note:** For the purposes of this task, you can **assume** that only lower case letters need to be encrypted.

Start with this checklist for your program:

**Checklist**: Tick (✔) the corresponding box if your program:

|  |  |  |
| --- | --- | --- |
|

|  |
| --- |
|  |

 | Prompts the user for a **single character** to be encrypted (you can assume that only lower case letters need to be encrypted). |
|

|  |
| --- |
|  |

 | Prompts the user for the *key*, i.e. the number of positions that this character needs to be shifted by (you can assume that this is positive). |
|

|  |
| --- |
|  |

 | Computes and displays the **encrypted character**, i.e. the letter that comes *key* positions after it in the alphabet (use the examples provided to test your program). |
|

|  |
| --- |
|  |

 | Works correctly even for letters that the method circles back to the beginning of the alphabet (use the examples provided to test your program). |

Task 2 .

Now, **extend** your program to meet the criteria in the checklist below:

**Checklist**: Tick (✔) the corresponding box if your program:

|  |  |  |
| --- | --- | --- |
|

|  |
| --- |
|  |

 | Prompts the user for the **plaintext**, i.e. the message to be encrypted (you can assume that only lower case letters need to be encrypted). |
|

|  |
| --- |
|  |

 | Prompts the user for the *key*, i..e. the number of positions that each letter needs to be shifted by (you can assume that this is positive). |
|

|  |
| --- |
|  |

 | Computes and displays the **ciphertext**, where each letter of the original message has been replaced by the letter that comes *key* positions after it in the alphabet (use the examples provided to test your program). |
|

|  |
| --- |
|  |

 | Works correctly even for letters that the method circles back to the beginning of the alphabet (use the examples provided to test your program). |

 Testing the output.

You can test your program’s output using this website: <https://www.dcode.fr/caesar-cipher>

**Take a screenshot of your code that does both tasks and upload to Bourne To Learn to the earn your platinum badge.**

Clues . Look here if you need help

**What are the variables I will need?**

Think about the quantities you will need to refer to in your program, i.e. the values that your program will need to keep track of.

You will probably need:

* The key and the plaintext entered by the user
* Each individual character in the plaintext
* The ciphertext produced by the program

You will need additional variables as well, that you will find in the code that follows (e.g. letters, position, and cipherlist).

**How do I iterate over the characters in the plaintext?**

Read the plaintext as a piece of text. Use a for-loop to iterate over each character in the plaintext. The pseudocode below illustrates the idea:

|  |  |
| --- | --- |
|  | for character in plaintext: process the character |

**How can I know the position of a letter in the alphabet?**

Start with this string of letters in the beginning of your program:

|  |  |
| --- | --- |
|  | letters = 'abcdefghijklmnopqrstuvwxyz' |

This is essentially the alphabet. For any given character, as long as it belongs to these letters, you can get its numerical position in the alphabet using this code:

|  |  |
| --- | --- |
|  | position = letters.index(character) |

So, to give a couple of examples, position would be 0 for 'a' and 10 for 'k'.

**Note:** This code will result in a **ValueError** if the character is not a letter. So, make sure your program only executes this code when it is appropriate.

**How can I find the letter that comes after *k* positions in the alphabet?**

If you know the numerical position of a letter in the alphabet, you can add the key to this position and retrieve the corresponding character back from the letters. You will need to figure out what to do when adding the key to the position results in a new position that’s greater than 25.

**What do I do with the characters in the ciphertext**

For each character in the plaintext, your program will produce the corresponding ciphertext character. Collect all these characters into a list, say cipherlist. Don’t forget to initialise the cipherlist.

At the end of your program, use the code below to join all the characters you have collected in cipherlist into a single ciphertext string:

|  |  |
| --- | --- |
|  | ciphertext = "".join(cipherlist) |

This resource is available online at [ncce.io/prg5-5-a3-wc](http://ncce.io/prg5-5-a3-wc). Resources are updated regularly — please check that you are using the latest version.

This resource is licensed under the Open Government Licence, version 3. For more information on this licence, see [ncce.io/ogl](http://ncce.io/ogl).